Naval Ocean Systems Center
San Diego, CA 92152-5000

(2)

Technical Document 1853
July 1990

Literature Survey on Tools

C. G. Murphy

Science Applications International Corporation

ADMINISTRATIVE INFORMATION

This report was prepared by Science Applications International Corporation (SAIC), under contract N66001-87-D-0039 for Code 733 of the Naval Ocean Systems Center.

Released by
D. K. Barbour, Head
Signal Processing
Technology Branch

Under authority of
J. A. Roese, Head
Signal and Information
Processing Division

JJ

LITERATURE SURVEY OF PARALLEL PROCESSING TOOLS

TABLE OF CONTENTS

LIST OF APPENDICES

VOLUME I - EXECUTIVE SUMMARY

LITERATURE SURVEY OF PARALLEL PROCESSING TOOLS

LITERATURE SURVEY OF PARALLEL PROCESSING

VOLUME I. - EXECUTIVE SUMMARY

Section 1. Summary

1.1 Overview

This literature survey of parallel programming tools, including more than 200 references, is designed to allow continued browsing and probing of different specific areas of interest. Therefore, the first look at parallel programming tools is a broad one, where each reference article is briefly described. Key words (or phrases) were assigned and provided in sorts by author and by assignment to a hierarchy of high performance computing technology. This hierarchy ranges from algorithms through tools environments to architectures and technology. A table of commercial programming tools is also provided.

At the most focused level, a review is made of articles providing information which is important to support signal processing parallel tools.

A general discussion of the state-of-the-technology, and definitions and discussions of key terms, and concepts of parallel computing are also given.

1.2 Software State

Designing and building multiprocessors has proceeded at a dramatic pace; the development of effective ways to program them generally has not. Yet, software development is the most critical element in a system's design. The immense complexity of parallel computation can only increase our dependency on software.

1.3 Concurrency

The fundamental thesis of parallel computing is that concurrency can be expressed by users and operated upon by parallel computers to achieve significantly higher speed than in conventional computers. However, present peak-to-delivered performance for many applications is too large and too different for varying modes of concurrency.

1.4 Languages

A programming language cannot be general purpose if only a handful of experts grasp it and use it effectively. See Shapiro [LANG 4]. The future of parallel computing depends upon the creation of simple, yet effective parallel-programming models (reflected in appropriate language designs) that make the *detail of the underlying architecture transparent to the user.*

1.5 Mapping and Programming

There are three parts to matching algorithms to architectures:

Understanding the application and its concurrency
Selection of optimal architecture
Mapping the algorithm to the architecture

It is clear that these two parts must be continually iterated to remain at optimum performance.

Performance programmers on massively parallel machines must consider several new elements of machine balance. [Stone] These are:

Processor bandwidth - partition processes among processors
Memory bandwidth - access data in parallel
I/O bandwidth - rate must sustain full computational power
Communications bandwidth - move data between processes
Synchronization bandwidth - coordinate activity of processes
Multiple purposes - maintain flexibility for multiple parallel processing modes

1.6 Resource Contention

All parts of the parallel processor must operate at a reasonable design efficiency. This be expected by the user. The user, coding (specifying) at the concurrency level, cannot be required to anticipate that hidden pattern-related resource contention degradation can occur in his solution. Networks and memories should be conflict-free and support parallel synchronization through instruction level operation. Concurrent processes must be independent of the functional units and processors used to compute the results.

1.7 Portability

The need for concurrent languages, independent of the computer architecture, is evident from the literature survey.

In the parallel marketplace there is now very little to no portability between applications on parallel processors. Each requires mapping the application to the machine to achieve reasonable performance. Parallel extensions to sequential languages provided are not supported on other machines. Present language extensions are too assembly-like, and are very machine oriented and dependent.

1.8 Instrumentation and Performance Measurement

The user must be provided with highly visual tools to understand the operation of his algorithm on the machine. This will enable the user to express the concurrency in the problem in other forms to discover better solutions. The user may be mapping the concurrent algorithm to the machine by using this feedback but does so at a higher level. Other parallel processor designs may operate less effectively, but a reasonable level of operation could be expected without tuning.

1.9 Flexibility

Systems must be adequately robust to allow reprogramming without entire rewrites, while maintaining reasonable performance. Small changes should not severely impact the performance, and the user must be able to assume this; otherwise the maintenance of parallel computer codes will require that the developers be retained to constantly update them.

Section 2. Recommendations

The literature survey discovered no complete or machine-independent set of parallel processing tools for signal processing. Academia has made excellent starts on defining the elements of machine-independent programming environments, but implementations tend to be architecture (processing mode) directed toward MIMD, SIMD, Dataflow, or special purpose architecture. However, the survey results can now be analyzed to define the characteristics and possibly the specifications of a parallel programming environment for signal processing.

A wide diversity of architectures and associated approaches is evident from the survey. In addition, the rapid pace of change in architectures quickly renders obsolete many parts of tools. Use extreme care to ensure that machine-dependent material is not critical to the approach or results. However, this diversity leads to several common recommendations:

Abstraction at several levels is required
Object-oriented approaches are a common approach to abstraction
Layering of an environment's design is necessary to maintain long range robustness
Use concurrent or parallel functional language without sequential bias

These common elements were recognized as necessary to successful parallel programming tool environments:

Visual or graphical programming
Mapping into heterogeneous environments
Control and data partitioning tools
Debugging and performance measurement tools
Architectural modeling tools
Extensions to languages and adaptation of parallel abstract languages
Extensive libraries for each architecture with machine-independent common user interfaces
Sets of machine dependent tools for use by support specialist in building the general signal processing environment

The long range view of a parallel programming environment for signal processing includes the following organization:

Graphical Human Computer Interface
Rapid Prototyping Systems Environment
Scenario Driven Experiment Support
Algorithm analysis, Design, and Performance Evaluation
Parallel Architecture Modeling
Machine Independent Programming and Mapping tools
- performance monitor and debug
- language extensions
- operating system extensions
- library

Machine Dependent Mapping Tools for each architecture

Section 3. Conclusion

The challenge of the tool building effort will be maintaining machine independence when developing applications, algorithms, and tool environments. The rapidly changing architectures force a machine-independent approach in long range applications. In addition, new architecture requirements may result from algorithm advances exceeding advances in hardware speed.

Abstraction, layering, and architecture modeling and tracking are essential to maintain machine independence.

Object-oriented programming in a concurrent language (for example, Concurrent C++) is needed to ensure that systems designers are freed from building to the wrong architecture.

A carefully designed set of standards should be evolved which supports a layered tool environment, along the lines of the OSI/ISO communication standards.

Section 6 gives a list of near term tools which start the effort on each tool area. The tool categories are:

Visualization
Concurrency estimator
Standardization
General Aids
Abstraction
Architectural Model

VOLUME II - TECHNICAL

LITERATURE SURVEY OF PARALLEL PROCESSING TOOLS

LITERATURE SURVEY OF PARALLEL PROCESSING

VOLUME II. - TECHNICAL

Section 1. Overview

1.1 Scope and Objectives

This survey provides a software tools review for moderate to massive parallel processing and identifies opportunities to apply these tools to signal processing applications. The survey covers modern tools which have been reported in the literature during the past two years, including the full hierarchy of software development. It first sets the tools scene by drawing out brief ideas on tool, methods, and information. The survey pinpoints matches then exploring those in more detail.

1.1.1 Tool Hierarchy

The levels of the hierarchy include the following:

LEVEL	LABEL
Science and Mathematics	[Sci]
Numerical Methods	[Num]
General Algorithms	[Alg]
Libraries	[Lib]
General Programming Methods	[Prog]
Modeling, simulation, and Analysis tools	[Sim]
Application Mapping tools	[Map]
Human-computer interface	[HCI]
Parallel environment	[Env]
Support Utilities and Standards	[Sup]
User extensions to Languages	[ExtLang]
User extensions to Operating systems	[ExtOS]
Languages and compilers	[Lang]
Operating Systems	[OS]
Architectures	[Arch]
Technology	[Tech]

Labels in the square brackets "[...]" are identifying labels used to organize the material. The effort will concentrate on the middle of the hierarchy, but ideas from elements outside the control of the user are included in the hierarchy above, because these may lead to extensions and environment tools. Note that the survey labeled no references as [Sci] or [Num] because other labels were more appropriate when those topics were part of the references. For example, McBryan [PROG 22] discusses numerical methods but his overall work was better labeled in another topic (programming).

1.2 Survey Method

The goal of this survey is to identify the parallel programming tools which can assist in the difficult task of programming signal processing applications in parallel architectures. Therefore, the filter used in selection of materials is to select those which allow rapid prototyping, a supporting data base of test and evaluation inputs, architecture modeling, parallel algorithm development, and supporting graphical human interfaces, programmer mapping aids, language extensions, libraries, and machine mapping. The study also includes software engineering tools, concurrent algorithms and methods, vendor supplied programming languages and mapping tools for decomposition, synchronization, load balancing, and grain control, vendor supplied extensions, debugging and profiling tools, and operating systems. A letter was sent to parallel computer suppliers to receive their latest information.

The topics which are prime opportunities for impacting near-term needs for signal processing activities are covered in more detail. This concentration is, therefore, on those tools believed to be appropriate to moderately parallel (16 to 64 processors) and massively parallel (above 64 processors) and SIMD machines. Some reviews were included to give additional breadth.

1.3 Document Overview

Section 2 provides brief statements of some the key issues of parallel programming. Appendix A provides additional definitions of general parallel.

Section 3 provides a very brief review of selected journal articles from the general parallel survey. Section 4 provides a summary table of materials from commercial vendors of parallel computers. Section 5 contains short reviews of articles which could significantly affect the effort. Section 6 provides a framework for organizing an environment for parallel signal processing development through rapid prototyping, algorithm development, architecture modeling, and experimental evaluation. This environment is defined to be consistent with ideas and information gained through the literature survey.

Appendix A defines terms used in parallel processing. A list of references identified and filtered from the literature are provided in Appendix B. Appendix C provides key words and a sorted list by author. These lists give cross references of key words and an alphabetical list of authors to help identify references which are applicable to more than one level of the hierarchy.

Section 2. Parallel Processing Background

In this section, we discuss some of the issues which must be addressed in parallel programming [Murphy]. (References for general text discussions are given at the end of Appendix A.)

2.1 Software State

The technical challenge of parallel processing is to provide a massively parallel computer which can be effectively programmed from a language which refers to the concurrency in the problem, not the computer architecture. This abstraction must be possible with only a small loss from the peak speed for any concurrent expression. This requires the building of parallel processor and languages which run from a specification level language without mapping.

There are substantial gains to be realized from parallel processing but these require careful selection of the proper architecture to match the problem and changing of the skills of the scientists and engineers who perform the application. Matching highly structured applications makes possible a large expansion of problem space, and potentially leads to breakthroughs. Problems must have significant concurrency which matches available architectures.

Designing and building multiprocessors have proceeded at a dramatic pace, yet the development of effective ways to program them generally has not. Yet, software development is the most critical element in a system's design. The immense complexity of parallel computation can only increase our dependency on software.

The difference in the performances of well mapped (performance programmed) and abstract tool mapped (convenience programmed) applications is too large. The peak-to-delivered performance ratio drops rapidly if the application is not completely matched and mapped to the machine. Several efforts are underway to solve this problem.

2.2 Concurrency

Nature contains a tremendous amount of concurrency, but it is reduced by the steps involved in choosing physical models, algorithms, computational methods, programming, and computer language constraints, combined with hardware architecture. The fundamental thesis of parallel computing is that concurrency can be expressed by users and operated upon by parallel computers to achieve significantly higher speed than in conventional computers. However, present peak-to-delivered performance for many applications is too large and too different for varying modes of concurrency.

However, present parallel computer designs only support one or two modes of operation (forms of concurrency) because a single parallelism mode has been concentrated upon. For example, highly independent and homogeneous numerical applications may run well on Architecture 1 and poorly on Architecture 2, but the opposite performance rating occurs when the application is changed. The concern is not that this happens, but that the difference in performance between the two cases is very large. Peak-to-delivered performance for many applications is too large and too different for varying modes of concurrency.

2.3 Languages

A programming language cannot be general purpose if only a handful of experts grasp it and use it effectively. The future of parallel computing depends upon the creation of simple but effective parallel-programming models (reflected in appropriate language designs) that make the detail of the underlying architecture transparent to the user. See Shapiro [LANG 4].

Parallel architectures presently lack clear abstraction constraints, leaving the user to cope with problems of immense complexity at one time.

The safe approach is to use a layered hierarchy and structure to allow independent work on small portions of a problem. Separation of algorithm design from architectural considerations is ideal and several approaches for this are found in the literature survey. Conventional software engineering tools and techniques approach complex system applications in this manner. Abstraction constraints aid in vector and systolic programming and are candidates for concurrency investigations. There are other concurrent methods which require dividing a problem into cells or volumes corresponding to data or functional concurrency or into synchronizing (or communicating) tasks. Concurrent paradigms and languages do exist but are not yet well mapped into widespread machines.

Object-oriented languages are a potential solution to the abstraction problem.

2.4 Mapping and Programming

There are three parts to matching algorithms to architectures:

Understanding the application and its concurrency
Selection of optimal architecture
Mapping the algorithm to the architecture

Both must be performed well to achieve the expected speed up and performance results. Long-range efforts must continually iterate this process because new architectures and new algorithms can bring large performance increases.

Some of the details that have to be considered when selecting the computer are how each computing mode is supported and the tools provided. Details of supporting architectural features which must be considered are:

Concurrency conservation and loss due to overheads
Processor context switch latency
Memory access latency
Scheduling and mapping difficulty
Control requirements
Data storage and data flow
Communications and the memory access method
Topology
Control and scheduling

Synchronization (Megasynchronizations per second - MSYS)

Performance programmers on massively parallel machines must consider several new elements of machine balance. [Stone] These are:

Processor bandwidth - partition processes among processors
Memory bandwidth - access data in parallel
I/O bandwidth - sustain full computational power
Communications bandwidth - move data between processes
Synchronization bandwidth - coordinate activity of processes
Multiple purposes - maintain flexibility for multiple applications modes

2.5 Resource Contention

All parts of the parallel processor must operate at a reasonable design efficiency and this should be expected by the user. It is inappropriate to require a user coding at the concurrency level to anticipate that hidden pattern related resource contention degradation can occur in his solution. Networks and memories should be conflict-free and support parallel synchronization through instruction level operation. Concurrent processes must be independent of the functional units and processors used to compute the results. Instrumentation must be performed to allow for detection of contention problems because these can be highly non-linear.

2.6 Portability

The need for concurrent machine independent languages, independent of the computer architecture, is evident from the literature survey.

There is now very little or no portability between applications on parallel processors. Each requires that the application must be mapped to the machine in order to achieve reasonable performance. Other machines will not support parallel extensions to the sequential languages provided. Present language extensions are too assembly-like, and are very machine-oriented and dependent. The challenge is to develop concurrent languages which support, without machine considerations, the expression of all forms of concurrency. Architects can then implement computers, machine dependent compiler backends and tools which allow a concurrent expression to be run on the given architecture.

Many expect that these languages will represent the algorithm in both graphical and word forms. The capabilities of present workstations and the standardization of tools such as X-Windows, facilitate this development.

2.7 Instrumentation and Performance Measurement

The user must be provided with highly visual tools to understand the operation of his algorithm on the machine. Providing this operational look allows the user to discover better solutions by expressing the concurrency of the problem in other forms. The user may be mapping the concurrent algorithm to the machine by using this feedback, but does so at a higher level. Other parallel processor designs may operate less effectively but a reasonable level of operation could be expected without tuning.

In addition, it is necessary to establish new and different ways to describe a machine to allow users to effectively evaluate the potential for an application. This action is also at the higher level of abstraction. These machine descriptions must reveal the performance across all modes of processing, not only on highly structured processing domains. Concurrency can be expressed in other domains but is seldom supported by existing parallel processors.

2.8 Flexibility

Systems must be sufficiently robust to maintain reasonable performance while allowing reprogramming without entire rewrites. Small changes should not severely impact the performance. The user must be able to work with this assumption; otherwise the maintenance of parallel computer codes will require that the developers be retained for constant updates. This problem has a parallel with the large maintenance costs incurred by over reliance on performance programming in many systems. If this problem cannot be solved, large life cycle costs and a corresponding reduction in the market for massively parallel computers can be expected.

A recent paper by Kumar [KUMAR] demonstrates that the average level of concurrency in large scientific Fortran benchmark codes is 500 to 3,500 concurrent actions over the application. This level of concurrency increases with the size of the application.

[KUMAR] has shown that concurrency in very large problems is quite high. Kumar's example codes are existing Fortran applications, not specially coded problems in which direct and expanded concurrency could be expressed.

2.9 Architectures

Additional architecture features may be considered, among them are:

- Machine Cycle
- Instructions
- Addressing Modes
- Memory Hierarchy
- Coupling (Local to Shared Memory)
- Cache Strategy
- Secondary Storage and I/O
- Basic Instruction Suite (RISC/CISC Issues)
- Functional Unit Adjudication
- Micro-task scheduling

[Stone] considers these to be the challenges to a computer architect:

- Eliminate the synchronization bottleneck (MSYS)
- Reduce overhead for scheduling tasks
- Solve the cache coherency problem, or
- Find a means of increasing local memory
- Map serial programs to parallel programs
- Identify useful parallelism
- Avoid inefficient forms of parallelism

Section 3. General Parallel Software Survey

Brief descriptions of each article selected for inclusion are given below. Many parallel references were found but not included because they were redundant, esoteric, or remote to parallel tool environments. Names of environments or tools are given in parentheses at the end of each paragraph.

3.1 Algorithms

Signal Processing System Environment

Lager [ALG 1 *][1] gives an overview of a graphically oriented signal processing system which provides the necessary environment for exploring algorithms to build systems. This article is limited to sequential systems. Similar displays have been given in simulation conferences in the past two years.

Purtilo [ALG 2] describes an interprocessor communications support system for design which allows separation of specification and implementation. (POLYLITH)

Bokhari [ALG 3] gives optimal assignment methods using the sum-bottleneck path algorithm which allows polynomial time solutions in some cases. Chains or rings of processors are considered, as well as single host, multiple servers.

Jamieson [ALG 4][2] gives a check list of algorithm characteristics and architecture characteristics which should be dealt with in a complete system.

Chen [ALG 5 *] uses the data dependence graph as a tool for designing algorithms. Forest and multistage graphs are included. Regular and semi-regular graphs are discussed in other papers.

Frieze [ALG 6] provides parallel algorithms for solutions to the quadratic assignment problem on the DAP.

Engstrom [ALG 7] A systolic array programming system is presented. Notation, C code production, and high level specification to allow interfacing with others are stressed.

McCrosky [ALG 8 *] provides a Algorithms for array based computation for fine-grained SIMD machines.

Stone [ALG 9] Stone shows that the speed of the Connection Machine on one of the data search methods is due to I/O bandwidth, not the processing power.

O'Hallaron [ALG 10 *] describes Kalman filter implementation on the WARP.

[1] A star (*) indicates that this article has been reviewed in more detail in the report.

Alexànder [ALG 11] provides multidimensional signal processing methods to optimize communication overheads.

Lin [ALG 12] parallel gives a matrix inversion method for dynamic communication restructuring machines.

Arya [ALG 13] provides system modeling combined with notation for data transfers. This is a tool for algorithm optimization.

Fox [ALGl 14] gives hypercube algorithms for matrix manipulations. Code for this set is available from Fox in C.

Feng [ALG 15] gives communicating sequential processes to develop the alternative construct, thus enabling a process to non-deterministically select one communication among many. These processes were implemented on the BB & N Butterfly.

Armstrong [ALG 16] describes multiple algorithm methods for one-dimensional FFTs for Convex computers.

Swarztrauber [ALG 17] gives both vector multiprocessor and hypercube FFT algorithms. Eight existing FFTs are reviewed.

[ALG 18] is the cover of a special issue of IEEE Transactions on Computers on parallel and distributed algorithms.

3.2 Library

Dongarra [LIB 1] shows how to interpret the results of the LINPACK benchmark from the LINPACK library.

Sneiling [LIB 2] compares several libraries on supercomputers.

Hammarling [LIB 3] reviews the NAG Library on supercomputers.

3.3 Simulation

Ammar [SIM 1 *] shows a method for deriving the time cost of parallel computation, defines classifications of computing structures, and derives an approach to the time cost of each.

Bain [SIM 2] describes Hypersim, a parallel performance simulator for architecture decision making for the Sun and iPSC.

Yoder [SIM 3] compares the simulation of SIMD and a VLSI processor arrays. A word recognition case is used as the application.

Ramamoorthy [SIM 4] provides a set of rules with which to incrementally expand a system and maintain logical correctness. Petri nets are used with the rules to verify logical properties.

Yaw [SIM 5] gives a method of computing the cycle time of concurrent systems modeled by a restricted set of Petri nets.

Chung [SIM 6] develops parallel execution schemes for a Petri net model of a task.

Krauss [SIM 7] presents formal verification of a computer program using labeled Petri nets.

Stotts [SIM 8 *] gives parallel flow graphs (PFG) with visual and hierarchical properties, using a Petri net.

Hura [SIM 9] describes an environment called PNSOFT used for Petri net modeling.

Bray [SIM 10] gives a description of a tools set, including concurrency detection tools, architecture modeling, and optimization of architectures to match the concurrency in the algorithm.

3.4 Mapping

Nicol [MAP 1] uses the elliptical partial differential equation to study the relationship of problem partitioning parameters, determining values to achieve optimal speed up.

Kruskal [MAP 2] gives a series of definitions of granularity and attempts to define two forms: (1) natural, based on the time between required synchronizations; and(2) architectural, based on the communications overhead.

McDowell [MAP 3] shows that reachable program states grow exponentially with the number of tasks. A virtual state is used to merge a set of related reachable states, allowing the static analysis of parallel programs for concurrency.

Berman [MAP 4] presents a solution to the problem of mapping when there are topological mismatches and the number of processors required by the algorithm exceeds the number available.

Cherkassky [MAP 5 *] uses two matrix algorithms to demonstrate one way to optimize balance between communications and computation overheads. Two matrix algorithms are used to demonstrate the approach. Parallel gaussian elimination is also demonstrated.

Moreno [MAP 6] gives a method of partitioning algorithms for application in systolic arrays.

Kumar [MAP 7] gives a general method to reduce bandwidth requirements by mapping two-dimensional arrays into one-dimensional arrays.

Reddy [MAP 8] reduces the links to needed for I/O in hypercube architectures when I/O is embedded in each node.

Dubois [MAP 9] separates parallel computing into throughput-oriented and speed up-oriented approaches. He also discusses synchronization issues in multiprocessors.

Wu [MAP 10] gives an automatic scheduling and synchronization tool for hypercube architectures using a macro data flow graph.

Muhlenbein [MAP 11 *] shows the evolution approach to parallel programming. Parallel programming is shown to be reduced to the graph partitioning problem, which is solved through a biologically-based evolution approach. The example used is the traveling salesman problem.

Bailey [MAP 12] Cover, Proceedings of the 1988 International Conference on Parallel Programming.

Kim [MAP 13] uses a heuristic method to provide a graph representation of a parallel computation, the merges the parallel computation to produce optimized computation and communications costs.

McDowell [MAP 14] detects anomalies in parallel programs using static analyzer methods. McDowell uses a state graph called the Concurrency History Graph as a visual indicator of the concurrency history.

Greenbaum [MAP 15] analyzes execution and waiting time due to processor synchronization. Less restrictive forms of barrier synchronization are proposed for certain granularities.

Stout [MAP 16] gives problems of mapping between various types of architectures. Vision algorithms are the subject used. Mesh, hypercube, mesh-of-trees, pyramid, and Parallel Random Access Machines (PRAMs) are considered.

[MAP 17] Cover, Supercomputing '88.

Martin [MAP 18] gives an overview of mapping.

Fox [MAP 19] Cover, Third Conference On Hypercube Concurrent Computers And Applications.

Fox [MAP 20] gives communications algorithms which map neural network simulations on the hypercube.

Fox [MAP 21] gives neural network methods of load balancing and scheduling on the hypercube.

Salmon [MAP 22] develops the method of scattered decomposition as a tool to show the load imbalance.

Pettey [MAP 23] discusses minimizing communications and memory utilitization in process placement mapping. The NP-complete problem is approached by heuristic and the method of simulated annealing is discussed as one solution.

Livingston [MAP 24] considers the problem of distributing resource units to processors in a hypercube to meet performance requirements.

Ercal [MAP 25] gives a method of task allocation by recursive minicut bi-partitioning.

Chen [MAP 26] studies topology mismatch between the algorithm and the machine. The average path length is to be minimized but the problem is NP-complete. Approximation algorithms are used to find good mapping in specific cases. A greedy mapping strategy is analyzed.

Bell [MAP 27] discusses data partitioning.

Kruatrachue [MAP 28] gives an automatic technique for determining the grain sizes in a parallel program. Optimal execution requires solution to grains sizing and scheduling. Grainpacking is used to get optimal results.

Ramanujam [MAP 29] applies simulated annealing to avoid the local minimum trap problem.

[MAP 30] Cover, Proceedings Supercomputing '88.

Kramer [MAP 31] discusses process assignment in distributed memory parallel computers. A robustness measure is introduced for hypercube machines and SUPRENUM. The result shows that optimal mapping is important to the hypercube but that simple mapping schemes give nearly optimal results in the SUPRENUM.

Musciano [MAP 32] discusses medium grain dynamic scheduling and the SPOC environment implemented in Simultaneous Pascal.

Weiss [MAP 33] gives dynamic scheduling for DOALL-loops and FORK-JOINS to support the output of parallelizing Fortran preprocessors.

Peir [MAP 34] approaches linear recurrences and how to minimize their impact on multiprocessor systems. A new method, minimum distance, is used to create totally independent computations from linear recurrences.

[MAP 35] Cover, Proceedings of 1988 International Conference on Parallel Processing.

Missirlis [MAP 36] discuss the successive over-relaxation (SOR) method on asynchronous multiprocessors.

Kapenga [MAP 37] gives an adaptive scheme for task partitioning on MIMD machines. Many serial adaptive procedures are directly translated into parallel versions. A high level set of macros was developed and used over the Argonne macro package.

Bisiani [MAP 38] describes Aguora a system for multilanguage parallel applications for heterogenous machines. The system uses shared memory abstraction to program across different machines.

Colin [MAP 39] presents a method for task allocation on distributed memory machines using a graph model to show dependencies. A virtual parallel machine is introduced with infinite processor count and has a complete connection graph.

Rosenburg [MAP 40] describes a software behavior simulation for a new SIMD architecture. Mapping between massively parallel SIMD machines and vector architecture was exploited to provide architectures guidance and high level software development before construction. BLITZEN is an evolution of the NASA MPP. Very high peformance simulations were achieved using the mapping form SIMD to vector architectures.

deJong [MAP 41] describes a mapping tool which verifies matrix bounds through symbolic operations, providing additional abstraction for the user.

Katsef [MAP 42] gives a data partitioning method for implementing an assembler on a message passing multiprocessor system. The author supplies other methods for partitioning program text and sharing global information.

3.5 Programming

Gokhale [PROG 1] A data flow language, PS. Globally referenced user-defined data types are defined and then modules are described. Each module is a side-effect free function. Values are assigned by use of definitions which equate a variables name to an expression. No control constructs are allowed. The author analyzes data dependencies and synthesizes order of execution. C is generated but other code may be required to complete the application.

Purtilo [PROG 2] see [ALG 2]

Browne [PROG 3 *] has developed a unified approach to parallel programming. CODE is an architecture-independent programming tool which allows graphical programming using *Computational Units* and *Dependency Relations*. Computational units *have a functionality and a firing rule*. Dependency relations are used to compose the computational units into a parallel computational structure. An architecturally dependant translator (TOAD) is used to map to specific machines. Constraint dependencies are also supported.

Neves [PROG 4] provides a commentary on the problems of parallel programming and the lack of fulfillment of promise of software tools.

DeMarco [PROG 5] discusses four signs of change in parallel processing: data flow methods for systems representation, reusable components, object-oriented languages, and parallel architectures.

Cavano [PROG 6] identifies future costs of software as the driving force in systems.

Cavano [PROG 7] gives a list of short falls in parallel software tools.

Russell [PROG 8] describes a plan for using expert systems to program parallel computers.

Fox [PROG 9] gives categories of coarse grain architectures and three approaches to programming: user provides whole program, user prepares large grain, and user prepares fine grain objects. In the whole program case, C is identified as difficult to automatically decompose because of use of pointers; for large-grain objects Fox has developed annealing and neural network methods of allocation and decomposition, but there is no tool to help the user: all of it is hand embedded. Smalltalk and C++ are reported as having some promise, when object-oriented and virtual machine techniques are used. Fine grain objects are likened tɔ VLSI methods of design. Fox reports California Institute of Technology work on MIMD generalizations of C* programs from the Connection Machine. Gluing compilers are used to build the code grain.

Chandy [PROG 10 *] describes UNITY, a specification notation and proof system for parallel programming, and suggests that hardware architects build computers which effectively execute functional code. Functional, logic, and sequential programming advantages are briefly mentioned.

Sobek [PROG 11 *] gives additional CODE material (See Browne [PROG 3]) on the constraint specification.

Nicol [PROG 12] gives relationships on grid size, stencil type, partitioning strategy, processor execution time, and communication network type. Effects of communication and synchronization overheads are included. Optimal speedup is the goal for PDEs.

Chandy [PROG 13] (Description of UNITY continued.) This article describes stages and the tool required for each: general strategy and restriction of class of solutions, considering architectural issues to rule out more solutions, and designing programs in a series of refinement steps.

Sabot [PROG 14] is a book about architectural independent programming using parallel relations. This is implemented on the Connection Machine.

Hudak [PROG 15] provides a general overview of functional programming. In functional programming the specification and the implementation are separate components. Para-functional programming is a rapid prototyping method. It is functional in that the results of a program operation are to be expressed declaratively, with no state or imperative constructs. Several views of an object (operational and functional) are needed (ParAlfl).

Hudak [PROG 16] see Hudak [PROG 15].

Fox [PROG 17] Proceedings Cover.

Kallstrom [PROG 18] Three parallel computers are compared for ease of programming and performance. The traveling salesman problem was the test. Computers were: iPSC in C), a network of Transputers (in Occam), and Sequent Balance (in C).

Hey [PROG 19] discusses the reconfigurable interconnection of Transputers. Load balancing and communication overheads are the discussion points.

[PROG 20] Cover, 1988 International Conference On Computer Languages.

Bagrodia [PROG 21 *] discusses programming on the Connection Machine and a language, SC, an enhancement of C, which adds data types and primitives to develop parallel programs. UNITY is the basis for the primitives. A data parallel programming style is supported.

McBryan [PROG 22] reviews PDE solutions for the Connection Machine.

Bershad [PROG 23 *] describes PRESTO, an object-oriented parallel programming environment. Pre-defined object types, threads and synchronizing objects, are given to simplify programming and allow multiple threads to cooperate. PRESTO is written in C++.

Brandes [PROG 24] gives a knowledge-based parallelization tool.

Zima [PROG 25] describes SUPERB: a SUPRENUM parallelization tool. Important classes of numerical algorithms hav a knowledge base to support their use.

Percus [PROG 26] discusses parallel software development on MIMD machines using as an example random number generators for Monte Carlo simulators.

Storøy [PROG 27] gives the monitor synchronization concept for defining the highest level of parallelism possible in parallel algorithm design. Concepts are for MIMD machines.

Karp [PROG 28] gives two styles: fork-join and single program, multiple data. A general overview of the state of the parallel programming, including taxonomy, software synchronization issues. The author concludes that the present state is a sorry one.

Polychronopoulos [PROG 29] gives schemes for parallelization of arbitrarily nested loops.

Oldehoeft [PROG 30] describes a prototype applicative language, Streams, and iteration in a single assignment language (SISAL). SISAL is derived from a dataflow language (VAL). An intermediate language is used to separate machine dependencies from SISAL. Applicative languages are declarative, i.e., a set of function definitions. Only data dependencies constrain the evaluation order, giving a higher concurrency.

[PROG 31] Cover, Supercomputing, 1st International Conference, Athens.

Solchenbach [PROG 32] gives parallel multigrid methods for SUPRENUM. SUPRENUM has mapping libraries for processors and communications libraries for data exchange. Multigrid methods use iteration between coarse and fine blocks of nodes to extend the accuracy and speed of numerical computations.

Dongarra [PROG 33 *]describes SCHEDULE and a transportable linear algebra based library. SCHEDULE is a programming environment for explicit programming of MIMD parallel computers.

Jayasimha [PROG 34] uses Markov chains to estimate completion times in barrier synchronization. Implementation algorithms are given for barrier synchronization.

Francis [PROG 35] describes parallel invocation and cessation of processes. The author provides a programming model of multiple simultaneous executing threads which share a single program's code and data space. A parallel procedure call is used to introduce and coordinate parallelism.

Eisenstadter [PROG 36] examines locality of reference in MIMD symbolic computations. Significant gains are seen if locality can be increased.

Weihl [PROG 37] analyzes the level of concurrency obtained in distributed systems, especially transaction systems, when one uses abstract data types, for example, in the presence of commutation between operations. Concern is placed on system integrity features such a reliability and fault tolerance.

Bastani [PROG 38] advocates breaking systems into components of four classes: abstract data types, functional, interface, and control. Two methods of use of abstract data types are described.

Martin [PROG 39] describes a system for parallel procedure calls and models them as network processes. Currently designed in C and C++. A parallel procedure executes a procedure in n different address spaces in parallel.

Parkinson [PROG 40] evaluates the cost of summation operator in a wide range of contexts, demonstrating that simple extrapolations from small number cases have little relevance to the many processor case.

Terrano [PROG 41] developed a compiler for distributed memory reconfigurable architectures, using programmer directives to program, partition, map, and automatically generate communications code.

Preiss [PROG 42] gives a semi-static dataflow method which partitions programs into a collection of data flow graphs. A dynamic method of splicing data flow graphs is used to dynamically create operating contexts.

McGregor [PROG 43] is the introduction to a special issue of the Communications of the ACM.

3.6 Human Computer Interface

Yau [HCI 1] discusses issues of visual languages and major features and the impact on software development from three aspects: software visualization, iconic representation, and graphical grammars.

Krishnamoorthy [HCI 2] proposes graphics primitives for algorithm animation.

Tomboulian [HCI 3] describes a graphical programming environment for the Navier Stokes computer.

Gannon [HCI 4] describes an interactive software tool for optimizing execution on vector-multiprocessors and which support organization and analysis of large application codes. Memory hierarchy problems are identified, a database of global data dependencies is provided and performance is estimated. (SIGMA)

Myers [HCI 5] provides a survey of user-interface development tools. A concise table with both commercial and experimental tools is given.

Bailey [HCI 6] describes Voyeur, an application-specific, graphical views programming tool. Three views are used: Icon, Simulator, and Vector.

Brown [HCI 7 *] describes a large number of algorithm animation applications. Many authors cite this reference because of its completeness and many illustrations.

Hsai [HCI 8] uses pictorial programming instead of instrumentation of code to begin the visualization and animation process. Called Pictorial Transformations (PT), the graphics are converted into tuples (like association pairs in Lisp). Both scenes and films are produced to view the underlying process operations in the programmed algorithms.

3.7 Environments

Bisiani [ENV 1] describes a tool which is a planner for coordinating other tools, allowing the optimum sequence to be followed. The tool is intended to handle tedious and error-prone jobs in software development. Elements include: (1) model of program, (2) high level descriptions to be semi-automatically transformed into low level programs, (3) automation of sequence of tool use This tool is added to the UNIX environment (Marvel).

Dart [ENV 2] provides a taxonomy of software development tools. This is separated into

language-centered, based on one language
structure-oriented, allows the user to directly manipulate structures
toolkit environments, language independent tasks, and
method based, based on a development method

Three tables give a sampling of these.

Reeves [ENV 3 *] describes a programming environment for parallel multiprocessors using Parallel Pascal and stressing system integrity features. NASA's MPP, an SIMD machine, is the target architecture. The multicomputer is also considered.

Carle [ENV 4] describes the R_n scientific programming environment, which is intended for scientific Fortran support of large codes. Automatic parallelization and vectorization are considered. Editors, execution monitors, and optimization are included. (PFC and PTOOL)

Smith [ENV 5] gives Fortran parallelizing tools to aid microtasking of Fortran-based vector supercomputers, including a parallelizer, a static analyzer, and a dynamic debugger. (PAT)

Guarna [ENV 6] describes a support environment for large scientific programs using X-Windows and NFS. Interactive compilation and optimization, integrated editors and compilers, and portability are emphasized. Other managers are compared in a table. (Faust)

Appelbe [ENV 7] (see Smith [ENV 5] provides a parallel programming toolkit including a concurrency history graph, which educates as it automates. Cray microtasking is supported. (START/PAT)

[ENV 8] Cover, Supercomputing '88.

Guarna [ENV 9] (see [ENV 6]) describes an integrated setting for software development. (Faust)

Ertel [ENV 10] describes the Intel iPSC/2 programming environment.

Parasoft [ENV 11] describes the CrOS operating system, CUBIX version of UNIX for the hypercube, Plotix graphical capabilities, MOOSE asynchronous operating system, debugger, and PC-CUBE, an education simulator.

Peinze [ENV 12] describes tools for the SUPRENUM machine including the architecture, runtime environment, and tools. The explicit and implicit methods which support both the program development tools automatic vectorizer, communication libraries, mapping library, and make scripts. Program verification includes a simulator and a debugger. The program evaluation component provides graphics I/O, process visualization (time, statistical, or dynamic maps), and profiling analysis to the subroutine level.

Dongarra [ENV 13 *] describes SCHEDULE, a standard user interface to several shared memory parallel machines. Fortran is supported with calls to SCHEDULE to enforce large grain data dependencies of the algorithm. Machine dependencies are hidden from the programmer and are internal to SCHEDULE.

Pratt [ENV 14] describes a scientific programming environment with extended Fortran, a configuration environment for setting up runs on parallel processors, and a run-time environment for monitoring and controlling program execution.

Dongarra [ENV 15 *] describes SCHEDULE.

Pike [ENV 16] describes in terms of communication channels a concurrent window system with interfaces. Complex programs are assembled from small self-contained units which use these channels. The window system may run recursively to implement sub-windows. This is a Windows software tool.

3.8 Support

Lopriore [SUP 1] describes tools for monitoring program behavior. A user interface has been defined for program debugging, program performance evaluation, and program structure analysis. These tools support many common debugging techniques, performance indices, and structure statistics.

Martin [SUP 2] gives a general method to evalute the performance of a supercomputer system.

Gupta [SUP 3] uses a trace compiler embedded in the debugger to evaluate VLIW computers.

Mills [SUP 4] describes a three-part debugger for the DADO tree computer. A network server, a window manager, and a parallel debugger are provided.

Reeves [SUP 5] gives a method for measuring the performance of the MPP on data permutations, FFTs, convolutions, and arbitrary data mappings. The impact of the high level language is also measured.

Hough [SUP 6] uses animated patterns to explore debugging. Patterns of data and control flow are used to verify fine grain, tightly coupled processes. (Belvedere)

Burkhart [SUP 7 #] presents monitoring tools for debugging and performance measurement. Tools are: breakpoint monitors, mailbox monitor for synchronization, and bus monitor for bus load.

Callahan [SUP 8] describes a collection of 100 loops used to check vectorizing compiler preprocessors.

Eager [SUP 9] analyzes the compromises between speedup and efficiency in parallel systems. Contention, communication, and structure are given as reasons for inefficiencies of processor idle time. The average parallelism of the software is given as the primary factor in the tradeoff.

Pan [SUP 10] describes the Concurrent debugger for the Intel iPSC.

Bohm [SUP 11] is concerned with monitoring and performance comparison in realistic application areas. Models and measures are addressed for execution on a data flow machine.

Flower [SUP 12] gives software utilities for writing parallel code and porting sequential code. [See PLOTIX and CUBIX discussion by Parasoft. (Comfort)]

McGuire [SUP 13] develops a method for measuring the concurrency in a workload. The effect of this concurrency is related to system performance for the Alliant FX/8.

So [SUP 14] presents a quick way to evaluate the performance of parallel programs. A multiprocessor scheduling model is a set of identical processing elements, and units of computation as task. Task marking and task tracing are supported to achieve this result. A speed up analyzer is also described. (SPAN)

Allen [SUP 15] attacks the problem of nondeterminism during debugging. A nondeterminism detector is provided as a debugging tool.

Feo [SUP 16] analyzes the complexity and structure of the Livermore Loops.

Griffin [SUP 17] gives a parallel processing simulator, a window/mouse based debugging tool, and a set of realtime display routines to develop a parallel process debugger. Interface in made to the Sun dbxtool.

Bremmerl [SUP 18 *] presents a layered model to describe debugging, performance analysis, and visualization of multiprocessors and program execution.

Goldberg [SUP 19] represents distributed sequential processes as clones. The Transparent Process Cloning tool keeps the clones of a process mutually consistent. The tool has been used as a load scheduling method.

Kumar [SUP 20] gives a method for extracting general currency from large scientific codes. The concurrency is shown to be far larger than expected for a specific mode or type of parallelism.

Whelan [SUP 21 *] provides optimal decomposition methods for matrices as the matrix changes in dimension. Shared memory multiprocessors are considered.

Miller [SUP 22 *] gives a debugging method which allows for incremental tracing. The Parallel Program Debugger (PPD) is directed toward shared memory machines. Inter-procedural and data flows are analyzed using PPD.

Stone [SUP 23] describes speculative reply, a method of debugging, which creates a concurrency map, and allows investigation through back-up of the process.

McCreary [SUP 24] developed a graph method which automatically determines grain size for problem partitioning. A DRAM (distributed random access model) is used to allow the inclusion of communication costs.

Cheng [SUP 25] provides a knowledge-based system for parallel programming. The method is programming language-independent. The interprocess communications are the target of the method, because the individual processes are sequential and handled by normal methods. Distributed Event Based Language (DEBL) is a specification level language allowing automatic execution of debugging, along with the program itself, in the distributed environment.

3.9 Language Extensions

Gehani [EXTLANG 1 *] describes two examples using Concurrent C, an upward-compatible C language. C++ is integrated with Concurrent C to provide data abstraction and concurrent programming.

Carlton [EXTLANG 2] describes a parallel Prolog implementation which uses AND parallelism across a network.

Shibayama [EXTLANG 3] gives transformation rules for concurrent object systems made up of computational agents capable of concurrent execution and message passing. The author also explores methods of splitting and merging concurrent objects.

Stevenson [EXTLANG 4] gives a compiler based system for analyzing sequential programs for concurrency. Data groupings, operations, communications, and control flow are the subjects of the analysis.

Chen [EXTLANG 5] describes a mathematical notation language to address programmability and performance of parallel machines. The language (Crystal) expresses concurrency without unseen sequential dependencies. The algorithm designer is responsible for the amount of currency, but the language allows for concurrency to be easily expressed.

Grossman [EXTLANG 6 *] describes a language system for scientific engineering, and mathematical application programming. Textbook mathematical expressions are used. The syntax is like technical English. The screen editor supports four cases of English, Greek, and math symbols.

Wholey [EXTLANG 7] presents Connection Machine Lips. Objects, similar to arrays or hash tables, called xappings, are used to express concurrency. CM Lisp is embedded in Common Lisp. The author claims the CM Lisp is suitable for other computes, such as, NON-VON and Ultracomputer.

Fisher [EXTLANG 8] provides an abstraction mechanism for SIMD machines allowing powerful code optimization techniques to be applied. Fisher claims that this compiler produces code of hand-crafted quality.

Ruppelt [EXTLANG 9] describes the principles of automatic transformation system which transform specifications into parallel programs for the SUPRENUM. PDEs are supported with vectors, matrices, domains, and grids at a high level of abstraction. (SUSPENSE)

Halstead [EXTLANG 10 *] defines *futures*, a symbolic parallel programming method and the MultiLisp language. The highly data dependent sequence of operations in symbolic processing is supported by these constructs. (Multilisp)

Felten [EXTLANG 11] describes Coherent Parallel C, a concurrent language which provides a parallel programming model with one entire process for each data object. Transparent task assignments are made by the system. Communication calls are not seen at the user level. CPC runs on the NCUBE. (CPC)

Dally [EXTLANG 12] defines a Smalltalk-based language for concurrent object oriented programming. Distributed objects, locks, and synchronization support is provided. The language is for fine-grain parallel computers. (CST)

Rosing [EXTLANG 13] describes a C-based distributed memory parallel processor language. Interprocess communication and process control are primary concerns of the development. The user defines a virtual machine into which data structures are distributed. C++ was used to develop a prototype of this language. (DINO)

Wolfe [EXTLANG 14] discusses synchronization schemes for multiprocessors, including data dependence, removal of synchronization points, random synchronization, pipelining, barrier synchronization, and critical sections.

[EXTLANG 15] Cover of ESOP 86 European Symposium on Programming held in Saarbrucken.

Triolet [EXTLANG 16] addresses automatic parallelization of Fortran programs with procedure calls. The author provides a method for paralleizing CALL statements.

Zorn [EXTLANG 17] discusses the extension required to parallelize Common Lisp and compares this to the requirements for parallelizing other concurrent Lisp ystems is given including Spur Lisp.

Allen [EXTLANG 18] describes the IBM parallel compiler used in 3090 Vector facilities.

Callahan [EXTLANG 19] provides an approach to implicitly programming distributed memory parallel computers. The language describes distribution of shared array elements in the parallel computer.

Mehrotra [EXTLANG 20] describes a block-structured language for multiprocessor programming. Array arithmetic, forall loops, and accumulation operators are provided for fine-grained concurrency. An applicative or functional procedure invocation is used for coarse grain compiler assistance. (BLAZE)

3.10 Operating System Extensions

Chen [EXTOS 1 *] presents a dynamic memory management system which tracks dependencies and improves speedup using on-the-fly sorting of data elements.

Beck [EXTOS 2] shows how Sequent extended UNIX single-process programming models to support parallelism, including extensions to C, Fortran, and their assemblers and linkers to provide declaration and intializing of shared data. The author also discusses run-time support for shared memory initialization, expansion and heap management and provides a parallel multitasking model for C++.

Ellis [EXTOS 3] gives method for dynamic storage allocation for shared memory multiprocessors, providing four algorithms each for a different granularity. The author describes the conducted experiments to compare the performance of the 4 schemes.

Wolfstahl [EXTOS 4] provides special system calls to support mapping of processes to processors. Mapping directives are used to signal forthcoming changes in communications patterns and occurrence of mapping related events.

Bain [EXTOS 5] describes Interwork II, the iPSC concurrent workbench.

Tolle [EXTOS 6] describes UNIX utilities for the NCUBE.

Angus [EXTOS 7] describes Fortran CUBIX, an I/O facility of the CrOS for hypercubes.

Schwan [EXTOS 8] describes an operating system construct to implement communication graphs linking multiple tasks of a parallel program. Global services are also provided by the construct. (Topologies)

Gait [EXTOS 9] gives a process scheduler which adaptively controls a two-tier system moving processes between local memories using a shared bus.

Schroder [EXTOS 10] describes a decentralized and distributed operating system in an MIMD environment for process execution and communications support. This system operates on top of SUPRENUM. (PEACE)

Stevenson [EXTOS 11] analyzes the problems associated with distributing many virtual processes across a multiprocessor, and discusses the issues which must be resolved for operating systems.

LeBlanc [EXTOS 12] describes controlling set of processes which interact to execute a function. This paper discusses processes not completely parallel, but which must be partially ordered to proceed in parallel. A balanced binary tree structure is used to organize the execution, which was implemented on the BB & N Butterfly.

Malony [EXTOS 13] describes a message-passing facility for shared memory multiprocessors, implemented in a portable C library. (MPF)

Vornberger [EXTOS 14] describes a method used to process Prolog programs on a network of personal computers. The author discusses TERM, AND and OR forms of parallelism.

Rahgozar [EXTOS 15] describes a distributed data base system which increases parallel processing efficiency by means of semantic information of transactions and data.

Garg [EXTOS 16] defines two constructs which support high level specification of distributed systems: the *handshake* and the *unit*. Handshake is a remote procedure call for multiple parties. The unit construct provides synchronization and call restrictions on the handshake. These are part of a formal model, which can be automatically analyzed, called Synchronous Token based Communicating State (STOCS). Addition of these constructs to C to allow concurrent programming are given. (ConC)

Fleckenstein [EXTOS 17] describes a utility for make which operates on multiple workstations to achieve significant speedup and ease of operation. The version of the UNIX make tool controls compilation and linkage of a number of programs across a network.

Baalbergen [EXTOS 18] gives his version of parallel make and gives an analysis of performance. The target is multiple processor systems.

3.11 Languages

DiNitto [LANG 1] discusses future programming languages, basing projections on areas of past under-accomplishments in languages despite project goals and extensive research.

Perrott [LANG 2] describes an array and vector language which is architecture independent. (Actus)

Tripathi [LANG 3] describes an object oriented concurrent language and inter-object communication support. Concurrency and synchronization between objects have been the focus. (SINA)

Sharpiro [LANG 4] provides an overview of Concurrent Prolog, a parallel logic oriented language. (Concurrent Prolog)

Baldwin [LANG 5] describes a parallel constraint language. The constraint languages are special case of predicate calculus methods and are based on abstract systems of constraints, such as logic programming. (Consul)

Tick [LANG 6] compares parallel logic programming architectures, both derived from Prolog. The author compares the speed of developing OR- and AND- logic programs.

Clark [LANG 7] describes a parallel Prolog language. (Parlog)

Yamazaki [LANG 8] explores object-oriented programming at the low level (operational units and registers) and high level (interacting processors).

Hansen [LANG 9] presents a Communicating Sequential Processes (CSP) and Pascal based language for parallel programs. Concurrent agents communicate through unbuffered channels. The language is written for the Encore Multimax. (CSP)

Karp [LANG 10] compares Fortran languages for these computers: Alliant FX/8, BB & N Butterfly, Cray X-MP, ELXSI 6400, Encore Multimax, Flex/32, IBM 3090/VF, Intel iPSC/2, and Sequent Balance.

Gelernter [LANG 11] discusses issues in parallel languages. This article is the introduction to a special issue of Computer on parallel languages.

Mundie [LANG 12 *] discusses parallel processing in Ada, concentrating on the real time environment and Ada's tasking model.

Goldman [LANG 13] describes a multiprocessing Lisp designed for multiprocessors, which supports medium-grain parallelism, explicit parallelism, and run on a shared memory space multiprocessor. (QLisp)

Polychronopoulos [LANG 14] describes compiler optimizations and their impacts on architecture design. Automatic detection of parallelism is the concern. Barrier synchronization is identified as one of the serious sources of runtime overhead.

Guzzi [LANG 15] describes Fortran for the vector multiprocessor Cedar computer.

Girkar [LANG 16] describes methods of identifying data dependencies for use in automatically vectorizing and parallelizing by compilers. A program transformation called loop spreading is used to execute adjacent loops with interloop dependencies.

Welch [LANG 17] describes occam as a language for the Transputer, and shows how the language supports abstraction, structuring, and information hiding.

Clapp [LANG 18] gives a demonstration of Ada operations on a hypercube. The paper concentrates on the runtime system required for a machine independent support of Ada. The author also describes implementation of this system on the hypercube.

DeForest [LANG 19] describes a tagged demand driven dataflow model of parallel computation and how it runs on the hypercube. The declarative language Lucid is also presented. (Lucid)

Lake [LANG 20] reviews the approaches to parallel languages taken in Fortran 77, SISAL, occam, Fortran 8X, and concurrent Prolog.

Jordan [LANG 21] addresses the need to direct the activity of a large number of processes and the structure of parallel languages for large-scale computers. (Force)

Ahuja [LANG 22] describes the distributed language Linda, the demands for its use, programming methods, and the Linda kernal. (Linda)

Whiteside [LANG 23 #] gives the results of a Linda experiment on a Local Area Network of Digital Equipment Corporation VAXs. A matrix multiplication ''master'' is given. Several computational examples are analyzed.

Lunberg [LANG 24] gives a parallel Ada system on an MIMD multiprocessor. A single unmodified Ada program with a number of tasks executes in parallel on different processors, transparent to the programmer. The run-time system controls allocation and migration of the tasks.

Watson [LANG 25] provides a strong argument for using inherently concurrent languages for parallel problem solving. He advocates a real life problem specification which is declarative.

[LANG 26] is the cover of the Institution of Electrical Engineers (IEE) specialist seminar on design and application of parallel digital processors.

3.12 Operating Systems

Bradley [OS 1] describes a flexible operating system experiment testbed for hypercube machines. The testbed was designed to investigate communication paradigms, task scheduling, global virtual memory, heterogeneous system resources, and peripheral management. (Picasso)

Krumme [OS 2] describes the NCUBE operating system, debugging support, communications, and design. The debugger provides a top-level view of the progress of a computation over time.

Salmon [OS 3] describes a multitasking operating system (MOOSE) for the hypercube used to research both load balancing and the decomposition of irregular and dynamic problems. (MOOSE)

Pierce [OS 4] describes the NX/2 operating system for iPSC/2. (NX/2)

3.13 Architectures

Dinning [ARCH 1] surveys methods of synchronization of MIMD parallel computers. Computers covered are: BB & N Butterfly, Cedar, HEP, HM^{2}p, Sequent, Transputer and the Ultracomputer.

McBryan [ARCH 2] reviews parallel computer architectures for computational science, concentrating on PDE solutions.

Snyder [ARCH 3] gives a classification scheme extending Flynn's classical one. Categories are: von Neumann, packed von Neumann, SIMD with no addressability (MPP, Connection Machine, and systolic arrays), SIMD multigauge (splitting multipath), addressable SIMD (Illiac IV and CM/2), VLIW, MIMD multigauge, and MIMD parallel (Ultracomputer and Cosmic Cube).

Casavant [ARCH 4] is a panel announcement listing architects and programming tool builders of reconfigurable parallel computers.

Martin [ARCH 5] addresses the general issues of building a performance benchmark for vector and parallel computers.

Hack [ARCH 6] discusses the teraflop computer and the lack of direction toward general purpose computing, which could realize large parallel speed increases. The author discusses potential limitations of speedup for general purpose computing.

Hwang [ARCH 7] surveys supercomputing architectures, including the state of software tools.

Harp [ARCH 8] describes the reconfigurable Transputer project and its architecture.

Bronnenberg [ARCH 9] describes an architecture built to support object-oriented parallel processing. An associated language, (POOL and DOOM), is also described.

Treleaven [ARCH 10] describes commercially available architectures and explores their potential applications.

3.14 Technology

Bell [TECH 1] discusses the new forms of computing becoming available due to highly parallel machines.

3.15 Additional References

As this survey went to press IEEE's Computer released a special issue on Visualization in Computing. The following articles are the latest references in visual programming:

> Ambler, A.L.and Burnett, M.M., ''Influence of Visual Technology on the Evolution of Language Environments,'' pp.9-22.
>
> Roman, G., and Fox, K.C., ''A Declarative Approach to Visualizing Concurrent Computations,'' pp.25-36.
>
> Lehr, T., et al., ''Visualizing Performance Debugging,'' pp.38-51.
>
> Kramer, J., Magee, J., and Ng, K., ''Graphical Configuration Programming,'' pp.53-65.

All are from Computer, Vol. 22, No. 10, October 1989.

Section 4. Commercial Literature Survey

4.1 Commercial Support

Table 4-A lists commercially developed parallel programming tools. These were received in response to a letter requesting parallel processing tool information.

COMPUTER/COMPANY	TOOL/AREA	DESCRIPTION
Gesellschaft Fur Numerische Supperrechner MBH SUPRENUM	The Filter/ Graphical	Filtered process data reporting
	The Dynamic Map/ Graphical	Displays activities of a distributed application
	The Statistical Map/ Graphical	Generates a final evaluation of the process data & displays some statistics on the performance of a distributed APF.
	Automatic Vectorizer/ Algorithm	Automatically transforms existing Fortran 77 codes using a vectorizor
	Automatic parallelizer	Parallelizing sequential programs
	Communications Library	Facilitates programming and testing of grid-based problems
	Mapping Library	Mapping of processes
	Make	Provides comfortable environment to initiate runs on simulator or SUPRENUM hardware
	Fortran (Automatic Vectorizor)	
Helios/Distributed	Helios AMPP	Assembler macro pre-software Limited processor for programming in high-level macro notation (includes 30+ macros &

		ability to create new macros)
Butterfly/BB & N	Math advantage Butterfly Fortran Butterfly LISP Butterfly Ada Butterfly C	Industry standard subroutine Library with 200+ frequently used mathematical algorithms
iPSC/2 Intel	VECLIB	Math function; library extensive library of arithmetic functions that can be called from a Fortran program BLAS Vector multiplication, division, scaling Transcendentals: SIN, COS, EN, EXP Gather, scatter (versions: single, double, complex, integer)
	DECON (Concurrent debugger) VAST-2 Fortran vectorizor	
	User Software Library: Numeric Software	
	LINPACK	Linear equations
	SPARSEPACK	Sparse matrices
	FISHPACK	3-d fast poisson solve
	SUPRENUM	Special communication subroutines
	NAVIER	Fluid dynamics code which solve NAVIER-STOKES equations
VP (Vector Processors) Series/Star Technologies, Inc.	Arithmetic Control Processor (ACP) macros	1) Vector arithmetic: clear, fill, Star add, subtract, multiply, divide, square root, square signed square 2) Multiple-operation arithmetic (vector multiple and add)
DAP (Distributed Array	Hardware, processing	

COMPUTER/COMPANY	TOOL/AREA	DESCRIPTION
Processor)/Active Memory Technology (AMT)	capabilities, applications, technical reports	
Scalable Parallel Supercomputer (SPS-2)/ Myrias Computer Corp.	PARDO programming model	Used as extension to Fortran & C to express parallelism in applications
	MPF	Myrias parallel Fortran
	MPC	Myrias parallel C
	G System	Target language for the compliers
	Myrias UNIX	
Horizon Tera Computer Company	Future Supercomputer - 1993 (MIMD)	Automatic parallelizing compilers Fortran and C
FLEX/32 Multicomputer System/ Flexible Computer Corp.	Weitek Floating Point Library	
	QTC Scientific Floating Point Library	
	Programming and Operating Environments:	
	UNIX System V	Sequential or concurrent operating environment
	Concurrency Simulator under UNIX System V	Simulated MMOS concurrent operating environment
	Multicomputing Multi-tasking Operating System (MMOS)	Concurrent operating environment
	Concurrent C Concurrent Fortran Ada	
	Concurrency Simulator Debugging Tool	
FPS M64 Series Floating Point Systems	Program Development Software (PDS)	
	Optimizing Fortran 77	

COMPUTER/COMPANY	TOOL/AREA	DESCRIPTION
	Compiler Optimizing C Compiler Overlay Linker Object Librarian Interactive Debugger Math Library	
Computing Surface/ Meiko	Development Environment: Meiko OS VMS SUN OS Standard editors and utilities: Fortran 77 C Pascal	
IBM	Parallel Fortran	
IP-1/ International Parallel Machines	Architecture	
Cogent Research XTM Parallel Computing Environment Cogent	Linda Programming Environment Developing Parallel Versions: C++ Fortran	A parallel programming environment
Trace/300 Series Multiflow Computer	Trace Compiler: Fortran C Pascal Common LISP Ada Enhanced Math Libraries I/O Libraries String Manipulation Libraries UNIX System Call Libraries VMS-Compatible System Call Library TRACE/UNIX: Compilers Libraries Debuggers	

COMPUTER/COMPANY	TOOL/AREA	DESCRIPTION
	Profiles Source Code Management Tools	
Computer System Architects/ Transputer Chip	Transputer-based Parallel Processing Products	
	Express System (Added to existing operating system)	Parallel operating environment
	C Fortran Parallel Source Level Debugger Parallel Performance Monitor	
	Runs on a variety of parallel computers	
The Connection Machine/ CM-2 Thinking Machine	C* Paris *LISP FORTRAN CM-LISP	Assembly language of CM-2
NCUBE	Hypercube Supercomputing	
NCUBE Fortran 77	Axis system C Graphics/four	UNIX-like operating
ORYX Super Signal Processor	Graphic Flowgraph Editor	
ORYX	Flowgraph Compiler	
	Loader	
E & S Parallel Programming Environment (ESPRE) Evans & Sutherland	Manual Methods	Parallel FORTRAN compiler directives SCHEDULE library C Thread library
	Automatic Methods	Compiler Optimization ES/FORTRAN compiler Math libraries with parallel algorithms
	Support	Perf. tuning (gprof)

COMPUTER/COMPANY	TOOL/AREA	DESCRIPTION
		Debugging (gdb) Parallel program building (make)
	Compiler	ES/FORTRAN ANSI std. with VAX extensions Multiple pipeline optimization PCF parallel programming directives
	Operating System	ESIX
	Library	IMSL, NAG, P-STAT, MATH ADVANTAGE, ELSPACK, UNPACK, BLAS, LAPACK FFT

Section 5. Reviews of Selected Articles and Information

Articles identified with a * are reviewed in this section.

5.1 Algorithms

Lager is reviewed to provide an overall concept for a graphically-oriented signal processing algorithm development environment. Chen is reviewed because of the importance of data dependencies to parallel programming, and the foundation he provides for building data dependency tools. McCroskey's work matches immediate SIMD programming requirements. O'Hallaron gives an implementation of a Kalman filter on the Warp.

5.1.1 Signal Processing Algorithm Environment

Lager [Lager, D. L. and Asevedo, S.G., ''SIG - A General-Purpose Signal Processing Program,'' Proc. IEEE, Vol. 75, No.9, September 1987, pp. 1322-1332. [ALG 1]

SIG is an environment which supports the use of several signal processing methods with a given application.

The functions of SIG are

simulation and reading of signals for input to the system
arithmetic and scaling operations on the signals
correlations, time-frequency transformations
model coefficients
curve sitting and plotting
generation of text files for reporting

The key to efficient use is a dual mode user interface for either expert or casual users. The user menu is programmable and new functions may be easily added. The user interface supports starting new programs under its control, and operating system commands can be given to the system. A database is used to store text files and parameters. Databases for time signals, frequency spectra, real and complex coefficients are stored is another database.

The components of SIG are

command processor
data base
parameter file
menu package
help software
graphics software
user interface
signal processing
user commands
user software

A command summary, given on pages 1328 and 1329, gives a wide variety of supported commands.

SIG has been widely distributed and copies are available from the National Energy Software Center, Argonne National Laboratory and from Techni-Soft, Livermore, CA.

The usefulness of SIG to the parallel processing tool environment is that a similar set of functions is needed for algorithm development. Additions must be made to allow for the interaction with parallel architecture models but SIG or a similar environment is a useful sequential starting point.

<u>5.1.1 Data Dependency Tool</u>

Chen, G.H., and Chern, M., ''Designing Parallel Algorithms from Forests and Multistage Graphs,'' Proc. **The Twelfth Annual International Computer Software & Applications Conference**, Chicago, October 5-7, 1988, Editor: G.J.Knafl, pp 292-298. [ALG 5]

Chen gives a three-stage process for designing parallel algorithms, expressing the dependencies using a data dependency graph (DDG). The three stages are: finding a solution method, constructing a data dependency graph, and designing the algorithm from the dependency graph. The DDG is essential for detecting parallelism and Chen has identified four classes: forest, multistage, regular, and semi-regular. Data dependency graphs are constructed from nodes and arcs. Nodes represent each intermediate value and arcs represent the dependencies. Trees representing a root with other nodes having a single input are common in parallel processing. Stages occur in recurrent equations and dynamic programming algorithms.

A forest is a graph containing several disjoint trees. A multistage DDG made up of multiple disjoint stages. Chen shows how these are mapped into shared memory, linear arrays, multiple linear arrays, and two-dimensional arrays using the DDG, and also shows an approach to computing identical intermediate values.
The design of a parallel processing tool set requires that data dependencies be understood and supported through node-arc formulations and approaches such as Chen's.

<u>5.1.2 Systolic Programming Tool</u>

McCrosky, C., ''Realizing the Parallelism of Array-Based Computation,'' **Parallel Computing**, Vol. 10, pp. 29-43, 1989. [ALG 8]

McCrosky provides array operations for SIMD machines and describes algorithms, and finds the relationship of communications and parallelism to be an important part of algorithm development. Array data structures and operations provide an abstraction constraint which allows a concise statement of the algorithm. Arrays correspond to many data structures in the problem. Parallelism is naturally expressed through arrays. Three methods are given for using parallel processors: smart compilers for automatic detection of parallelism in existing languages, use of explicitly programmed parallelism, and new languages with semantics which allow the expression of parallelism.

McCrosky concludes that a new generation of languages are necessary for highly parallel machines.

5.1.3 Parallel Algorithm Implementation

O'Hallaron, D.R. and Radhakisan, S.B., ''Parallel Implementation of a Kalman Filter on the Warp Computer,'' **Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications** Vol. III, Pennsylvania State University Press,August 15-19, 1988, pp. 108-11. [ALG 10]

Matrix multiplications, triangularization, coordinate transformation, and Jacobian transformations are used to develop a parallel Kalman filter algorithm. The Warp computer is used for the processing. The Warp is a linear array of processors is suited to the Kalman filters compute-intensive nature. A directed acyclic graph (DAG) is used to express the algorithm, and an associated table gives the computational requirements for each operation. Nodes and arcs are labeled with data items and represent the precedence relation. These DAGs are convenient ways to express linear systolic array, data flow, and shared memory MIMD problems. The mapping to the Warp uses a topological ordering where each operation is assigned to Warp cell in the precedence sequence. The cells receive the results of the processor ahead of it, performs its operation, and sends the results to the following cell. Data is stored locally when possible, but is shifted along through the cells when global access is required. Mapping was not optimal, no attempt was made to exploit all the parallelism, and only six cells of the Warp were used. One-half the sample time was consumed by restarting the computation from the Sun controller for each sample.

5.2 Library

5.2.1 Standard Libraries

Snelling, D.F. and Hoffman, G., ''A Comparative Study of Libraries for Parallel Processing,'' **Parallel Computing**, 8,pp. 255-66, 1988. [LIB_02]

Snelling reviews existing libraries on parallel supercomputers. Snelling believes that Fortran has inherently sequential bias and that constructs such as COMMON and DATA lead to poor parallel programming. He identifies occam, Ada, Concurrent Pascal, and SISAL as parallel languages for scientific applications. Data control, debug effects, hardware affinity and complexity/ease of use are the evaluation terms. All libraries have TEST-AND-SET operations which allow for intertask synchronization. Cray, IBM, ETA, FPS T-Series, and Snelling's portable library (SPLIB) are reviewed.

Data control is identified as the most important aspect of parallel programming. Five types of Fortran data control are given: private, invariant, result, reduction, and message.

Debug effects are exacerbated in parallel programming because of the difficulty in analyzing debug dumps. Effects are grouped into stampede, bystander, deadlock, irreproducibility, and Heisenberg. The stampede effect is to the inability of a single process to stop all others when an error is encountered. The bystander effect when one process corrupts another processes's data space. The deadlock effect occurs when synchronization errors cause all processors to stop or trade synchronization signals without progressing. The irreproducibility effect is due to the non-deterministic nature of parellel programs. This may be proper because round off differences can result in different answers, using different proper paths through the problem. The Heisenburg effect is the impact of debugging instrumentation or diagnostics on the results, creating error through perturbation of the operational environment.

Hardware affinity of libraries is the change in library structure or content due to the hardware architecture or the architectural history. Complexity and ease of use issues are a measure of the number of paths into the library, number of parameters of subroutines or arrays, the number of memory levels, and the number of altered lines of code required to convert a parallelization of a simple program.

Standard Parallel Library (SPLIB) is built on constructs of processes, barriers, channels, and shared matrices. A process is a sequence of independent operations, a barrier is a form of synchronization which allows a set of processes to complete together. Channels provide synchronous communication between processes and the shared matrix provides communication between processes and a large globally-shared memory. Access to the SPLIB is through Fortran 77 subroutine calls.

5.3 Simulation

5.3.1 Time Cost Model

Ammar, R.A. and Qin, B., ''A Technique to Derive the Detailed Time Costs of Parallel Computation,'' Proceedings Computer Software & Applications Conference, 5-7 October, 1988, Chicago. COMPSAC '88 pp. 113-119, 1988. [SIM_01]

Ammar develops computation structure models which are used to derive the time costs of parallel processing. Five categories are defined and described using directed graphs for control and data flow. The time costs for each is determined. To apply the method, a parallel computation is recursively reduced to a sequential one using combinations of the five categories of parallel structure. The method requires expansion to include varying numbers of processors, contention, execution overhead and environment.

5.3.2 Visual Programming

Stotts, P.D., ''The PFG Language: Visual Programming for Concurrent Computation,'' **Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III**, Pennsylvania State University Press, Aug 15-19, 1988. [SIM_08]

Stotts describes Parallel Flow Graphics (PFG) which allows expression of concurrent, time-dependent computations. Timed Petri nets and hierarchical graphs are the basis for the semantics. Each syntactic structure has a direct translation into a portion of a timed Petri net model from which, concurrent properties can be analyzed. Visual programming with bit-mapped graphics and icons are used. The PFG is a convenient way for the user to specify the mathematical model of an algorithm. Both static and dynamic program analysis through execution of the hierarchical Petri net model is used. A formalism called the Hierarchical Graph (HG) software system model which represents time-dependent systems (software and hardware). Three models are used: data model, static program model, control flow model.

The data model uses graphs of structure and interrelationships among collections of data to be transformed by the computation under study. The static program model is a representation of operations on data a set of non-overlapping blocks. Operations within each block are sequential. Each has its own local data area and may be data copied in or out as required. Complete determination of operations that alter data is supported. The control flow model expresses the possible parallel execution threads of a concurrent computation. A thread is a sequence of basic blocks form the static program model. The control flow model is a timed Petri net interpretation.

The graphical set of icons include a base down triangle for concurrency branching, a half-circle for a non-deterministic branch, a base up triangle for join, and a rectangle for a sequential block. The block has an associated data transformation which is entered, displayed, or modified by clicking on the icon. Each arc connecting the icons is labeled and is governed by constraints Timing information is handled by associating durations with blocks. Primitives are primitive (sequential) or parallel. A recursive method is imposed to reduce parallel operations to a timed primitive one for time estimating purposes. Minimum and maximum timed modes are derived through path analysis of the concurrent reachability tree during the time analysis.

Uses of PFG include the dual timing, detecting improper accesses of shared data, and detecting deadlocks. The system is developed on the Sun Workstation and languages such as Ada and Modula-2 are also possible in the PFG environment. Executable code is generated after analysis by the Petri net method.

5.4 Mapping

5.4.1 Optimal Mapping on Hypercubes

Cherkasky, V., and Smith, R., ''Efficient Map Program and Implementation of Matrix Algorithms on a Hypercube," Journal of Supercomputing, 2, pp.7-27 (1988). [MAP05]

Cherkasky is concerned with the optimal speed up for parallel matrix operations on hypercube architectures (specifically NCUBE). The overhead costs of communications and computation are the parameters he controls. The maximum number of processors for optimal solution is the target and the authors considered square matrices only. A toroidal mesh topology is the decomposition target. A matrix/submatrix partitioning notation is used. Matrix multiplication is performed (A X B = C) wherein each node holds a portion of A, B, and C. Communications require orthogonal movement of A and B and computation of C. Cherakassy's algorithm has reduced the communication time over that of Fox [ALG 14]. The reason cited is that only one complete circuit of the toroid is necessary for A and B to pass though a given node. Cherakassy also examines Gaussian elimination. In the analytical models as the submatrix size n increases the arithmetic increases by O(n^3) for both Gaussian elimination and matrix multiplication. In comparing the two algorithms the complexity of considering communications tradeoffs and computational size is exposed.

5.4.2 Novel Mapping for Massive Parallelism

Muhlenbein, M. G., Gorges-Schleuter, M. and Kramer, O., ''New Solutions to the Mapping Problem of Parallel Systems: The Evolution Approach," **Parallel Computing**, 4, pp.269-79,(1987). [MAP 11]

This article contends that massively parallel machines are better understood by models derived from the natural sciences. Simulation of these models is best done by the massively parallel computer. The programming model is for SUPRENUM. The model presented is evolutionary, that is, based on a model of evolution. The mapping problem presented here is the graph representation consisting of process set and a communications matrix. The process structure has both static and a dynamic components. For the dynamic case the communication matrix must be created during the runtime. The process set is partitioned into clusters, clusters are then randomly assigned to processors. Evolutionary theory is applied through a three-step process: replication, mutation, and selection. The processor mapping is in terms of creation, randomization, and selection. Applications are made to graph partitioning and the traveling salesman problems and results are rated as surprisingly good. The method is a form of data level parallelism programming.

5.4.3 Bounds Checking

de Jong, V.J., ''Symbolic Bounds Checking in a Matrix Language,'' Proc. 1988 Int. Conf. on **Parallel Processing**, Vol. III, Penn State University Press, pp.73-80, Aug 88 [MAP 41]

de Jong describes a symbolic range checking tool which generates run time restrictions on input variables. APL, SAS/IML, and MATLAB are given as interpreted matrix languages. These lack performance because they are not compiled but are very useful in rapid prototyping. One of the requirements of matrix languages is that the user should be isolated from matrix notation problems, and associated type checking and dimension bound checking issues. de Jong gives facilities that should be provided to the user - input restrictions, error recovery, help, and database interfaces. de Jong developed CONDUCTOR, a statistical software matrix language allowing experienced technical users to avoid matrix notation issues. He takes advantage of the simplified language structures necessary to express matrices to define a symbolic approach to checking ranges. The imposed restrictions are: index variables are not reassigned inside conditional statements and index expressing are monotone increasing or decreasing. The matrix language syntax is given.

5.5 Machine Independent Programming

5.5.1 Graphical-Machine Independent Programming

Browne, J.C., Azam, M., and Sobek, S., ''CODE: A Unified Approach to Parallel Programming,'' **IEEE Software**, pp.10-18, July 1989. [PROG03]

and

Sobek, S., Azam, M., and Browne, J.C., ''Architecture and Language Independent Parallel Programming: A Feasibility Demonstration,'' **Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications** Vol. III, Pennsylvania State University Press, Aug 15-19, 1988. [PROG 11]

The goal of this University of Texas at Austin team is to unify approaches to parallelism to provide portability and higher abstraction level programming. Conventional languages are extended to handle parallelism on shared memory computers. Operations for synchronization are added: Lock, unlock, and semaphores. Dependencies are resolved by regulation of access. Message passing mechanisms are used for partitioned memory architectures. The Computation-oriented Display environment is a graphical programming system which allows the user to define computational units and dependency relations. The computational unit defines the functionality and firing control. Functionality is the transformation on the input set to the output set. Firing rules specify the states of input dependencies which allow the unit to execute. The Dependency relations compose the computational units into a parallel structure. Dependency types in CODE are data, demand, mutual exclusion, and control.

Architectural independence is a key goal of the CODE system. This is achieved by the following:

Separating dependency specification from computational unit specification
Separating firing rule specification from functionality specification
Raising the abstraction level at which dependencies and firing rules are specified

CODE is a version of generalized dependency graphs. Each graph node represents a computational unit or a subgraph. Each arc represents a dependency. Languages used to specify computational units are Ada, C, Fortran, and Pascal. Firing rules are predicated on the state of the computation. Programming is graphical: creation of nodes and arcs and with support for windows allowing detailed information to be associated with the node or arc. The steps in programming are

Draw dependency graph
Fill in forms to define dependencies among computational units
Fill in forms to define degree of parallelism and functional code of the computational units
Specify firing rules
Invoke CODE to create a machine independent program specification

The functional code is entered in a sequential form and CODE generates the necessary headers and expands it to the degree of parallelism specified.

When it is ready to run a machine dependent translator is used to generate code for a given architecture. The translators generate code to implement communication and synchronization among computational units. These translators have been created for The Sequent Balance, Digital Equipment Corporation's VAX clusters, Intel hypercube, and Cray XMP. Code is implemented on the Sun in C, and uses SunView (TM of Sun Microsystems) graphics.

5.5.2 High Abstraction Programming

Chandy, M.C., and Misra, J., ''Architecture Independent Programming,'' **Proceedings: Third International Conference on Supercomputing**, Supercomputing '88, Vol., III, pp.345-351, 1988. [PROG 10]

Chandy advocates that the programmer should be able to choose the programming style best suited to the development of correct maintainable programs and design for machines best suited to the selected approach. Functional languages should be developed by the programmer and the computer architects should develop machines which run the languages effectively. Programming notation should provide machine independent forks, joins, and messages. This avoids the explicit representations in a given architecture. Note that MIMD and SIMD architectures do not necessarily support the same parallel constructs. UNITY is a specification notation, a programming notation and a proof system.

UNITY was developed to maximize flexibility in language, architectures, and compilers; to optimize portability and efficiency; and to resolve the conflicts between these goals. Language standardization is not likely, so a variety of languages are anticipated. Likewise architectures have immense diversity. Compilers for parallel systems cannot be standardized because programming methods are too immature to incorporate. Portability is needed because of the rapid flux of architectures. Efficiency and associated performance are critical to the use of parallel computers and the use of machine dependent primitives. Chandy offers a design method to resolve the conflicts between these goals. Stages in his design are general strategy, architectural issues, and machine tailoring. They are supported with a specification notation, programming notation, and a proof theory.

5.5.3 Tools for Connection Machine Programming

Bagrodia, R., and Chandy, K.M., ''Programming The Connection Machine,'' **Proceedings: 1988 Intl. Conference on Computer Languages**, Miami IEEE Computer Society, Oct 9-13, 1988, pp. 50-57. [PROG 21]

The authors describe a language called SC which makes it easier to write programs for the Connection Machine (CM). SC is based on the UNITY approach (See [PROG 10] above.) and is an enhancement to C, adding data types and primitives to develop parallel programs. The purpose is to make programming easier, supporting MIMD in addition to the SIMD mode of the CM, adding machine-dependent programing to obtain final efficiencies. The UNITY based notation is compiled into C* and executed on the CM. Primitives and data structures common to scientific programming are supported (arrays, sets of indices, and lists). The primitives defined operate on the SC declaration type index-set. The operational primitive classes are: reduction - binary operation on a set of operands, parallel computation - (par) which operates on a set of statements, sequence - to handle non independent operations, and equations - the central form of expression.

5.5.4 Object-Oriented Parallel Programming

Bershad, B.N., Lasowska, E.D., and Levy, H.M., ''PRESTO: A System for Object Oriented Parallel Programming,'' **Software Pract. and Exp.** 18(3), pp.713-32, Aug 88. [PROG 23]

Bershad describes an object-oriented programming environment for multiprocessors, using a set of predefined object types to simplify parallel programming under Dynix on Sequent Balance and Symmetry computers. C++ is the language used. He observes that object-oriented programming makes it easier to understand distributed systems programming. Problem decomposition and run-time synchronization are details described by an object model. Each object performs a small part of the problem and maintains its own consistency checking. Efficient concurrency and synchronization mechanisms can be worked out and supplied to the user in a higher level of abstraction. PRESTO allows redefining the primitives to avoid limiting the user to a particular paradigm of programming. Several classes are provided to support the parallel programmer in using the constructs of threads and synchronization. Threads are created or started. Synchronization constructs are relinquishing locks, non-relinquishing locks, monitors and condition variables, and atomic integers.

A run-time library system supports PRESTO to map user threads onto physical processors and to provide access to a global shared memory where objects reside. A single scheduler object keeps track of all threads which are runnable but not yet running and to prevent duplicate running on different processors. Migration of threads is possible. Costs of threads and synchronization are provided.

5.6 General Programming

5.6.1 Scientific Programming Environment

Dongarra, J.J., and Sorensen, D.C., ''A Portable Environment for Developing Parallel FORTRAN Programs," **Parallel Computing**, 5, pp.175-86, 1987. [ENV 13]

Dongarra, J.J., **SCHEDULE: Tools for Developing and Analyzing Parallel fortran Programs**, Argonne National Laboratory, TM# 86, Nov. 1986. [ENV 15]

Dongarra, J.J., et al., ''Programming Methodology and Performance Issues for Advanced Computer Architectures,'' **Parallel Computing**, 8, pp. 41-58, 1988. [PROG_33]

Dongarra surveys the techniques of programming computers with advanced architectures and presents the SCHEDULE programming environment. He emphasizes that conversion of existing algorithms and invention of new algorithms must be

supported; and that the compromise of performance without portability has tempted many architects and programmers of parallel machines. Dongarra gives two approaches.

The first approach is recasting algorithms in terms of high-level modules (e.g., the Basic Linear Algebra (BLAS) routines), and coding in high level modules allowing the user to avoid the tedium and errors of machine-dependent coding to gain performance.

The second approach is use of a standard interface to exploit the parallel capabilities of machines with multiple processors through explicit parallel programming. A large grain control flow form of coding is presented.

Modularization by encapsulation of basic matrix and vector operations into a set of high level modules provides improved clarity, potability, and ease of maintenance, making feasible high performance and portability across a large number of machines. The BLAS are given as an example, now in their 16th year and third level (Level 2 is vector-matrix and level 3 is matrix-matrix). Standardization is given as one of the important contributions of BLAS. LINPACK was originally coded in level 1 BLAS but performance was improved through level 2 BLAS. The level 3 version now allows multiple vector processor operations.

Explicit parallel programming of low-level detail has been successful but has limited portability. Dongarra reports that the tools which require multiple parallelism and dynamic allocation to be less than adequate. The range of extensions provided by computer vendors is highly architecturally-dependent. The available synchronization, automatic loop parallelism and process initiation with widely varying costs (as much as six orders of magnitude) point to the immaturity of the industry. SCHEDULE is a package which allows for portability calls of Fortran parallel programs. Machine dependencies are hidden within SCHEDULE, possibly requiring difficult mapping, but providing portable applications code.

Dongarra uses the term execution dependency for assertions made by the user about the order in which operations are to occur. The execution dependency is expressed in a control flow graph. Large-grain control flow dependencies must be understood by the programmer for successful programming. A SCHEDULE program consists of processes and the control flow graph. SCHEDULE provides a mechanism for expressing the execution dependencies and the processes, leaving the user to determine the correctness. In partitioning the problem data, is separated into local and global. SCHEDULE supports both static and dynamic graphs.

The SCHEDULE environment includes goals for debugging and performance monitoring. Some potential errors are avoided because explicit scheduling is not a part of SCHEDULE. SCHEDULE is used by first developing and debugging a sequential version on a conventional computer. A triangular matrix solution is given as an example. SCHEDULE is now implemented on multivector processors and implementation on hypercube machines is contemplated. The NCUBE implementation uses one master node to control the dependency graph operations.

5.6.2 Data Flow Programming

Preiss, B.R., and Hamacher, V.C., ''Semi-static Dat flow,'' Proc. 1988 Int. Conf. on Parallel Processing, Penn State University Press, Vol. III, Aug 88, pp.127-134. [PROG 42]

Preiss describes a data flow programming method which allows both static and dynamic execution. Method used is dynamic data flow graph splicing: creating new contexts and moving data tokens between processes. Activity templates are used to represent static dataflow programs. Dynamic data flow requires multiply-linked activity templates. Program graphs are not re-entrant for the static case, but the dynamic case allows reentry and loop unraveling. A context is a small to medium grain process which evaluates an activity graph. This concept is extended in dynamic dataflow to include conditional statements which modify the context. Communication channels are used for transfer of data into and out of the context. The Preiss approach dynamically splices the data flow graph in four phases to allow for function invocation. The four phases are: context generation, parameter passing, concurrent execution, and result passing. Preiss demonstrates sequential iteration using the method. Programs (written in occam) were used to measure performance parameters for matrix multiplication, fast Fourier transform, Cholesky decomposition, and congruence transformations.

5.7 Human Computer Interface

5.7.1 Algorithm Animation

Brown, M.H., ''Exploring Algorithms Using Balsa II,'' Computer, May 1988, pp.14-36. [HCI 7]

Graphical representation of programs as they are running (or were run) provide useful insight into the algorithm. A more complete understanding of the algorithm is possible. Brown presents an extensive collection of algorithm animations which allow the program to be controlled during simulation as well as changing the way information on the running process is presented to the developer. In Balsa-II, the user watches through several viewing modes. The view location, size, zoom level, pan direction, and point of observation can be adjusted as the algorithm proceeds. A scripting facility which records the user's actions is also available. Balsa-II is a programmer's tool with facilities for controlling execution, managing display, and scripting.

There are three phases in using Balsa-II:

(1) splitting the program into the components of algorithm, input generators, and views which present the animation,

(2) implementing each component (e.g., annotating the algorithm to mark an interesting event), and

(3) setting up the Balsa-II by identifying the views and input generators to be used by textural names.

Significant programming skill and knowledge may be required to instrument an algorithm so this is a programmer's tool which provides a user's view.

During operation, pull-down menus provide user control. Views such as sticks provide a bar graph of parameters vs. time or dots which spatially illustrate the operational distribution. The partition tree view provides an execution graph. The H-bars show convergence. A history function compares the same view at different points in time. An adjacency matrix view provides the data distribution over matrix values. Dialog boxes provide textural information on the runs.

Programming to use Balsa-II involves inserting event markers and defining the parameters which control how the event is to be animated. Controls for both the algorithm and the input generator are defined; event and selection routing are also set up. Balsa-II uses adapters to transform the views to be presented.

Payoffs of visualization are significant. Balsa-II is now a sequential system tool, but it presents a good model for parallel algorithm analysis.

5.7.2 Pictorial Mapping

Hsia, Y. and Ambler, A.L., ''Programming through Pictorial Transformations,'' Proc. 1988 Int. Conf. on **Parallel Processing**, Vol. III, Penn State University Press, pp.10-16, Aug. 1988. [HCI 8]

This paper describes using visual representation of data structure and manipulating the data structure to develop program algorithms. Pictorial Transformation (PT) is a combination of graphical programming and processing visualization. The user develops both a picture and a film (sequence of pictures). The process of building these is captured and recorded as films. Control is exercised through a selection path and a selection condition set. The initial situation must be defined and is changed using predicates during the film process.

5.8 Environments

5.8.1 Multi-Computer Programming Environment

Reeves, A.P., ''Programming Environments for Highly Parallel Multi-Computer,'' **The Third Conference on Hypercube Concurrent Computers & Applications** - Vol. I, California Institute of Technology, Jan. 1988 pp. 458-467. [ENV 3]

Reeves analyzes the difficulty in developing a programming environment due to the wide range of different programming paradigms, even on the same architecture. The requirements are for environments which provide for fault tolerance, dynamic load balancing, dynamic algorithm selection, and automatic task decomposition and allocation.[3]

[3]Others point to the need to combine small grain into large grain for multiprocessor use.

Parallel Pascal on the Massively Parallel Processor (MPP) is the high level programming language that is the basis for the programming environment described in this article. It includes parallel expressions and array allocation across processors. There are three additional classes of operations: data reduction, data permutation, and data broadcast. Reeves mentions the Gibbs framework for scientific programming in which each idea is captured in a single module, called a chapter. The first chapter gives the algorithm, the second the grid structure, the third the numerical method, etc. For parallel programming, primitives are more efficient than compiled operations, and each architecture type has different synchronization support. Primitives are (1) shared memory machines using semaphores, forks and joins, (2) message passing machines using send, and receive, and (3) vector machines using loops. These primitives are closely related to the machine architectures to obtain maximum efficiency. The combination of parallel scientific programming for SIMD machines requires permutation and data mapping primitives, using complex index expressions to replace program loops. The data structures must also be mapped to the size of the processor array.

Reeves classifies problems as trivial, SIMD, and complex. Trivial class problems have a high degree of parallelism with no requirement for communication. SIMD are those for which efficient solutions are known. The complex class consists of problems which have irregular interactions between processing units, requiring load balancing and scheduling.

The SIMD programming environment includes support for the following:

Parallel expressions - array data types
Parallel data declaration - to specify the memory in which the array should reside
Permutation function - shift, rotate, and transpose
Reduction functions - maximum, minimum, sum and product, plus boolean functions any and all

Distribution functions - implicit distribution of scalars to arrays (expand)
Sub-array Selection - select row or column
Conditional Execution - ''where-do-otherwise''

Reeves reports implementation on multicomputers as well as the MPP, and discusses additional extensions for vector indexing, concurrent partitioned program paths, global data permutations, local data permutations, and sparse matrix handling.

5.8.2 Parallel Programming Tool Requirements

Bremmerl, T., ''An Integrated and Portable Tool Environment for Parallel Computers,'' **Proceedings of the 1988 International Conference on Parallel Processing: Algorithms and Applications, Vol. III**, Pennsylvania State University Press, Aug 15-19, 1988, pp.50-53. [SUP18]

The tool environment Multiprocessor Monitoring System (MMS) provides debugging, performance analysis, and visualization of program execution. It is portable and expandable to various architectures and supports several abstraction levels. A hierarchical layered model for tool environments is presented in the paper.

Bremmerl remarks that almost all parallel processors have a host/target environment which is well supported with programming languages, compilers, and linkers on the conventional host. The missing components are debuggers and performance measuring tools. Those which are available are not integrated into the programming environment, which forces monitoring to be a very low level of abstraction. Those tools typically focus on instrumentation techniques and synchronization concepts. The programming interest provides source code instrumentation, operating system instrumentation, and runtime instrumentation forcing a batch mode of operation and self-effects of the instrumentation on the running problem. Also the tools are machine or at best architecture class dependent.

Requirements for a tool are

(1) Window based concurrent debugger
  Display and modify states of programs running on multiprocessor
  Specification of complex predicates about dynamic execution
  Control of breakpoints and tracing

(2) Performance analyzer for optimizing concurrent programs
  Provide information on efficiency of communications between processes activation of processes

  Access to variables
  Access to operating system
  Recording or scripting these
  Help identify bottlenecks

(3) Visualization of processes and operations
  Graphical view of execution
  Flow of communication
  Display complex data types
  View of control and data flow

The tool must be interactive and controlled by the user. Monitoring must be at all levels: hardware, operating system, object code, and hybrid. The abstraction level must vary from language to assembly. Portability to other architectures is necessary.

The design is layered, much like a communications system. For example, monitors might evaluate predicates about control flow, data flow, concurrency object (tasks, mailboxes, semaphores) and combinations of these. This monitor can be applied a the hardware, software or a combination.

The initial target for the tool set is the Intel iPSC, but the host is the VAX under Ultrix.

5.8.3 Partitioning for Matrix Multiplication

Whelan, M., Gao, G.R., Yum, T.K., ''Optimal Decomposition of Matrix Multiplication on Multiprocessor Architectures,'' **Proceedings of the 1988 International Conference on Parallel Processing: Algorithms and Applications, Vol. III**, Pennsylvania State University Press, Aug 15-19, 1988, pp.181-185. [SUP 21]

Whelan addresses the appropriate grain size for parallel tasks and the allocation to optimize a parallel system. A decomposition method is used for matrix multiplication since it is a typical benchmark in parallel systems. The regular iterative nature of matrices is used to decompose the matrix operation into tasks. An analytical model of cost of computation and communication results. An architectural model is used for multiprocessors which includes times to transfer data elements, fetching code, cache organization, and cache size. A simple model of decomposition relates computational and communication requirements to the number of MXM matrices used in the decomposition as shown in the table below:

TABLE OF DECOMPOSITION MODELS

Processor	Compute Time	Communicate Time
1. Uniprocessor	M^3 T*	$2M^2$ T_1
2. M Processors(1 per column)	M^2 T*	$(1+M)$ M^2 T_1
3 M^2 Processors	M T*	2 M^3 T_1

Where t* is a compute time and T_1 is a communication time for the multiprocessor.

In decomposition, submatricies are used and the models applied to determine efficiency. The optimum partitioning for a given processor can be found by varying the regions of the partition. Whelan describes methods of communications overlap effects and optimizing to the partition sizes. A curve is shown which allows communications computing tradeoffs to be made.

5.8.4 Replay for Debugging

Miller, B.P., Choi, J., ''A Mechanism for Efficient Debugging of Parallel Programs,'' **Proc. SIGPLAN '88 Conference on Programming Language Design and Implementation**, Atlanta, June 22-24, 1988, pp.135-144. [SUP 22]

Miller and Choi describe the speculative replay approach, which approach allows the reconstruction of the behavior of a program from histories of its individual processes. Known time dependencies between events in different processes are used to break the processes into dependency blocks. A concurrency map history is constructed. Known dependencies are preserved and compared to the replay. If a mismatch occurs then the process backs up and an alternative ordering is used to create a match. Additional dependencies allow added levels of detail. A shared queue example is presented.

5.9 Language Extensions

5.9.1 Combined Object Oriented and Concurrent Language

Gehani, N.H., and Roome, W.,D., ''Concurrent C++: Concurrent Programming with Class(es),'' **Software Pract. and Exp.**, 18(12), pp.1157-77,Dec 1988. [EXT LANG 1]

Two upward compatible supersets of C are described - Concurrent C and C++ - which provide data abstraction and parallel programming facilities. Concurrent C++ is an integrated language combining the two. Integration issues are given in the paper.

C++ provides the *class* for data abstraction, with specification and body components. All information needed by user for use and by the compiler to allocate class objects is included in the specification. The body contains functions declared in the class specification. There are private and public components of a class: private class components are data items and functions which implement class objects which are not accessible by the user; public class components are data items, constructors, destructors, member functions or operators, and friend functions. The public components make up the user interface.

Concurrent C consists of *processes*, which are a set of components that execute in parallel. Facilities are provided for declaring and creating processes, process synchronization and interaction, process termination and abortion, priority specification and waiting for multiple events. *Transactions* are used for process interaction, and are classified as services called by other processes. Synchronous transactions block and asynchronous transactions are non-blocking.

The two supersets of C are orthogonal, the benefits of object oriented C++ can be gained for Concurrent C when the two are integrated. Classes can be used to ensure that the protocol for interacting with a process is properly observed and that implementation details are hidden form the user. Gehani gives an example of a disk driver where the multiple use of the driver is hidden from the user and the multiple disks in the system are supported without inputting multiple versions of the driver. In a second example, the Concurrent C window manager provides multiple virtual terminals. Concurrent C++ is used to increase the robustness of the window function by the addition of classes.

The two languages are separate preprocessors for C. A complile time option selects whether or not Concurrent C or Concurrent C++ are used. Ther merger of the two languages should make it possible to use data abstraction along with concurrent programming facilities. A goal is to have the same flavor or representation to the user.

In the merged version the following data abstractions extend the concurrent environment:

Class variables can be used in process bodies
Class types can be used for transaction and process argument types and as transaction return-value types

Reference types can be used for transaction and process argument types (in shared memory implementations)

Class names can be used as a type with a class keyword and process names can be used as a type

Classes and processes are both abstraction facilities. Processes can be used to implement some of the same objects as classes, albeit with reduced efficiency. Classes are more efficient when concurrency is not required because of a smaller overhead.

5.9.2 Mathematical Language

Grossman, F., Klerer, R.J., and Klerer, M., ''A Language for High-Level Programming of Mathematical Applications,'' **Proceedings: 1988 Intl. Conference on Computer Languages**,
Miami IEEE Computer Society, Oct 9-13, 1988, pp.31-40. [EXT LANG 6]

The AUTOMATED PROGRAMMER is intended to reduce the effort in programming scientific and engineering applications. The notation is modeled after textbook mathematical representation. Self-documentation is used to aid in reducing error. English Greek, and two cases of math are provided. Matrix arithmetic is accomplished by using standard textbook notation.

Instead of Fortran calls to functions, this language provides a facility for using mathematical notation to express problems and compile them for solution. A full screen editor is used and frequent symbols for integration, summation, product series, square root, etc. are provided by function keys. Variable may by named or represented by Greek symbols. A syntax is supported for English-like statement for control of program flow and for reads, writes, printing, etc. Superscript operations on matrices are supported (superscript t for transpose, superscript -1 for inverse, superscript n to multiply a matrix by itself n times, etc.

The automated programmer works through translation into Fortran and creation of program modules from libraries. The DOS operating system is used on the PC or PS/2 environment.

5.9.3 Parallel Symbolic Lisp

Halstead, R.H., ''Parallel Symbolic Computing,'' **Computer**, August 1986, pp.35-43. [EXT LANG 10]

Halstead gives a review of parallel symbolic programming. The spectrum of programs has numeric programs at one end and symbolic at the other. The differences suggest different approaches to parallel processing.

Numeric Programs are characterized by

Function is to deliver numbers to an ALU to calculate a result.
Generally has a data independent flow of control, arithmetic is emphasized.
Same sequence of operations are performed no matter what the values of the operands are.

Matrices and vectors are common data structures.
Programming tools include automatically parallelizing compilers and languages featuring explicit parallelism following a communicating sequential process model.

Symbolic programs are characterized by

Programs emphasize the rearrangement of data. (To the degree that when data is changed it is called a ''side effect.'')

Principle function of a symbolic program is the reorganization of a set of data so that the relevant information in it is more useful or easier to extract.

Sequence of operations is often data dependent.
No simple operation style (abstraction constraint) such as found in numeric programs.

Applications include sorting, compiling, database, symbolic algebra, expert systems, and AI.

Concurrency is found in recursion on composite data structures such as trees, lists, and sets.

The concurrent language developed by Halstead is Multilisp. Multiisp is a version of Scheme (a public domain Lisp system) having extensions which allow the programmer to express concurrent execution. Scheme differs form other Lisp systems because it has lexical scoping to promote modularity and a privileged class of procedures which may be passed as arguments, returned as values, stored in data structures, and treated as other kinds of values. Multilisp includes side-effect primitives for changing values and altering the data structure, this feature requires additional control to ensure deterministic execution. However, the default is sequential execution. The (future X) construct is used to invoke concurrent operation. With future command a future is returned as a place holder for the value of X and a task is created to concurrently evaluate X. When the value is ready the value of X replaces the future at X. Any task needing the future's value is suspended until the future is resolved. Lazy evaluation, where expressions are not computed until they are requested, is accomplished by a delay primitive. The future construct creates a data flow architecture-like style. Much of the value of future is in the ease of gluing of programs together for concurrent execution.

5.10 Operating Systems Extensions

5.10.1 Memory Management on Massively Parallel Architectures

Chen, M.C. and Jacquemin, M., ''Footprints of Dependency: Towards Dynamic Memory Management for Massively Parallel Structures,'' Proceedings: Third International Conference on Supercomputing - Supercomputing'88, Vol. III, Editor: Kartashev, pp.486-494. [EXT OS 1]

This paper explores the techniques for increasing performance on the Connection Machine by recording data dependencies during execution and dynamically sorting portions of the data elements according to dependencies. Dependencies may cause load unbalance, cache inefficiencies, page misses, extra communication, or other execute efficiencies. Chen believes that the amount of dependencies is an inverse measure of the parallelism of the algorithm. Both code and data must be distributed among processors. (For SIMD machines the same code is distributed to all.)

Chen attempts a dynamic data allocation method for static data dependency graphs. Key to the method is the recognition that dependencies act to spread data out in time across the computer, so one solution would be to assign data according to the time epoch in which they are needed.

The author classifies parallel problems based on dependency and regularity (systolic).

Problems which are easily addressed in parallel fall into three classes:

> Few to no dependencies and highly regular (called embarrassingly parallel or trivial by some)
>
> Significant local dependencies but with a high regularity
> Few dependencies but irregular with little or no structure

Two other classes are more difficult:

> Significant dependencies but no known structure
> Significant dependencies with a dynamic structure

Chen's method applies to the first of these. In these cases, optimization of cache and secondary storage systems used by the parallel processor is needed. Anti-dependent (non-assigned) data are distributed by association with the processor. Chen uses dynamic (on-the-fly) sorting to keep the system finely tuned and upgraded to maintain optimal operation, which is feasible because sorting is very efficient on the Connection Machine. Significant speedup gains are reported for example problems.

5.10.2 Parallel Ada?

Mundie, D.A. and Fisher, D.A., ''Parallel Processing in Ada,'' Computer, August 1986, pp.20-25. [LANG 11]

Mundie presents an overview of Ada's microtasking model. Ada provides high-level mechanisms for task creation and synchronization which are machine independent and operating system independent. (Model is defined as the relation between the program structure and the multiple virtual processors on which the model will be implemented.) The tasking model for Ada is based upon macro-tasking by process-level models. Synchronization of tasks is the fundamental issue in concurrent programming of this type, it is required both as a mechanism for mutual exclusion, preventing multiple processes from accessing the same computer resource simultaneously and for data exchange, preventing transmission prior to receiving process being ready. Ada avoids difficulty and complexity of programming and maintaining semaphores and signals, provides structured primitives for task synchronization and multiway waiting, and also allows for a compromise between shared and message-passing memory-based machines. The compromise works because synchronization is implicit and can be implemented efficiently on both types of processors.

Shared memory, the most effective method, is difficult to extend to widely-dispersed, loosely-coupled systems. Message-based systems have adequate local memory for wide dispersal, but the performance penalty is substantial when compared to shared memory. In Ada the implicit synchronization allows multiple access with restrictions that allow for loosely coupled systems. Updating is guaranteed only at synchronization points, allowing the shared variable to exist in several memories, with copies being passed back and forth only during rendezvous.

The usefulness of Ada as a tool in parallel programming is limited principally because restrictions in tasking make parallel programming unnecessarily difficult. These restrictions are

Calling tasks must name the task whose entries they are calling
Entries must be declared in the task that accepts them, and
The select statement, used to perform multiple simultaneous rendezvous, does not allow mixed or multiple calls

5.10.3 Distributed Computing

Ahuja, S., Carriero, N., and Gelernter, D., ''Linda and Friends,'' Computer, August 1986, pp.26-34. [LANG] 22

Ahuja describes the language Linda. In parallel programming the programmer is concerned with many points and their relationships in computational time-space. The Linda language attempts to make the parallel programming process easier by eliminating much of the programmer's concern about coupling between processes. Actions are taken through replication in addition to the normal partitioning found in other concurrent languages.

The Goals of Linda are to meet the needs of programmers. These are

(1) A machine-independent and portable programming vehicle

A high level programming model without idiosyncratic system calls to support a particular variant of message-passing, memory sharing or SIMD computer

Must run on a range of architectures from one language
Must be able to communicate about parallel algorithms allowing growth of knowledge of methods

Tools must be suited to user needs not the architecture's

(2) A programming tool that eliminates dealing with spatial and temporal relationships among parallel processes:

No coupling require for knowledge of other processes existence
No response is necessary before the sending process proceeds with its actions.

(3) A programming tool that allows tasks to be dynamically distributed at runtime:

More available concurrent tasks than the processors can use
Providing the even distribution needed for good speedup. A dynamic scheduling is needed because static mapping may be impossible.

(4) A programming tool that can be implemented efficiently on existing hardware:

The language must run on the appropriate class of computers.
Run-time systems and operating systems that defeat the strengths of the language must be present.

Linda uses a distributed data structure, called the tuple, to achieve much of its power. Actions are read, add, and remove and a logical name is used. Revising tuples requires removal, revision and reinsertion. This ensures that many processes can share access. Tuples are coarse-grained and access to them can be efficient on loosely coupled architectures.

In Linda the program is replicated as many times as there are processors and all processors attempt to work over a distributed data structure (tuple space). The worker programs ignore each other in this process. The advantages which accrue are

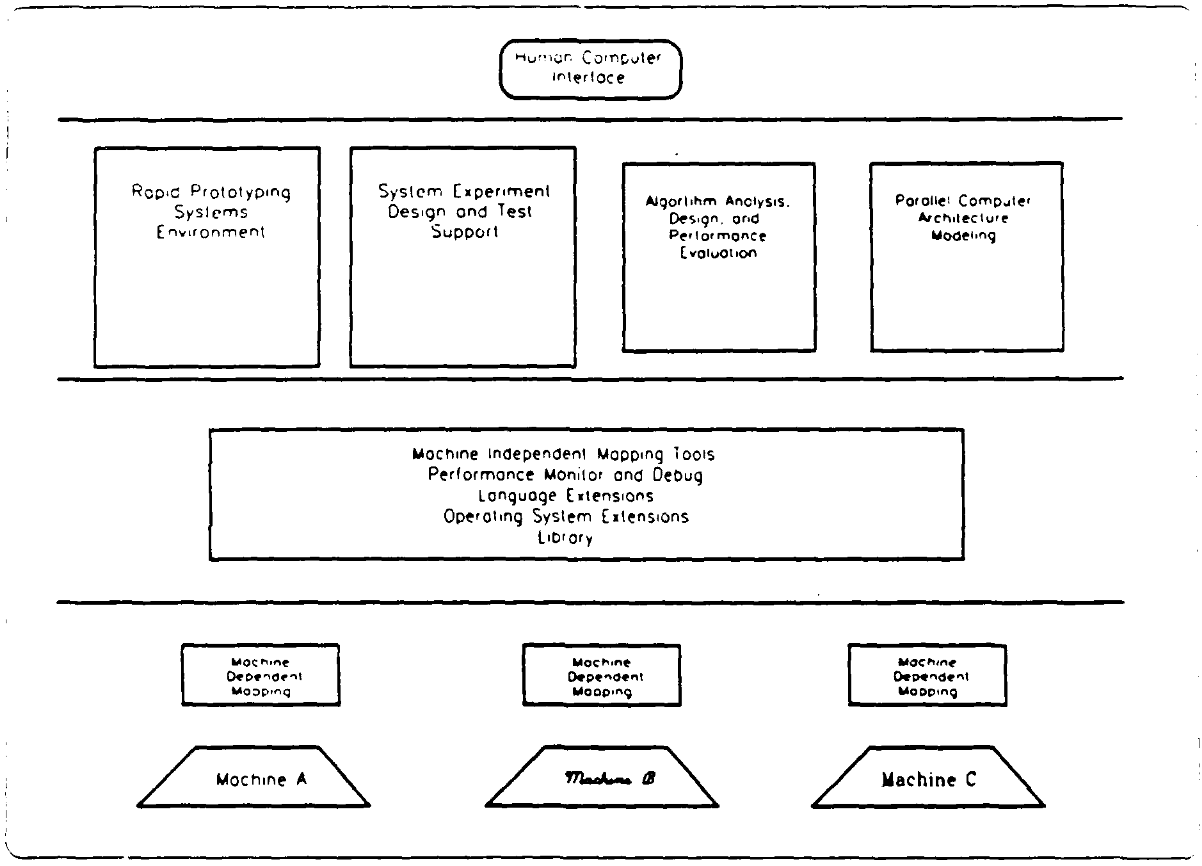
Transparent scaling
Elimination of context switching
Dynamic load balancing

Mixed partitioning is used when programs cannot be cast into a pure replicated form.

Section 6. A Parallel Programming Environment for Parallel Signal Processing

6.1 Tool Environment

Figure 6-1 gives an overview of a ''total environment'' as defined from the literature survey.


Human Computer Interface
Rapid Prototyping Systems Environment
System Experiment Design and Test Support
Algorithm Analysis, Design, and Performance Evaluation
Parallel Computer Architecture Modeling
Machine Independent Mapping Tools
Performance Monitor and Debug
Language Extensions
Operating System Extensions
Library
Machine Dependent Mapping
Machine Dependent Mapping
Machine Dependent Mapping
Machine A
Machine B
Machine C

6.2 Parallel Programming Tools

6.2.1 Visualization

Visualization tools allow the user to observe the concurrency in the language and machine level code. Visualization is necessary to debug parallel algorithms and code, allowing observation of sequential bottlenecks and overloaded processors and network links. Adjustment of the algorithm and the code is provided immediate feedback to the operation. G. Fox [Fox 3] describes the methodology used by a graduate student, who developed a very fast-sorting algorithm. The student used visualization followed by continuous iteration of observation and rework.

Architecture simulators and instrumented architectures are needed to provide visual feedback to the programmer on the operation of the machine on the concurrent code. To identify bottlenecks such instrumentation should provide the lengths of all queues in the system.

Work in animation of sequential programming techniques should be transferred to concurrent software engineering tools. One example is the work of Brown [HCI 7] in his Balsa II environment. By annotating and splitting code into components (algorithm, input generators, and views to present animated pictures), the Balsa II systems provide algorithm animation displays. These displays are used to track dynamic and abstract processes, providing visualization needed to refine and debug algorithms. Similar tools are especially necessary for parallel programming, debugging, and advancing algorithm development.

6.2.2 Tools for Heterogenous Mapping

Network research on distributed systems has led to development of several operating system approaches: Mach, CRONUS, and ISIS are distributed operating systems. Ada, Linda, and ANSpec are languages which support parallel distributed programming (with the support of an underlying parallel real time support system). Tools such as automatic microtasking (spreading of processes across several identical MIMD machines) and automatic vector identification and processing are the most advanced at this time. Such tools work best with hardware assistance. For example, the Cray Y-MP has added memory functional support which allows a new microtasking compiler to more easily spread its work across the eight processors.

6.2.3 Applications Probing and Design Allocation Tools

The importance of understanding the application cannot be overemphasized. Tools which allow the user analyze code for the concurrency expressed in it would be very valuable. A concurrent code analysis tool acts upon high level specification or source code or on the output of a parallelization precompiler and produces directed graphs, much as that done by the front end of a standard language compiler. Analysis engines for different ideal modes of parallelism are used to pass over these graphs and produce a histogram result which identifies the levels of concurrency of different parallel types expressed in the compiled code. Such a tool of this nature would allow the user to identify the requirement for parallel types, allow the recoding of sequential elements and tuning of performance, provide estimates of performance on real implementations of the ideal architectures. In fact such a tool is necessary if the result is to be flexible and programmable. In addition, the tool is needed for the allocation of functions and algorithms to parallel processing modes and supporting architectures. Sequential bottlenecks in many parallel architectures can be identified by this tool. The characteristic performance evaluation process allows each architecture to be used in its best mode.

6.2.4 Modeling and Performance Tuning Tools

The output of the concurrent code analyzer and the allocation to processors is a design with each function spread into parallel processes characterized by a uniform processing mode. These processing modes are used to match those available on the parallel system. The system building process now requires that these processes be tied with the proper synchronization and data transfers to perform the application. The high-level tool necessary for this binding is a modeling tool which simulates each mode in the architecture. The user can define performance parameters by rule insertion. The application system is described by the object-oriented connection of icons representing the computing architectures and networks. The tool models system resources and processes, computes performance parameters, and is capable of monitoring the actual distributed system.

6.2.5 Comparison of Model and Application Requirements

The code application analyzer produces numerics on the degree to which the required processes fit the architecture in the system performance levels either from the ideal architecture (computing mode) or from actual architectures (distributed system). A graph of operations is generated by automatically generating a PERT program. A coarse model is used for each mode of processing allowing coarse-level tuning. The process follows that used in VLSI circuit engineering. The processors in the system are considered parts, as though they were from the parts catalog. Models of their operations are used in the initial simulation until the design stabilizes. Then more refined models are used and more detail added to the processing definitions.

6.2.6 Parallel Machine Coding Tools

After a candidate overall system design is reached, detailed coding for the actual machines can be performed. This code is fed back into the concurrent code analysis tool and allocations and process sequence graphs reverified for function and performance. This system engineering approach allows new architectures to be evaluated and also allows measurement and evaluation of the impact of changes in requirements. In addition, incremental and phased development of systems using mixtures of codes represented by high level models of their future implementation and new parallel codes. Care should be taken to provide a hierarchy with machine independent, architecture type, and machine-specific layers.

These tools are very similar in approach and levels of complexity to those used in VLSI circuit design computer-aided engineering. The difference of a more complex input is offset by adding the concurrent code analyzer tool. Note the assumption that the outcome of the system design results in the proper code, control, and data granularity for operation within the constraints of the network and its controlling operating system or language.

The detailed coding of each process in the application should be allowed in different languages. The best concurrent expression of a process is in a concurrent language without artificial constraints caused by sequential languages and the use of extensions. Because there is no standard concurrent language programming which is common to each architecture and its specialized mode of operation, the user must be able to use available languages on the machines in his repertoire. The network design must provide interfaces to languages already developed for the processor. Examples of these languages are

C * and *Lisp (Connection Machine)
Fortran (BB & N Butterfly)
Signal processing macros (Warp)
Occam (Transputer based parallel processors)
Ada (Alliant)
Pacific Sierra preprocessors for Fortran (Cray)

6.3 Algorithm Development

Algorithm development and tuning are among the crucial elements of parallel application development. **The largest gains are to be found in the discovery of new algorithms which express concurrency.** Hardware and software development follow thereafter. The gains provided by the struggle to program parallel machines will be overshadowed by insightful algorithms which provide a concurrent view of the application. However, one cannot rely upon timely progress and the state of mind necessary for development cannot be attained, without attempting to perform the parallel application. Abstraction, portability, fault tolerance, recovery, tools for visualization of code processes, and configuration management are needed.

The need for abstraction is noted as a desirable feature, but the need to understand the architecture when programming remains a necessity. The goal is to at least isolate the system developers and signal processing users from the details of the parallel architecture, leaving machine specific coding to a team of specialists. Tools for debugging, run profiling, instrumentation display in three-dimensional graphics, and retracing operations in fixed sequence are needed. Results must be deterministic in the debugging mode with software and hardware paths made repeatable.

The hardware design should include the concept of instrumented architectures to display profiling and debugging results, including all queues, functional unit utilitization, switch utilitization, memory accesses and patterns, feedback of concurrency destroying processes, and serial bottlenecks. Instrumented architectures meeting these criteria must be provided with no overhead to the parallel process under computation. A hardware design similar to the serial diagnostics chain provided by some systems could be easily implemented to independently (orthogonally) present machine performance information on a monitor window.

6.4 Requirements for Parallel Processing Tool Environment

6.4.1 Ideal Parallel Programming Environment

The user must be able to express the problem in terms of natural independent processes (unrelated to processors!) which best represent the concurrency available. Advocates of the conventional processor strategy are hopeful that combining, chunking, or gluing compilers will be forthcoming, because these can automatically convert natural level concurrency into coarse grain processor code (See [Fox]).

'' *The major issue is that of partitioning a problem into many processes that can be executed in parallel on an MIMD computer. For a small number of processors, say two to four processors, this problem is not a significant one because the parallelism available is not significant. For several processors, say sixteen or thirty-two, the problem is extremely difficult. Programs without a specific iterative structure are seldom so compl x that they have sixteen to thirty-two distinct sub-processes. Programs with an iterative structure are likely to be better suited to SIMD computers and execute with somewhat lower efficiency on MIMD computers because of resource allocation and synchronization overhead.* (From [Stone])''

6.4.2 Data Decomposition

Data decomposition into an even distribution across the computer is used when the calculation is based on a *large static data structure and the amount of work is about the same for each data element*. Even small differences in the computational load may result in idle waiting of many processors. A small fraction of this waiting can significantly impact the overall speed up. Amdahl's law is never beaten, only avoided by careful balancing and scheduling and by increasing the size of the parallel computation. Data decomposition is typically applied to highly structured numerical computations.

6.4.3 Control Decomposition

At the other extreme is the decomposition required when data structures are irregular and control is unpredictable because each portion has widely varying run times. Control decomposition can be any of the following types:

Manager-worker
Large grain pipelines
distributed blackboards
Functional

In the manager-worker control, a manager node is programmed to maintain a global data structure, monitor and track subprograms, and assign tasks to workers. Worker nodes request work when finished performing their indicated tasks.

Large grain pipelines are organized around stages which receive data from the previous stage in the pipe, operate on the data using a coarse grain code or systolic array approach, and pass the resulting data to the next stage.

Functional decomposition is the matching of the functions and communication organization of the algorithm by physical location of processors and communication lines.

6.4.4 Performance Tuning

The issues of performance tuning are load balance, communication ratio (granularity), sequential bottlenecks, and synchronization.

Load balance is the degree to which all processors remain active during the progress of the application.

Communications ratio is the ratio of communication time to computation time (Code granularity). Because the weak link in many parallel machines is the communication between processors it is important that the overhead of communication does not cause processors to be idle waiting. The smaller the ratio the coarser the code granularity and the higher the efficiency and performance.

Sequential bottlenecks are points in the computation where all processors have to wait for a single processor (or other part of the system, e.g. the network) (Control granularity). Any bottleneck dramatically impacts performance in parallel systems.

Synchronization is required when processes must come together and coordinate their activities. The use of mega synchronizations per second as a rating is advocated by [Stone].

(1) Load balance improvement techniques

Decrease and make grain size more uniform
Dynamically redistribute data structures or tasks
Redistribution of static data structures
Increase degree of multiprocessing per node

(2) Communication/computation ratio improvement techniques

Increase grain size
Restructure for fewer but larger messages
Combine multiple logical messages into a single message

(3) Sequential bottleneck improvement techniques

Modify or reorder algorithm to overlap sequential code with other computations
Spread work of overloaded processors among several processors

(4) Synchronization improvement techniques
Send needed values as soon as possible
Reorder or modify algorithm to eliminate synchronization where not needed

The key to the approach to MIMD mapping is recognizing the conflict between load balancing, that is, requiring decreased uniform grains, and communication/computation ratio minimization, that is, requiring fewer and larger grains and messages. The solution to this dilemma is the gluing compiler, where grains of the smallest sizes are programmed and compiled to the intermediate stage. The intermediate compiler structure is in Directed Acyclic Graphs or DAG form where independent code blocks and their relationships are developed and stored. The gluing compiler then forms the correct sizes for mapping the architecture. At this level of the compiler a standard parallel computer intermediate form could be defined which would allow each manufacturer to produce his own back-end. This standard would allow application and library software vendors the same target. Users would then be able to write their applications in the most concurrent possible form, then adapt and compile that code for a series of different parallel machines, preserving their effort on the science, numerical methods, algorithms, and specification-level language work. Portability could be achieved in this manner. The gluing compiler optimizes for a particular machine by aggregating together the small code blocks into the proper sizes and grains necessary for good operation on the selected machine.

Appendix A - Definitions and General References

Appendix A - Definitions and General References

These basic definitions define most of the terms of parallel supercomputers. These references were used to develop these definitions: [Stone], [Desrochers], and [Fox].

Actor - A processor which consists of operations, firing rules, and associated objects and states. (in programming)

Algorithmic overhead - The overhead associated in dividing a problem into parallel parts, such as the extra calculations performed at each processor which are not necessary in the sequential algorithm.

Amdahl's Law - The maximum speedup attainable by an algorithm and its implementation on a computer is dominated by its sequential components. *Performance will be dominated by the amount of work which must be done for the slowest mode of execution in the application. (See [Martin] and Martin [ARCH5].*

Balance - When processor, memory, interconnection network, synchronization, an I/O bandwidth are arranged so that no one of them strongly dominates the system throughput.

Barrier synchronization - Halting a set of processes until every process in the associated set has completed.

Breakeven point - The number of processors required in a multiprocessor to match the performance of a single processor of the same power.

Chunksize - The number of iterations or operations to be grouped as a single task to increase task code grain size or granularity.

Coarse-grain parallelism - Parallel execution in which the amount of computation is many times larger than the overhead and communication expended per task.

Conflict - The condition which exists when two or more operations require the same system resource, such as functional unit, memory bank, or network port.

Contention - The interference among tasks competing for the same resource resulting in idle task waiting.

Context switch - The process of saving the state of one task and restoring another task, thus changing the execution from one task to another.

Critical Section - A section of a program which may be executed by only one process at a time.

Data flow - The sequence of processes and data transmissions to be performed on a data set.

Deadlock - The state which exists when parallel processing elements wait for events which never occur.

do par - A program statement that permits iterations of a loop to be executed in parallel.

do seq - A program statement that forces iterations of a loop to be sequential.

Domain - A set of objects or data which define the scope of a problem. A set of grid points or a set of bodies are examples of domains. Domains are split into grains, granules and members.

Domain decomposition - The division of domains into parts, one grain per processor or memory.

Domain of processing - A mode of processing, for example, highly structured numeric computation.

Efficiency - The speedup per node.

Explicit parallelism - Parallelism that is purposely constructed from sections of a program and defined in detail.

Fine-grain parallelism - Typically refers to SIMD single-bit control granularity. (See granularity discussion at end of this Section).

Grain - A subdivision of a domain to be handled by a single node. Code, Control, and Data grain are the different types of grain. (See discussion at the end of this Section).

Grain size - The number or length of fundamental entities or members in a grain.

Granularity

The concept of grain describes program attributes which can be used effectively by each parallel machine category [Murphy]. The forms of grain are the following:

(1) Code Grain - the size of code segments capable of being run with communications and synchronization overhead remaining insignificant

(2) Control Grain - the degree, number, and flexibility of instructions which can be run in a given parallel machine cycle

(3) Data Grain - the relative association of data elements with each other which can be treated within the parallel machine

Code grain varies through the levels of the following sizes:

Coarse - More than 100 instructions
Large (or Medium) - approximately 100 instructions
Natural - around 10 instructions
Cycle - a single floating point instruction
Fine - 1-bit operations

The most general purpose machine would be able to handle all code grain size equally effectively. When MIMD machines are being used, the term coarse grain typically refers to coarse code grain.

Control grain varies from fine (one-bit) for SIMD machines, to Massive (N, the number of independently-controllable, computational-functional units in a fully synchronized, conflict free, and latency buffered parallel machine). The most general purpose machine would handle all control grains equally well, with ''massive'' being ideal. When SIMD machines are being discussed the term ''fine grain'' refers to control grain. Note that SIMD machines require coarse grain typically refers to coarse code grain.

Control grain varies from fine (one-bit) for SIMD machines, to Massive (N, the number of independently-controllable, computational-functional units in a fully synchronized, conflict free, and latency buffered parallel machine). The most general purpose machine would handle all control grains equally well, with ''massive'' being ideal. When SIMD machines are being discussed the term ''fine grain'' refers to control grain. Note that SIMD machines require coaunit. Typically the same as the grain.

Hypercube - The method of interconnection that treats individual processors as nodes of a multidimensional cube.

Implicit parallelism - The parallelism that is embedded in the normal structure of a program

Inhomogeneous Problem - Problems with domains in which members are of different types.

Interprocessor communication - The transmission of data and control information between multiple processors.

Irregular problem - A problem concerunit. Typically the same as the grain.

Hypercube - The method of interconnection that treats individual processors as nodes of a multidimensional cube.

Implicit parallelism - The parallelism that is embedded in the normal structure of a program

Inhomogeneous Problem - Problems with domains in which members are of different types.

Interprocessor communication - The transmission of data and control information between multiple processors.

Irregular problem - A problem concerors and their local memories, with a communications network providing control and data flow between the computers.

Multigrid method - The solution to partial differential equations in which a coarse grid is used to obtain an improved solution on a finer grid. Iterations among a hierarchy of coarse and fine grids can reduce error.

Multiprocessor - A parallel computer which is composed of multiple processors, shared memory, and facilities for their interaction and cooperation.

Nearest-neighbor interconnection - An interconnection structure that connects each processor to its nearest four neighbors in a rectangular grid.

Node - The elements of a parallel processor (processor and/or memory, and switch) located at the vertices of the interconnection system.

Overhead - Total overhead is the performance difference between linear speedup and that achieved by N processors. Overhead is due to algorithmic, software, load balancing, communication, scheduling, and context switching.

Recurrence relation - The relation in which items in a sequence are expressed in term of previous items in the sequence.

Scalar - An operation which manipulates individual data elements, completing each operation before starting any part of another.

Serial time - The time taken to execute an algorithm on a serial (single processor) computer.

Shared memory - A common memory in a multiprocessor which allows each processor to access memory locations of any other processors, typically through a memory access (using an interconnection network) to communicate between processes.

Small (Fine) grain size - Processors with memories around 1000 bytes per processing node, typically SIMD 1-bit operation; another definition is decomposition into small groupings.

Speedup - The ratio of time required to execute an efficient serial algorithm to the time required to execute a parallel version of the algorithm on a number of processors identical to the single processor. Often plotted as a function of the number of processors.

Synchronization - An operation in which two or more processes exchange information to coordinate their activities.

Utilitization - The fraction of a system's total resources that is being used.

Virtual concurrent processing - The support of more nodes in the problem set up than in the computer hardware.

General References

[Fox] Fox, G.C., et al., **Solving Problems on Concurrent Processors**, Volume I, General Techniques and Regular Problems,'' Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[Stone] Stone, H.S., **High-Performance Computer-Architecture**, Addison-Wesley, 1987. Pg. 383

[Martin] J.L. Martin, ''Mapping Applications to Architectures,'' **Conference Proceedings: Supercomputing '87, Volume I Industrial Systems, Prototype Architectures, and Supercomputer Projects**, 1987, pg. 475. Editors, L.P Kartashev and S.I. Kartashev, International Supercomputing Institute, St. Petersberg, Fla. 1987.

[Desrochers] Desrochers, G.R., **Principles of Parallel and Multiprocessing**, Intertext Publications, McGraw-Hill Book Co., New York, 1987.

[Murphy] Murphy C.G., **Specialized Parallel Processor**, Research Consortium, Inc., Minneapolis, MN, 1989.
-and-
Mapping Applications to Architectures, Research Consortium, Inc., Minneapolis, MN, 1989.

[Kumar] '', M. Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications,'' IEEE Trans. on Computers, 37(9), pp. 1088-98, Sep 88 (This is [SUP 20].)

APPENDIX B

SURVEY REFERENCES

AUTHOR	TITLE JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
.Cover ()	Supercomputing '88 Nov. 1988 Orlando IEEE Computer Society (1988)			
LAGER, D.L. [ALG 1]	SIG--A General Purpose Signal Processing Program Proceedings: IEEE	75(9)	1322	Sep 1987
PURTILO, J.M. [ALG 2]	Environments for Prototyping Parallel Algorithms Journal for Parallel & Distributed Computing	5	421-37	(1988)
BOKHARI, S.H. [ALG 3]	Partitioning Problems in Parallel, Pipelined, and Distributed Computing IEEE Trans. on Computers	37(1)	48-57	Jan 88
JAMIESON, L.H. [ALG 4]	Features of Parallel Algorithms PWC: Supercomputing '87-1	476-78	(1987)	
CHEN, G.H. [ALG 5]	Designing Parallel Algorithms from Forests and Multistage Graphs Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88262			
FRIEZE, A.M. [ALG 6]	Algorithms for Assignment Problems on an Array Processor Parallel Computing	11	151-62	(1989)
ENGSTROM, B.R. [ALG 7]	The SDEF Systolic Programming System Proc. 1988 Int. Conf. on Parallel Processing Pennsylvania State University Press	III	645	August 15-19, 1988
McCROSKY, C. [ALG 8]	Realizing the Parallelism of Array-Based Computation Parallel Computing	10	29-43	(1989)
STONE, H.S. [ALG 9]	Parallel Querying of Large Databases: A Case Study Computer		11	Oct 1987
O'HALLARON, D.R. [ALG 10]	Parallel Imp. of a Kalman Filter on the Warp Computer Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III Pennsylvania State University Press Aug 15-19, 1988		108-11	

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
ALEXANDER. W.E.	A New Approach to the Implementation of Multi- dimensional Signal Processing Algorithms			
[ALG 11]	Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III			
	Pennsylvania State University Press Aug 15-19, 1988		124-27	
LIN, W.	A Parallel Matrix Inversion Algorithm with Dynamic Communication Structures			
[ALG 12]	Proceedings: Third International Conference on Supercomputing - Supercomputing '88 Vol. II		460	
ARYA, S.	Parallel Algorithm Development Workbench			
[ALG 13]	Proceedings: Supercomputing '88 Nov. 14-18, 1988 Orlando			
	IEEE Computer Society		11-17	(1988)
FOX, G.C.	Matrix Algorithm on a Hypercube I: Matrix Multiplication			
[ALG 14]	Parallel Computing	4	17-31	(1987)
FENG, N.	A Shared Memory Algorithm and Performance Evaluation of the Generalized Alternative Construct in CSP			
[ALG 15]	Proceedings 1988 *International Conference on Parallel Processing* August 15-19, 1988 Vol. II		176	
ARMSTRONG, J.	A Multi-Algorithm Approach to Very High Performance One- Dimensional FFT's			
[ALG 16]	Journal of Supercomputing	2	415-33	(1988)
SWARTZTRAUBER, P.N.		Multi-Processor FFT's		
[ALG 17]	Parallel Computing	5	197-210	(1987)
	Parallel and Distributed Algorithms (Special Issue)			
[ALG 18]	IEEE Trans. on Computers	37(12)	1485-1676	Dec 88
DINNING, A.	A Survey of Synchronization Methods for Parallel Processors			
[ARCH 1]	Computer	22(7)	66-77	Jul 89
McBRYAN, O.A.	New Architecture: Performance Highlights and New Algorithms			
[ARCH 2]	Parallel Computing	7	477-99	(1988)
SNYDER, L.	A Taxonomy of Sychronous Parallel Machines			
[ARCH 3]	Proceedings: 1988 International Conference on Parallel Processing August 15-19, 1988 Vol. I			
	Pennsylvania State University Press		281	

AUTHOR	TITLE / JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
CASAVANT, T.L.	Experimental Parallel Processing Research on Reconfigurable SIMD and/or MIMD Computers			
[ARCH 4]	Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88267			
MARTIN, J.L.	Performance Evaluation: Applications and Architectures			
[ARCH 5]	Supercomputing '87	III	369-73	
HACH, J.J.	On the Promise of General-Purpose Parallel Computing			
[ARCH 6]	Parallel Computing	10	261-75	(1989)
HWANG, K.	Advanced Parallel Processing with Supercomputer Architectures			
[ARCH 7]	Proceedings IEEE	75(10)	1348-79	Oct 1987
HARP, J.G.	Esprit Project P1085 - Reconfigurable Transputer Project			
[ARCH 8]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. 1 California Institute of Technology Jan. 1988 122-127			
BRONNENBERG, W.	DOOM: An Object-Oriented Approach to Parallel Comput.ng			
[ARCH 9]	*Proceedings Int. Conf on Supercomputing* Supercomputing 1988 Vol. II International Supercomputing Institute 1988 - Vol. II		271	
TRELEAVAN, P.C.	Parallel Architecture Overview			
[ARCH 10]	Parallel Computing	8	59-70	(1988)
BISANI, R.	*A Tool to Coordinate Tools*			
[ENV 1]	*IEEE Software*		17-25	Nov 1988
DART, S. A.	Software Development Environments (Taxonomy)			
[ENV 2]	Computer		18	Nov 1987
REEVES, A.P.	Programming Environments for Highly Parallel Multi-processors			
[ENV 3]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. 1 California Institute of Technology Jan. 1988 458-467			
CARLE, A.	A Practical Environment for Scientific Programming			
[ENV 4]	Computer		75	Nov 1987

```
AUTHOR                                          TITLE
-----------------------------------------------------------------------------------------
     JOURNAL/PUBLISHER                          VOL (#)   PAGE          DATE
-----------------------------------------------------------------------------------------

SMITH, K. & APPELBE, W.                         PAT -- An Interactive FORTRAN Parallelizing Assistant Tool
[ENV 5]          Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III
                 Pennsylvania State University Press  Aug 15-19, 1988
                                                          58-62

GUARNA, V.A.     Faust: An Integrated Environment for Parallel Programming
[ENV 6]          IEE Software                             20-27         Jul 89

APPELBE, B.      Start/Pat: A Parallel-Programming Toolkit
[ENV 7]          IEEE Software                            29-38         Jul 89

.                Proceedings: Supercomputing '88 Nov. 14-18, 1988 Orlando
[ENV 8]          IEEE Computer Society                                  (1988)

GUARNA, V.A.     FAUST:An Environment for Programming Parallel Scientific Applications
[ENV 9]          Proceedings: Supercomputing '88 Nov. 14-18, 1988 Orlando
                 IEEE Computer Society                    3-10          (1988)

ERTEL, D.        Environment for Enhancing IPSC/2 Programmer Productivity
[ENV 10]         The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                 California Institute of Technology Jan. 1988  513

:Parasoft        A Portable Programming Environment for Parallel Computers - Parasoft Corporation (CROS)
[ENV 11]         Third Conference on Hypercube Concurrent Computers & Applications Vol I.
                 California Inst'tute of Technology  January 1988

PEINZE, K.       Tools for Concurrent Programming on SUPRENUM
[ENV 12]         Proceedings Int. Conf on Supercomputing
                 Supercomputing 1988 Vol. II
                 International Supercomputing Institute 1988 - Vol. II
                                                          28

DONGARRA, J.J.   A Portable Environment for Developing Parallel FORTRAN Programs
[ENV 13]         Parallel Computing              5        175-86        (1987)

PRATT, T.W.      The Pisces 2 Parallel Programming Environment
[ENV 14]         Proc. 1988 Int. Conf. on Parallel Processing
                 Pennsylvania State University Press  III  August 15-19, 1988
```

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
DONGARRA, J.J.	Schedule Tools for Developing and Analyzing Parallel Fortran Programs			
[ENV 15]	Argonne National Laboratory TM #86,			Nov 86
PIKE, R.	A Concurrent Window System			
[ENV 16]	USENIX Computing System	2	2	Spr 1989
			439	
GEHANI	Concurrent C++: Concurrent Programming with Class(es)			
[EXT_LANG 1]	Software Pract. & Exp.	18(12)	1157-77	Dec 88
SHIBAYAMA, E.	Program Transformation in an Object-Based Parallel Computing Model			
[EXT_LANG 3]	Program of Future Generation Computers II			
	Elsevier Science 1988 - Fuchi (Ed.)			
STEVENSON, D.E.	A Canonical Forum for Parallel Programs			
[EXT_LANG 4]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 536			
CHEN, M.	Crystal: From Functional Description to Efficient Parallel Code			
[EXT_LANG 5]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 417			
GROSSMAN, F.	A Language for High-Level Programming of Mathematical Applications			
[EXT_LANG 6]	Proceedings: 1988 Intl. Conference on Computer Languages			
	Miami IEE Computer Society Oct 9-13, 1988			
			31	
WHOLEY, S.	Connection Machine LISP: A Dialect of Common LISP for Data Parallel Programming			
[EXT_LANG 7]	Supercomputing '87	III	45-54	
FISHER, A.	Communication and Code Optimization in SIMD Programs			
[EXT_LANG 8]	Proceedings 1988 International Conference on Parallel Processing August 15-19, 1988 Vol. II			
			84	
RUPPELT, Th.	Automatic Transformation of High Level Object-Oriented Specifications into Parallel Programs			
[EXT_LANG 9]	Parallel Computing	10	15-28	(1989)
HALSTEAD, R.H.	Parallel Symbolic Computing			
[EXT_LANG 10]	Computer		35-43	Aug 1986

```
AUTHOR                                    TITLE
-------------------------------------------------------------------------------------------
    JOURNAL/PUBLISHER                     VOL (#)   PAGE          DATE
-------------------------------------------------------------------------------------------

FELTEN, E.W.       Coherent Parallel C
[EXT_LANG 11]      The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                   California Institute of Technology Jan. 1988  440

DALLY, W.J.        Object-Oriented Concurrent Programming in CST
[EXT_LANG 12]      The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                   California Institute of Technology Jan. 1988  434

ROSING, M.         DINO: Summary and Examples
[EXT_LANG 13]      The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                   California Institute of Technology Jan. 1988  472

WOLFE, M.          Multiprocessor Synchronization for Concurrent Loops
[EXT_LANG 14]      IEEE Software                                 34-42

ROBINET, B. (Editor)                      ESOP 86: European Symposium on Programming
[EXT_LANG 15]      (Saarbrucklakeen) Springer-Verlag             March 86

TRIOLET, R.        Automatic Parallelization of FORTRAN Porgrams in Presence of Procedure Calls
[EXT_LANG 16]      ESOP 86: European Symposium on Programming
                   (Saarbrucken) Springer-Verlag                 March 86

ZORN, B.           Multiprocessing Extensions in Spur Lisp
[EXT_LANG 17]      IEEE Software                                 41-49          Jul 89

ALLEN, F.          An Overview of the PTRAN Analysis System for Multi-processing
[EXT_LANG 18]      Supercomputing 1st International Conference - Athens
                   Springer-Vetlag Greece                                       Jun 87

CALLAHAN, D.       Compiling Programs for Distributed Memory Multi-processors
[EXT_LANG 19]      Journal of Supercomputing              2      151-69         (1988)

MEHROTRA, P.       The BLAZE Language: A Parallel Language for Scientific Programming
[EXT_LANG 20]      Parallel Computing                     5      339-61         (1987)

CARLTON, M.        A Distributed Prolog System With and Parallelism
```

```
AUTHOR                              TITLE
-----------------------------------------------------------------------------------------
    JOURNAL/PUBLISHER                   VOL (#)    PAGE          DATE
-----------------------------------------------------------------------------------------

[EXT_LANG 21]   IEEE Software                      43-51

CHEN, M.C.      Footprints of Dependency: Towards Dynamic Memory Management for Massively Parallel Architectures
[EXT_OS 1]      Proceedings:  Third International Conference on Supercomputing - Supercomputing '88 Vol. III
                                                   486-94

BECK, B.        A Parallel Programming Process Model
[EXT_OS 2]      IEEE Software                      63-72         May 1989

ELLIS, C.S.     Algorithms for Parallel Memory Allocation
[EXT_OS 3]      International Journal on Parallel Programming
                                         17(4)     303-45        (1988)

WOLFSTAHL, Y.   Mapping Parallel Programs to Multiprocessor: A Dynamic Approach
[EXT_OS 4]      Parallel Computing        10       45-50         (1989)

BAIN, W.L.      Interwork II Concurrent Programming Toolkit
[EXT_OS 5]      Third Conference on Hypercube Concurrent Computers & Applications Vol I.
                California Institute of Technology  January 1988
                                                   886

TOLLE, D.       A Collection of NCUBE UNIX Utilities
[EXT_OS 6]      Third Conference on Hypercube Concurrent Computers & Applications Vol I.
                California Institute of Technology  January 1988
                                                   832

ANGUS, I.G.     FORTRAN CUBIX: Definition and Implementation
[EXT_OS 7]      Third Conference on Hypercube Concurrent Computers & Applications Vol I.
                California Institute of Technology  January 1988
                                                   787

SCHWAN, K.      "TOPOLOGIES" - Computational Messages for Multi- computer
[EXT_OS 8]      The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                California Institute of Technology Jan. 1988  580

GAIT, J.        An Adaptive Two Tier Scheduler for Partitioned Multi-processing
[EXT_OS 9]      Proceedings: Third International Conference on Supercomputing - Supercomputing '88 Vol. III
                                                   144

SCHRODER, K.    PEACE: A Distributed Operating System for an MIMD Message-Passing Architecture
```

```
AUTHOR                                          TITLE
-----------------------------------------------------------------------------------------
   JOURNAL/PUBLISHER                            VOL (#)   PAGE          DATE
-----------------------------------------------------------------------------------------

[EXT_OS 10]       Proceedings Int. Conf on Supercomputing
                  Supercomputing 1988 Vol. II
                  International Supercomputing Institute 1988 - Vol. II
                                                          302

STEVENSON, D.E.   Consideration in System Language Design for Scientific Supercomputer Applications
[EXT_OS 11]       Proceedings:  Third International Conference on Supercomputing - Supercomputing '88 Vol. III
                                                          352

Le BLANC, T.J.    Crowd Control: Coordinating Processes in Parallel
[EXT_OS 12]       Proc. 1988 Int. Conf. on Parallel Processing
                  Pennsylvania State University Press  III  August 15-19, 1988
                                                          81

MALONEY, A.D.     MPF: A Portable Message Passing Utility for Shared Memory Multiprocessors
[EXT_OS 13]       Proc. 1988 Int. Conf. on Parallel Processing
                  Pennsylvania State University Press  III  August 15-19, 1988
                                                          739

VORNBERGER, O.    Parallel Prolog on a Local Area Network
[EXT_OS 14]       Proceedings:  Third International Conference on Supercomputing - Supercomputing '88 Vol. III
                                                          464

RANGOZAR, M.      A Proposal for Optimization of Distributed Concurrency        Control
[EXT_OS 15]       Proc. 1988 Int. Conf. on Parallel Processing
                  Pennsylvania State University Press  III  August 15-19        , 1988
                                                          93

GARG, V.K.        High-Level Communication Primatives for Concurrent Systems
[EXT_OS 16]       Proceedings: 1988 Intl. Conference on Computer Languages
                  Miami IEE Computer Society  Oct 9-13, 1988                    92

FLECKENSTEIN, C.J.Using a Global Name Space for Parallel Execution of UNIX Tools
[EXT OS 17]       Communications of the ACM          32(9)     1085-90       Sep 1989

BAALBERGEN, E.H.  Design and Implementation of Parallel Make
[EXT OS 18]       USENIX Computing Systems           1 (2)     135-158       Spr 1988

YAU, S.S.         Visual Languages and Software Specifications
```

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
[HCI 1]	Proceedings: 1988 Intl. Conference on Computer Languages Miami IEE Computer Society Oct 9-13, 1988		322	
KRISHNAMOORTHY, M.S. [HCI 2]	Program Tools for Algorithm Animation Software Prac. & Exper.	19(6)	505-13	Jun 1989
TOMHOULIAN, S. [HCI 3]	A Visual Programming Environment for the Navier-Stokes Computer Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III Pennsylvania State University Press Aug 15-19, 1988		67-71	
GANNON, D. [HCI 4]	The Sigma System: A Tool for Parallel Program Design Proceedings Int. Conf on Supercomputing Supercomputing 1988 Vol. II International Supercomputing Institute 1988 - Vol. II		157	
MYERS, B.A. [HCI 5]	User-Interface Tools: Introduction and Survey *IEEE Software*	6(1)	15-23	Jan 89
BAILEY, M.L. [HCI 6]	Debugging Parallel Programs using Graphical Views Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III Pennsylvania State University Press Aug 15-19, 1988		46-49	
BROWN, M.H. [HCI 7]	Exploring Alogrithms Using Balsa II Computer		14-36	May 1988
HSAI, Y.T. [HIC 8]	Programming through Pictorial Transformations Proc. 1988 Conference on Parallel Processing Pennsylvania State University	III		
AGHA [LANG 1]	Concurrent Programming Using Actors Object Oriented Concurrent Programming MIT PRESS 87			
DINITTO, S.A. [LANG 1]	Future Directions in Programming Languages Proceedings: 1988 Intl. Conference on Computer Languages Miami IEE Computer Society Oct 9-13, 1988			

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
			169	
PERROTT, R.H.	Supercomputer Languages			
[LANG 2]	Supercomputing '87	III	20-26	
TRIPATHI, A.	An Implementation of the Object-Oriented Concurrent Programming Language SINA			
[LANG 3]	Software Pract. & Exper.	19(3)	235-256	Mar 1989
SHAPIRO, E.	Concurrent Prolog: A Progress Report			
[LANG 4]	Computer		44-58	Aug 1986
BALDWIN, D.	Consul- A Parallel Constraint Language			
[LANG 5]	IEEE Software		62-69	Jul 89
TICK. E.	Comparing Two Parallel Logic-Programming Architectures			
[LANG 6]	IEEE Software		71-80	Jul 89
CLARK, K.L.	PARLOG and Its Applications			
[LANG 7]	IEEE Trans. on Software Engineering	14(12)	1797	(1988)
YAMAZAKI, K.	Low- and High-Level Parallelism for an Object-Oriented Language			
[LANG 8]	Proceedings: Third International Conference on Supercomputing - Supercomputing '88 Vol. III		383	
HANSEN, P.B.	A Multi-Porcessor Implementation of Joyce			
[LANG 9]	Software Prac. & Exper.	19(6)	579-92	Jun 1989
KARP, A.H.	A Comparison of 12 Parallel FORTRAN Dialects			
[LANG 10]	IEEE Software		57-67	Sep 1988
GELERNTER, D.	Domesticating Parallelism			
[LANG 11]	Computer		12-16	Aug 1986
MUNDIE, D.A.	Parallel Processing in Ada			
[LANG 12]	Computer		20-25	Aug 1986
GOLDMAN, R.	Qlisp: Parallel Processing in Lisp			
[LANG 13]	IEEE Software		51-59	Jul 89

```
AUTHOR                                  TITLE
------------------------------------------------------------------------------------------------

    JOURNAL/PUBLISHER                   VOL (#)   PAGE          DATE

------------------------------------------------------------------------------------------------

POLYCHRONOPOULOS, C.D.                  Compiler Optimizations for Enhancing Parallelism
[LANG 14]       IEEE Trans. on Computers, 37(8)       991       August 88

GUZZI, M.D.     Cedar FORTRAN and other Vector and Parallel FORTRAN Dialects
[LANG 15]       Supercomputing '88 Nov. 1988 Orlando
                IEEE Computer Society (1988)          114

GIRKAR, M.      Compiling Issues for Supercomputers
[LANG 16]       Supercomputing '88 Nov. 1988 Orlando
                IEEE Computer Society (1988)          164

WELCH, P.H.     An Occam Approach to Transputer Engineering
[LANG 17]       The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                California Institute of Technology Jan. 1988  138

CLAPP, R.M.     Ada on a Hypercube
[LANG 18]       The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                California Institute of Technology Jan. 1988  399-408

DeFOREST, D.    Hyperflow
[LANG 19]       The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                California Institute of Technology Jan. 1988  482

LAKE, T.        Language for Parallel Processing
[LANG 20]       Informationstechnik It                30        2             (1988)

JORDAN, H.F.    Programming Language Concepts for Multiprocessors
[LANG 21]       Parallel Computing                    8         31-40         (1988)

AHUJA, S.       Linda and Friends
[LANG 22]       Computer                                        26-34         Aug 1986

WHITESIDE, R.A. Using Linda for Supercomputing on a Local Area Network
[LANG 23]       Supercomputing '88 Nov. 1988 Orlando
                IEEE Computer Society (1988)          192

LUNBERG, L.     A Parallel Ada System on an Experimental Multiprocessor
[LANG 24]       Software-Prac. & Exper.               19(8)     787-800       Aug 1989

.COVER          The Design and Application of Parallel Digital Processor
```

AUTHOR	TITLE / JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
[LANG 26]	IEE The Gulbenkian Foundation, Lisbon 11-15 April 1988			
WATSON, I. [LANG 29]	Language First Philosophy IEE The Gulbenkian Foundation, Lisbon 11-15 April 1988		83-86	
DONGARRA, J.J. [LIB 1]	The LINPACK Benchmark: An Explanation Supercomputing 1st International Conference - Athens Springer-Vetlag Greece			Jun 87
SNELLING, D.F. [LIB 2]	A Comparative Study of Libraries for Parallel Processing Parallel Computing	8	255-66	(1988)
HAMMARLING, S. [LIB 3]	The NAG Library and Supercomputer Proceedings Int. Conf on Supercomputing Supercomputing 1988 Vol. II International Supercomputing Institute 1988 - Vol. II		329	
NICOL, D.M. [MAP 1]	Problem Size, Parallel Architecture, and Optimal Speedup Journal of Parallel and Distributed Computing	5	404-21	(1988)
NICOL, C.V. [MAP 1]	Problem Size, Parallel Architecture and Optimal Speedup Journal of Parallel & Distributed Computing	5	404-420	(1988)
KRUSKAL, C.P. [MAP 2]	On the Notion of Granularity Journal of Supercomputing	1	395-408	(1988)
McDOWELL, C.E. [MAP 3]	A Practical Algorithm for Static Analysis of Parallel Programs Journal of Parallel and Distributed Computing	6	515-36	(1989)
BERMAN, F. [MAP 4]	On Mapping Parallel Algorithms into Parallel Architecture Journal of Parallel and Distributed Computing	4	439-58	(1987)

```
AUTHOR                         TITLE
------------------------------------------------------------------------------------------
     JOURNAL/PUBLISHER                            VOL (#)   PAGE        DATE
------------------------------------------------------------------------------------------

CHERKASSKY, V.    Efficient Mapping and Implementation of Matrix    Algorithms on a Hypercube
[MAP 5]           Journal of Supercomputing 2                7-27     (1988)

MORENO, J.H.      Graph-based Partitioning of Matrix Algorithms for Systolic Arrays
[MAP 6]           Proceedings: 1988 International Conference on Parallel Processing August 15-19, 1988 Vol. I
                  Pennsylvania State University Press

KUMAR, V.K.P.     Mapping Two Dimensional Systolic Arrays for One-Dimensional Arrays and Applications
[MAP 7]           Proceedings: 1988 International Conference on Parallel          Processing August 15-19, 1988 Vol. I
                  Pennsylvania State University Press
                                                          39-

REDDY, A.L.N.     I/O Embedding in Hypercubes
[MAP 8            Proceedings: 1988 International Conference on Parallel Processing   August 15-19, 1988 Vol. I
                  Pennsylvania State University Press
                                                          331

DUBOIS, M.        Synchronization, Coherence and Event Ordering in Multiprocessors
[MAP 9]           Computer                        21        9           Feb 88

WU, M.Y.          A Programming Aid for Hypercube Architectures
[MAP 10]          Journal of Supercomputing       2         349-72      (1988)

MUHLENBEIN, H.    New Solutions to the Mapping Problem of Parallel Systems:  The Evolution Approach
[MAP 11]          Parallel Computing              4         269-79      (1987)

.BAILEY, D.H. (Editor)                  Proceedings of the 1988 International Conference on Parallel Processing Algorithms and
                                        Applications Vol. III
[MAP 12]          Pennsylvania State University Press  Aug 15-19, 1988

KIM, S.J.         A General Approach to Mapping Parallel Computation Upon Multiprocessor Architectures
[MAP 13]          Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III
                  Pennsylvania State University Press  Aug 15-19, 1988
                                                          1-8

McDOWELL, C.E.    Viewing Anomalous States in Parallel Programs
[MAP 14]          Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III
                  Pennsylvania State University Press  Aug 15-19, 1988
                                                          54-57
```

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
GREENBAUM, A.	Synchronization Costs on Multiprocessors			
[MAP 15]	Parallel Computing	10	3-14	(1989)
STOUT, Q.F.	Mapping Vision Algorithms to Parallel ArchitectureS			
[MAP 16]	Proceedings IEEE	76(8)	982-95	Aug 1988
.KARTASHEV, L.P.	Supercomputing '87 Proceedings: Second International Conference on Supercomputing			
[MAP 17]	International Supercomputing Institute, Inc. 1987			
MARTIN, J.L.	Mapping Applications for Architectures			
[MAP 18]	PWC: Supercomputing '87-I	475	(1987)	
FOX, G. (Editor)	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
[MAP 19]	California Institute of Technology Jan. 1988			
FOX, G.C.	Hypercube Algorithms for Neural Network Simulation. The Crystal...			
[MAP 20]	Third Conference on Hypercube Concurrent Computers & Applications Vol I.			
	California Institute of Technology January 1988			
FOX, G.C.	Load Balancing Loosely Synchronous Problems with a Neural Network			
[MAP 21]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 241			
SALMON, J.	A Mathematical Analysis of the Scattered Decomposition			
[MAP 22]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 239			
PETTEY, C.C.	Parallel Placement of Parallel Processes			
[MAP 23]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 232			
LIVINGSTON, M.	Distributing Resources in Hypercube Computers			
[MAP 24]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 222			
ERCAL, F.	Task Allocation on to a Hypercube by Recursive Minicut Bipartitioning			
[MAP 25]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I			
	California Institute of Technology Jan. 1988 210			

```
AUTHOR                          TITLE
-------------------------------------------------------------------------------------------
     JOURNAL/PUBLISHER          VOL (#)    PAGE          DATE
-------------------------------------------------------------------------------------------

CHEN, W.K.          A Graph-oriented Mapping Strategy for a Hypercube
[MAP 26]            The Third Conference on Hypercube Concurrent Computers & Applications Vol. I
                    California Institute of Technology Jan. 1988  200

BELL, J.            Data Management for Parallel Processing of Large Scale Scientific Programs
[MAP 27]            PVC:  Supercomputing '87-I          479        (1987)

KRUATRACHUE, B.     Grain Size Determination for Parallel Processing
[MAP 28]            IEEE Software                                  23-33          Jan 88

RAMANUJAN, J.       Task Allocation by Simulated Annealing
[MAP 29]            Proceedings:  Third International Conference on Supercomputing - Supercomputing '88 Vol. II 471

KARTASHEV, L.       Proceedings: Third International Conference on
[MAP 30]            Supercomputing - Supercomputing '88 Vol. III

KRAMER, O.          Mapping Strategies in Message-based Multiprocessor
[MAP 31]            Parallel Computing                  9          213-25         (1988/89)

MUSCIANO, A.J.      Efficient Dynamic Scheduling of Medium-Grained Tasks for General Purpose Parallel Processing
[MAP 32]            Proceedings 1988 International Conference on Parallel Processing  August 15-19, 1988 Vol. II
                                                                   166

WEISS, M.           Dynamic Scheduling and Memory Management for Parallel Programs
[MAP 33]            Proceedings 1988 International Conference on Parallel Processing  August 15-19, 1988 Vol. II
                                                                   161

PEIS, J.K.          Minimum Distance: A Method for Partitioning Recurrences for Multiprocessors
[MAP 34]            Proc. 1988 Int. Conf. on Parallel Processing
                    Pennsylvania State University Press  III  August 15-19, 1988
                                                                   217

BRIGGS, F.A.  (Editor)                  Proceedings: 1988 International Conference on Parallel Processing August 15-19, 1988 Vol. I
[MAP 35]            Pennsylvania State University Press

MISSIRLIS, N.M.     Scheduling Parallel Iterative Methods on Multi-processor Systems
[MAP 36]            Parallel Computing                  5          295-302        (1987)
```

AUTHOR	TITLE JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
KAPENGA, J.A.	A Parallelization of Adaptive Task Partitioning Algorithms			
[MAP 37]	Parallel Computing	7	211-25	(1988)
BISIANI, R.	Multilanguage Parallel Programming of Heterogeneous Machines			
[MAP 38]	IEE Trans. on Computers	37(8)	930-45	Aug 88
COLIN, J.Y.	Allocating Tasks on a Virtual Distributed System			
[MAP 39]	Proceedings: Third International Conference on Supercomputing - Supercomputing '88 Vol. III		481	
ROSENBERG, J.B.	Mapping Massive SIMD Parallelism onto Vector Architectures for Simulation			
[MAP 40]	Software Prac. & Exper.	19(8)	739-56	Aug 89
DE JONG, V.J.	Symbolic Dimension Bound Checking in a Matrix Language			
[MAP 41]	Proc. 1988 Conference on Parallel Processing Pennsylvania State University Press, Vol. III			
KATSEF, H.P.	Using Data Partitioning to Implement a Parallel Assembler			
[MAP 42]	*Proc. SIGPLAN '88*		66	
BRADLEY, D.K.	Picasso: An Experiment in Hypercube Operating Systems Design			
[OS 1]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I California Institute of Technology Jan. 1988 364			
KRUMME, D.W.	The SIMPLEX Operating System			
[OS 2]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I California Institute of Technology Jan. 1988 381			
SALMON, J.	MOOSE: A Multi-Tasking Operating System for Hypercubes			
[OS 3]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I California Institute of Technology Jan. 1988 391-96			
PIERCE, P.	The NX/2 Operating System			
[OS 4]	The Third Conference on Hypercube Concurrent Computers & Applications Vol. I California Institute of Technology Jan. 1988 384-90			
.STURGIS (Editor)	Cover	Vol. II Software		
[PROG 1]	Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III Pennsylvania State University Press Aug 15-19, 1988			

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
GOKHALE, M.B.	Programming in a Very High Level Data Flow Language			
[PROG 1]	Proceedings: Third International Conference on Supercomputing - Supercomputing '88 Vol. III		366	
BROWNE, J.C.	Code: A Unified Approach to Parallel Programming			
[PROG 3]	IEEE Software		10-18	Jul 89
NEVES, K.W.	Growing Discord: Programming Philosophy and Hardware Design			
[PROG 4]	Proceedings: Supercomputing '88 Nov. 14-18, 1988 Orlando			
	IEEE Computer Society		18	(1988)
DEMARCO, T.	On Parallel Software: An Interview			
[PROG 5]	IEEE Software		92-93	Mar 1988
CAVANO, J.P.	Software Developments Issues for Parallel Processing			
[PROG 6]	Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88299			
CAVANO, J.P.	Software Issues Facing Parallel Architecture			
[PROG 7]	Software Developments Issuer for Parallel Processing			
	Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88300			
RUSSELL, L.	Software Development Issues for Parallel Processing			
[PROG 8]	Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88306			
FOX, G.C.	Issues in Software Development for Concurrent Computer			
[PROG 9]	Proceedings Comput·r Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88302			
CHANDY, K.M.	Architecture Independent Programming (UNITY)			
[PROG 10]	Proceedings: Third International Conference on Supercomputing - Supercomputing '88 Vol. III		345	
SOBEK, S.	Architecture and Language Independent Parallel Programming			
[PROG 11]	Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III			
	Pennsylvania State University Press Aug 15-19, 1988	80		
NICOL, D.M.	Problem Size, Parallel Architecture, and Optimal Speedup			
[PROG 12]	Proc. 1988 Int. Conf. on Parallel Processing			
	Pennsylvania State University Press III August 15-19, 1988			

```
AUTHOR                              TITLE
------------------------------------------------------------------------------------------
    JOURNAL/PUBLISHER                               VOL (#)   PAGE          DATE
------------------------------------------------------------------------------------------

                                                              347

CHANDY, K.M.     Programming Parallel Computers
[PROG 13]        Proceedings: 1988 Intl. Conference on Computer Languages
                 Miami IEE Computer Society  Oct 9-13, 1988
                                                              314

SABOT, G.        The       ation Model
[PROG 14]        The MIT Press                                              1988

HUDAK, P.        Exploring Parafunctional Programming  Separating
[PROG 15]        The What From the How
                 IEEE Software                                54-61

HUDAK, P.        Para-functional Programming
[PROG 16]        Computer                                     60-70         Aug 1986

FOX, G.C.        Solving Problems on Concurrent Processors Vol. I General Techniques and Regular Problems
[PROG 17]        Prentice Hall 1988

KALLSTROM, M.    Programming Three Parallel Computers
[PROG 18]        IEEE Software                                11-22         Jan 88

HEY, A.J.G.      Practical Parallel Processing with Transputer
[PROG 19]        The Third Conference o  Hypercube Concurrent Computers & Applications Vol. I
                 California Institute  / Technology Jan. 1988  115

.                Proceedings: 1988 Intl. Conference on Computer Languages
[PROG 20]        Miami IEE Computer Society  Oct 9-13, 1988

BAGRODIA, R.     Programming The Connection Machine
[PROG 21]        Proceedings: 1988 Intl. Conference on Computer Languages
                 Miami IEE Computer Society  Oct 9-13, 1988
                                                              50-57

McBRYAN, O.A.    The Connection Machine:  PDE Solution on 65 536 Processors
[PROG 22]        Parallel Computing                 9          1-24          (1988/89)

BERSHAD, B.N.    PRESTO: A System for Object Oriented Parallel Programming
```

AUTHOR	TITLE			
	JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
[PROG 23]	Software Pract. & Exp.	18(8)	713-32	Aug 88
BVANDES, T.	Realization of an Knowledge-Based Parallelization Tool in a Programming Environment			
[PROG 24]	Supercomputing 1st International Conference - Athens			
	Springer-Vetlag Greece			Jun 87
ZIMA, H.P.	SUPERB: A Tool for Semi-Automatic MIMD/SIMD Parallelization			
[PROG 25]	Parallel Computing	6	1-18	(1988)
PERCUS, O.E.	Random Number Generators for MIMD Parallel Processors			
[PROG 26]	Journal of Parallel and Distributed Computing			
		6	477-97	(1989)
STOROY, S.	Holistic Algorithms: A Paradigm for Multi-Processor Programming			
[PROG 27]	Parallel Computing	10	221-29	(1989)
KARP, A.H.	Programming for Parallelism			
[PROG 28]	Computer		43-57	May 1987
POLYCHRONOPOULOS, C.D.		Utilizing Multidimensional Coop Parrallelism on large-scale Parallel Processor Systems		
[PROG 29]	IEEE Trans. on Computers, 38(9)	1285	Sept 89	
OLDEHOEFT, R.R.	Applicative Parallelism on a Shared Memory Multi- processor			
[PROG 30]	IEEE Software		62-70	
HOUSTIS (Editor)	Supercomputing 1st International Conference - Athens			
[PROG 31]	Springer-Vetlag Greece			Jun 87
SOLCHENBACH, K.	Parallel Multigrid Methods: Implementation on SUPRENUM-like Architectures and Applications			
[PROG 32]	Supercomputing: 1st Intn. Conf., Athens, June 1987, Springer-Verlag			
DONGARRA, J.J.	Programming Methodology and Performance Issues for Advanced Computer Architecture			
[PROG 33]	Parallel Computing	8	41-58	(1988)
JAYASIMHA, D.N.	Parallel Access to Synchronization Variables			
[PROG 34]	Proc. 1988 Int. Conf. on Parallel Processing			
	Pennsylvania State University Press III	August 15-19, 1988		
			97	
FRANCIS, R.	An Evaluation of Parallel Procedure Calls			

```
AUTHOR                                       TITLE
-----------------------------------------------------------------------------------------------
   JOURNAL/PUBLISHER                         VOL (#)   PAGE        DATE
-----------------------------------------------------------------------------------------------

[PROG 35]          Proc. 1988 Int. Conf. on Parallel Processing
                   Pennsylvania State University Press  III  August 15-19. 1988
                                                               663

EISENSTADTER, Y.   Exploiting Locality of Reference in MIMD Parallel Symbolic Computation
[PROG 36]          Proc. 1988 Int. Conf. on Parallel Processing
                   Pennsylvania State University Press  III August 15-19, 1988
                                                               742

WEIHL, W.E.        Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types
[PROG 37]          ACM Trans. on Programming Language & Systems
                                                     11(2)     249-82      Apr 1989

BASTANI, F.        Efficient Abstract Data Type Components for Distributed and Parallel Systems
[PROG 38]          Computer                                    33          Oct 1987

MARTIN, B.         PARPC: A System for Parallel Procedure Calls
[PROG 39]          Proc. 1988 Int. Conf. on Parallel Processing
                   Pennsylvania State University Press  III  August 15-19, 1988
                                                               449

PARKINSON, D.      Organizational Aspects of Using Parallel Computer
[PROG 40]          Parallel Computing                5         75-83       (1987)

TERRANO, A.E.      Using an Architectural Knowledge Base to Generate Code for Parallel Computers
[PROG 41]          Communications of the ACM         32(9)     1065-72     Sep 89

PREISS, B.R.       Semi Static Dataflow
[PROG 42]          Proc. 1988 Conference on Parallel Processing
                   Pennsylvania State University Press      III       127-34      Aug 88

McGREGOR, J.D.     Support for Multiprocessing
[PROG] 43          Ed. Introduction Communications of the ACM      32(9)       1062-64Sep 89

AMMAR, R. A.       A Technique to Derive the Detailed Time Costs of Parallel Computations
[SIM 1]            Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88113-119

BAIN, W.L.         Hyperion: A Hypecube Simulator for Parallel Systems Performance Modeling
[SIM 2]            Third Conference on Hypercube Concurrent Computers & Applications Vol 1.
                   California Institute of Technology  January 1988
```

```
AUTHOR                                          TITLE
--------------------------------------------------------------------------------------------------
     JOURNAL/PUBLISHER                          VOL (#)   PAGE          DATE
--------------------------------------------------------------------------------------------------
                                                          792
YODER, M.A.        Simulation of a Word Recognition System on Two Parallel Architectures
[SIM 3]            IEEE Trans. on Computers     38(9)     1269-84       Sep 1989

RAMAMOORTHY, C.V.  Synthesis Rules for Cyclic Interaction Among Processes in Concurrent Systems
[SIM 4]            Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88497

YAU, Y.            Extensions on Performance Evaluation Techniques for Concurrent Systems
[SIM 5]            Proceedings Computer Software & Applications Conference 5-7 October, 1988 Chicago COMPSAC '88480

CHUNG, W.H.        Parallel Execution Schemes in a Petri Net
[SIM 6]            Proceedings: 1988 International Conference on Parallel Processing  August 15-19, 1988 Vol. I
                   Pennsylvania State University Press
                                                          286
KRAUSS, K.G.       A Petri Net Method for the Formal Verification of Parallel Processes
[SIM 7]            Proceedings 1988 International Conference on Parallel Processing  August 15-19, 1988 Vol. II
                                                          157
STOUTS, P.D.       The PFG Language: Visual Programming for Concurrent Computation
[SIM 8]            Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III
                   Pennsylvania State University Press  Aug 15-19, 1988

MURA, G.S.         PNSOFT: A Menu-Driven Software Package for Petri-Net Modeling and Analys's
[SIM 9]            Proceedings: 1988 Intl. Conference on Computer Languages
                   Miami IEE Computer Society  Oct 9-13, 1988
                                                          41
BRAY, B.           The Computer Architect's Workbench
[SIM 10]           Supercomputing '88 - Poster Session

LOPRIORE, L.       A User Interface Specification for a Program Debugging & Measuring Environment
[SUP 1]            Software Prac. & Exper.      19(5)     437-60        May 1989

MARTIN, J.         Supercomputer Performance Evaluation:  Status and Direction
[SUP 2]            Journal of Supercomputing 1  87-104    (1987)

GUPTA, R.          Debugging Code Reorganization by a Trace Scheduling Compiler
```

```
AUTHOR
                                        TITLE
------------------------------------------------------------------------------------------
     JOURNAL/PUBLISHER
                                        VOL (#)    PAGE          DATE
------------------------------------------------------------------------------------------
[SUP 3]         Proceedings:  Third International Conference on Supercomputing - Supercomputing '88 Vol. III
                                                           422

MILLS, R.C.     A Debugging Environment for the DADO Parallel Computer
[SUP 4]         Proceedings:  Third International Conference on Supercomputing - Supercomputing '88 Vol. III
                                                           413

REEVES, A.P.    On Measuring the Performance of a Massively Parallel Processor
[SUP 5]         Proceedings: 1988 International Conference on Parallel          Processing August 15-19, 1988 Vol. I
                Pennsylvania State University Press
                                                           261

HOUGH, A.A.     Belvedere: Prototype of a Parallel Pattern-Oriented Debugger for Highly Parallel Computation
[SUP 6]         Proc. 1988 Int. Conf. on Parallel Processing
                Pennsylvania State University Press  III  August 15-19, 1988
                                                           735

BURKHART, H.    Performance Measurement Tools in a Multiprocessor Environment
[SUP 7]         IEEE Trans. on Computers             38(5)      725-37         May 1989

CALLAHAN, D.    Vectorizing Compilers:  A Test Suite and Results
[SUP 8]         Proceedings: Supercomputing '88 Nov. 14-18, 1988 Orlando
                IEEE Computer Society                          98          (1988)

EAGER, D.L.     Speedup Versus Efficiency in Parallel Systems
[SUP 9]         IEEE Trans. on Computers 38(3)       408-422    March 89

PAN, W.M.       A Concurrent Debugger for IPSC/2 Programmer
[SUP 10]        Third Conference on Hypercube Concurrent Computers & Applications Vol I.
                California Institute of Technology  January 1988
                                                           823

BOHN, A.P.W.    Tools for Performance Evaluation of Parallel Machines
[SUP 11]        Supercomputing 1st International Conference - Athens
                Springer-Vetlag Greece                                     Jun 87
                                                212

FLOWER, J.      Parallel Programming in Comfort
[SUP 12]        Third Conference on Hypercube Concurrent Computers & Applications Vol I.
                California Institute of Technology  January 1988
```

```
AUTHOR                              TITLE
--------------------------------------------------------------------------------------------
    JOURNAL/PUBLISHER               VOL (#)    PAGE        DATE
--------------------------------------------------------------------------------------------
                                                811

McGUIRE, P.J.   A Measurement-Based Study of Concurrency in a Multi-Processor
[SUP 13]        Proc. 1988 Int. Conf. on Parallel Processing
                Pennsylvania State University Press  III  August 15-19, 1988
                                                85

SO, K.          A speedup Analyzer for Parallel Programs
[SUP 14]        Proc. 1988 Int. Conf. on Parallel Processing
                Pennsylvania State University Press  III  August 15-19, 1988
                                                653

ALLEN, T.R.     Debugging FORTRAN on a Shared Memory Machine
[SUP 15]        Proc. 1988 Int. Conf. on Parallel Processing
                Pennsylvania State University Press  III  August 15-19, 1988
                                                721

FEO, J.T.       An Analysis of Computational and Parallel Complexity of the Livermore Loops
[SUP 16]        Parallel Computing              7          163-85      (1988)

GRIFFUR, J.H.   A Debugger for Parallel Processes
[SUP 17]        Software Pract. & Exp.          18(12)     1179-90     Dec 88

BEMMERL, L.T.   An Integrated and Portable Tool Environment for Parallel Computers
[SUP 18]        Proceedings of the 1988 International Conference on Parallel Processing Algorithms and Applications Vol. III
                Pennsylvania State University Press  Aug 15-19, 1988
                                                50-53

GOLDBERG, A.P.  Transparent Process Cloning: A Tool for Load Management of Distributed Programs
[SUP 19]        Proc. 1988 Int. Conf. on Parallel Processing
                Pennsylvania State University Press  III  August 15-19, 1988
                                                728

KUMAR, M.       Measuring Parallelism in Computation-Intensive Scientific/Engineering Applications
[SUP 20]        IEEE Trans. on Computers        37(9)      1088-98     Sep 88

WHELAN, M.      Optimal Decomposition of Matrix Multiplication on Multiprocessor Architectures
[SUP 21]        Proc. 1988 Conference on Parallel Processing
                Pennsylvania State University Press  III
```

AUTHOR	TITLE / JOURNAL/PUBLISHER	VOL (#)	PAGE	DATE
MILLER, B.P. [SUP 22]	A Mechanism for Efficient Debugging of Parallel Programs Proc. SIGPLAN '88		135-44	
STONE, J.M. [SUP 23]	Debugging Concurrent Processes: A Case Study *Proc. SIGPLAN '88 Conference on Programming Language Design and Impiementation,* Atlanta June 22-24, 1988		145-53	
McCREARY, C. [SUP 24]	Automatic Determination of Grain Size for Specifying and Debugging Distributed Programs Communications of the ACM	32(9)	1073-78	Sep 89
CHENG, W-H.S. [SUP 25]	DEBL: A Knowledge-based Language for Specifying and Debugging Distributed Programs Communications of the ACM	32(9)	1079-84	Sep 89
.Cover []	Proceedings Computer Software and Applications [ARCH] I			Conference 5-7 October, 1988 Chicago COMPSAC '88
BELL,, G. [TECH 01]	The Future of High Performance Computers in Science and Engineering Communications of the ACM	32(9)	1091-1101	Sep 89

APPENDIX C - KEY WORDS AND AUTHOR ALPHABETICAL LISTING

APPENDIX C - KEY WORDS AND AUTHOR ALPHABETICAL LISTING

Part 1: Key Words and Topics

Author/Label	Primary Topic	Supporting Topics
Lager [ALG 1]	Gen. Purp. Signal Processing	Customizable processing, analysis, & display; Library of operations; Separate specification from implementation; Communication structure support
Purtilo [ALG 2]	Design Method	Separate specification from implementation, communication structure support
Bokhari [ALG 3]	Module Assignment	Multiple structure
Jamieson [ALG 4]	Mapping Parameters	Algorithm/architecture relationships
Chen [ALG 5]	Data Dependency Graph	Parallel algorithm design, forest and multi-stages
Frieze [ALG 6]	quadratic assignment	DAP SIMD
Engstrom [ALG 7]	Systolic Development Tools	Executable notation, intermediate language
McCrosky [ALG 8]	Array Manipulation	SIMD algorithms, data structures
Stone [ALG 9]	Parallel Database Query	Data level parallelism
O'Hallaron [ALG 10]	Kalman Filer	Warp computer
Alexander [ALG 11]	Multidimensional (N-D) signal processing	State space model, linear infinitive state machine
Lin [ALG 12]	Matrix Inversion	Dynamic communication structure, target for reconfigurable multi-processes
Arya [ALG 13]	Performance Estimator Tool	Algorithm efficiency on varying interconnection architectures prior to code

Author/Label	Primary Topic	Supporting Topics
Fox [ALG 14]	Matrix algorithm-multiplication	Hypercube
Feng [ALG 15]	Communicating Sequential Processes	Communication and synchronization
Armstrong [ALG 16]	High Performance One-D FFTS; Matched Vector	
Swarztrauber [ALG 17]	Multiprocessor FFTS	Hypercube
. [ALG 18]	Cover of Special Issue on Parallel Algrorithms	
Dinning [ARCH 1]	Methods survey	MMD Synchronization
McBryan [ARCH 2]	Review of state-of-art in parallel architecture	
Snyder [ARCH 3]	Taxonomy	
Casavant [ARCH 4]	Panel on reconfigurable architecture	
Martin [ARCH 5]	Performance evaluation	
Hach [ARCH 6]	General purpose parallel computing	
Hwang [ARCH 7]	Supercomputing architecture review	
Harp [ARCH 8]	ESPRIT Project 1085-	Reconfigurable Transputer
Bronnenberg [ARCH 9]	Symbolic/object-oriented machine DOOM	
Treleaven [ARCH 10]	Review of state-of-art in parallel architecture	
Bisiani [ENV 1]	Tool Coordination Tool	Planner to sequence tools/shell developer
Dart [ENV 2]	Environment Review	Language, structure, toolkit, & method environments, environment list

Author/Label	Primary Topic	Supporting Topics
Reeves [ENV 3]	SIMD Programming Environment	Parallel Pascal, MPP, Scientific programming
Carle [ENV 4]	Scientific multiprocessor env. (R^n)	Fortran development and maintenance, loop analysis
Smith [ENV 5]	Parallelizing Assistant Tool (PAT)	Fortran development, loop analysis
Guarna [ENV 6]	Edit, debug & tune env. (FAUST)	X-window & UNIX, project manager & database, Fortran & C
Appelbe [ENV 7]	Parallelizing Assistant Tool (PAT)	Fortran 8X, dependency graphs, Cray
[ENV 8]	Cover: Proceedings Super-computing '88	
Guarna [ENV 9]	(Faust)	Window manager, loop restructuring
Ertel [ENV 10]	PSC/2	Multiple users, file access, workstation
[ENV 11]	Parasoft	Hypercube OS, UNIX, I/O (Types for parallel PLOTIX) (Graphics) MOOSE, (Asyn. Op. Sys.)
Pienze [ENV 12]	Concurrent Programming	SUPRENUM, vector node parallel programming make library (mapping & communications
Dongarra [ENV 13]	Parallel FORTRAN	SCHEDULE, portability, hiding machine dependence
Pratt [ENV 14]	Scientific parallel programming	UNIX, Virtual Machine, multiple target machines
Dongarra [ENV 15]	SCHEDULE	
Pike interfaces [ENV 16]	concurrent windows	communications
Gehani	Concurrent C++	Two supersets of C with

Author/Label	Primary Topic	Supporting Topics
[EXT.LANG. 1]		data abstraction and parallel programming
Carlton [EXT.LANG. 2]	Distributed Prolog parallelism	Message passing for AND
Shibayama [EXT.LANG. 3]	Object-based Parallel Computing	Transformational rules, merging & splitting concurrent objects
Stevenson [EXT.LANG. 4]	Analysis of Sequential Prog to Determine Concurrency	Compliler optimization methods-discover data grouping, operations, communications, control flow
Chen [EXT LANG. 5]	Functional Language	Mathematical notation/ lambda calculus parallel program optimization
Grossman [EXT LANG. 6]	English Language for Math Program	Matrix arithmetic
Wholey [EXT LANG. 7]	Connection Machine Lisp	Fine-grained, data-oriented style xappings (arrays/hash tables)
Fisher [EXT LANG. 8]	SIMD Optimization	Abstraction of communication, compiler
Ruppelt [EXT LANG. 9]	Object-oriented Specification parallel prog.	PDE spec. language, SUPRENUM
Halstead [EXT.LANG. 10]	Futures, Symbolic Computing	Multilisp
Felten [EXT.LANG. 11]	Coherent Parallel C	Data parallel model, elimination of domain boundary checks, transparent process distribution, hypercube
Dally [EXT.LANG. 12]	Object-oriented Concurrent Programming	CST (parallel Smalltalk -80) distributed objects (state us dist. across many nodes)
Rosing [EXT LANG. 13]	Modified C for Distributed Memory Mach.	Better process and communication control

Author/Label	Primary Topic	Supporting Topics
Wolfe [EXT.LANG. 14]	Synchronization in Multiprocessor	Large job turnaround on shared memory machine
. [EXT.LANG. 15]	Cover: ESOP 86 European Symposium on Programming March 86 Saarbrucken	
Triolet [EXT.LANG. 16]	Parallelization with CALLS present	Restructuring compiler
Zorn [EXT.LANG. 17]	Extension to Common Lisp-Spur Lisp	Multiprocessing extensions
Allen [EXT.LANG. 18]	IBM Parallel Fortran Translator	3090/VF
Callahan [EXT.LANG. 19]	Dist. Mem. Compiler Issues	Virtual machine, efficient message passing
Mehrotra [EXT.LANG. 20]	Block Structured Scientific Language abstrations for portability	Array arithmetic,
Chen [EXT.OS. 1]	Dynamic Memory Management	Data dependency history, connection machine
Beck [EXT.OS. 2]	Parallel Operating System Support	Extensions to UNIX process model language extension, runtime library, ADA & CTT, sequent balance
Ellis [EXT.OS. 3]	Dynamic Storage Allocation	Global shared memory
Wolfstahl [EXT.OS. 4]	System Calls (mapping directive)	Signal changes in communication structure or occurance of mapping-related events
:Bain [EXT.OS. 5]	Concurrent Programming Toolkit	Intel IPSC
Tolle [EXT.OS. 6]	UNIX Utilies	NCUBE
Angus [EXT.OS. 7]	Parallel I/O Facility (C & FORTRAN)	Hypercube
Schwan	Arbitrary Communication	Global Function, Intel

Author/Label	Primary Topic	Supporting Topics
[EXT.OS. 8]	Graphs	
Gait [EXT.OS. 9]	Scheduling in a 2-tier Memory Hierarchy	Shared & distributed memory combined
Schroder [EXT.OS. 10]	Process Execution and Communication Environmment	SUPRENUM
Stevenson [EXT.OS. 11]	System Language Design	Distribution of virtual programs and large arrays Holistic merge of op. syst. & language
LeBlanc [EXT.OS. 12]	Set of Cooperating Processors	Balanced binary tree
Malony [EXT OS. 13]	Message passing facility	Portable C Library
Vomberger [EXT.OS 14]	Parallel Prolog on LAN	Async multiprocessor arch.
Rahgoza [EXT OS. 15]	Semantic Language of transactions	Distributed data, concurrency control mechanisms
Garg [EXT OS. 16]	Comm. Primatives	Analyzer of comm. structure Concurrent C
Fleckenstein [EXTOS 17]	parallel make	
Baalbergen [EXTOS 18]	parallel make	
Yau [HCI 1]	Visual Languages	Visualization, Software specifications
Krishnamoorthy [HCI 2]	Algorithm Animation	Graphics primatives
Tomboulian [HCI 3]	Schematic Programming	Graphical editor, visualization, validity checks
Gannon [HCI 4]	Program Restructuring	Performance predictor & statistics, pop-down !
Myers [HCI 5]	Tools for User-Interface	User-interface survey Tools

Author/Label	Primary Topic	Supporting Topics
Bailey [HCI 6]	Graphical Views	Icon, Vector, Simulator Views
Brown [HCI 7]	Algorithm Animator	Executive monitor with graphical I/O
Hsai [HCI 8]	pictorial programming	visualization and animation through tuples with both scenes and films
DiNitto [LANG 1]	Next Century Languages	Inertia of dusty deck software; very high level languages
Perrott [LANG 2]	Array and Vector Process Pascal	Optimizing compiler
Tripathi [LANG 3]	Object-oriented Langugage <u>SINA</u>	Concurrent and distributed programming data abstraction, concurrency, synchronization, inter-object communication; inheritance reuseability, delegation
Shapiro [LANG 4]	Process-oriented Language <u>Concurrent Prolog</u>	Data flow synchronization, guarded-command indeterminancy control
Baldwin [LANG 5]	Parallel Constraint Language	Implicit parallelism via compilerConsul
Tick [LANG 6]	Parallel Logic Programming Architecture	Prolog based
Clark [LANG 7]	PARLOG	Systems programming & object-oriented programming applications
Yamazaki [LANG 8]	Object-oriented Language	High to low level parallelism small-talk 80 virtual machine
Hansen [LANG 9]	Communicating Sequential Processor	Concurrent agents w/comm. via unbuffered channels
Karp [LANG 10]	Parallel FORTRAN dialects (12)	Alliant FX/8, BB&N Butterfly Cray X-MD, Elxsi 6400, Encore

Author/Label	Primary Topic	Supporting Topics
		Multimax, Flex 32, IBM 3090-VF, Intel iPSC, Sequent Balance
Gelernter Parallel [LANG 11]	Compiler vs. Language Issues	CSP, Occam, Ada, Lisp, Concurrent Prolog, Functional
Mundie [LANG 12]	Ada Parallel Processing	Ada tasking model
Goldman [LANG 13]	Qlisp-parallel lisp	Futures, spawn
Polychronopoulos [LANG 14]	Compiler Optimization	Barrier sync., data dependencies, run-time dependence
Guzzi [LANG 15]	FORTRAN-Vector & Parallel	Multiprocessor & vector
Girkar [LANG 16]	Compilers	Data dependencies
Welch [LANG 17]	Transputer/Occam	Abstraction, structuring & information hiding
Clapp [LANG 18]	Ada on a Hypercube	Run-time system
DeForest [LANG 19]	Hyperflow Declarative Language	Hypercube Lucid Non-sequential language
Lake [LANG 20]	Language & model of computation	
Jordan [LANG 21]	Language concepts	Force task assignment structural, non-sequential methods
Ahuja [LANG 22]	Linda	Tuple space
Whiteside [LANG 23]	Linda	LAN supercomputing
Lunberg [LANG 24]	parallel Ada real time system	MIMD multiprocessor task execution scheme
Watson [LANG 25]	concurrent language	

Author/Label	Primary Topic	Supporting Topics
[LANG 26]	Cover of IEE seminar on parallel processing	
Dongarra [LIB 1]	LINPACK	LAPACK matrix-matrix
Snelling [LIB 2]	Parallel Library	Data control, debug effects, hardware affinity, complexity (ease of use), issuer of libraries, (SPLIB by Snelling portable parallel library
Hammarling [LIB 3]	NAG Library	Linear algebra routines
Nicol [MAP 1]	Speed up	Problem size and optimal speedup (dup)
Kruskal [MAP 2]	Granularity	Definitions (many)
McDowell [MAP 3]	Static Analysis	Program Analysis
Berman [MAP 4]	Parallel Algorithms	Mapping into parallel architecture
Cherkassky [MAP 5]	Matrix Operation	Hypercube
Moreno [MAP 6]	Partitioning Algorithms into Systolic Arrays	Coalescing, cut & pile, decomposition into systolic arrays
Kumar [MAP 7]	Systolic Mapping	Mapping 2-d systolic arrays into 1-d arrays, improved matrix
Reddy [MAP 8]	Mapping I/O	Hypercube
Dubois [MAP 9]	Synchronization	Event ordering on multiprocessor
Wu [MAP 10]	Scheduling and Synchronization	Hypercube programming aid
Muhlenbein	Evolution Approach	Assignment problem

Author/Label	Primary Topic	Supporting Topics
[MAP 11]		SUPRENUM, competition, cooperation-better results than known heuristics
.Bailey [MAP 12]	Cover: 88 Proc. Int. Conf. on Parallel Processing Vol. II	
Kim [MAP 13]	Architectural Independence	Graph representation mapping linear clusters
McDowell [MAP 14]	Static Anaysis	Concurrency history graph, all possible parallel states
Greenbaum [MAP 15]	Synchronization Costs	Barrier vs. free forms
Stout [MAP 16]	Vision Algorithm	Mapping to parallel architectures variations among architectures
. [MAP 17]	Cover: Proceedings Supercomputing '87	
.Martin [MAP 18]	Session Summary	State-of-art in mapping
.Fox [MAP 19]	Cover: Third Conf. on Hypercube Concurrent Computers and Applications	
Fox [MAP 20]	Neural Network Modeling Algorithm	Hypercube
Fox [MAP 21]	Load Balancing	Neural networks (simulated annealing)
Salmon [MAP 22]	Scattered Decomposition	Costs and speedup possible
Pettey [MAP 23]	Simulated Annealing	Process Placement
Livingstone [MAP 24]	Reference Distribution Methods	Hypercube
Ercal	Task Allocation	Recursive minicut

Author/Label	Primary Topic	Supporting Topics
[MAP 25]		bipartitioning, hypercube
Chen [MAP 26]	Graph-oriented mapping	Approximation algorithms Greedy mapping, hypercube
Bell [MAP 27]	Data Parallelism	Data management independent large block partitioning
Kruatrachue [MAP 28]	Automatic Grain Size Determinations	Conflict between load balancing and communications miniturization grain packing with schedule optimizer.
Ramanujam [MAP 29]	Task Allocation	Simulated annealing
. [MAP 30]	Cover: Proceedings Supercomputing '88	
Kramer [MAP 31]	Task Allocation	Comparison of optimum with random solution
Musciano [MAP 32]	Task Allocation Dynamic Scheduling	Simultaneous PASCAL thread management profile
Weiss [MAP 33]	Loop Allocation Dynamic Scheduling	DOALL FORK-JOIN
Peir [MAP 34]	Recurrence Programming	Partitioning recurrences linear recurrence minimum distance method totally independent computation
. [MAP 35]	Cover: Proceedings 1988 Int. Conf. on Parallel Processing Vol. I.	
Missirlis [MAP 36]	PDE Programming	Successive over relation (SOR)
Kapenga [MAP 37]	Task Allocation	Adaptive task partitioning - MIMD high level macros portable, to MIMD 2-d integration method

Author/Label	Primary Topic	Supporting Topics
Bisiani [MAP 38]	Multilanguage	(Agora) heterogeneous machine shared memory abstractions
Colin [MAP 39]	Task Allocation	Graph model dependency graph, virtual distributed system
Rosenberg [MAP 40]	SIMD simulation	SIMD to vector mapping BLITZEN machine for NASA
deJong [MAP 41]	matrix bounds	symbolic bounds checking and correction
Katsef [MAP 42]	data partitioning	message passing machine assembler
Bradley [OS 1]	Picasso	Hypercube operating system
Krumme [OS 2]	SIMPLEX	NCUBE hypercube operating system
Salmon [OS 3]	MOOSE	Hypercube operating system
:Pierce [OS 4]	NX/2	Hypercube operating system
Gokhale analysis, [PROG 1]	Data Flow Language	Data dependency data flow language
Purtilo [PROG 2]	Environment Design System	Communication structure provided from specification
Browne portability, [PROG 3]	Computer Oriented Display Environment	Abstraction, visual programming, computation units, dependency relations, independence from architecture
Neves [PROG 4]	Parallel Programming	Problems of mapping vs computer developers claims

Author/Label	Primary Topic	Supporting Topics
DeMarco [PROG 5]	Parallel Programming	Data flow, reusability, object oriented,
Cavano [PROG 6]	Parallel Programming	No "systems" approach to parallel
Cavano [PROG 7]	Parallel Programming	No "systems" approach to parallel
Russell [PROG 8]	Parallel Programming	Scheme for reworking of existing FORTRAN; diagram of environment
Fox [PROG 9]	Parallel Programming	Grain control, loosely synchronous close; virtual machine; neutral simulated annealing decomposition
Chandy [PROG 10]	Architectural Independence	Specification notation, UNITY, functional vs. imperative programming vs. logic
Sobek interface, [PROG 11]	Architectural Independence	CODE graphical encapsulation strategy
Nicol [PROG 12]	Speedup	Relation between problem size and architecture
Chandy [PROG 13]	Speedup	Cost of storage & communications, time, space
Sabot [PROG 14]	Paralation Model	Parallel relation, algorithm description data structure (1) and operator (3)
Hudak [PROG 15]	Functional Programming	Separate specification and implementations "parafunctional programming", declarative program execution, algorithm dependent, "ParaAlfl"
Hudak [PROG 16]	See [PROG 15]	

Author/Label	Primary Topic	Supporting Topics
Fox [PROG 17]	Parallel Algorithms	Hypercube matrix, FFT, Monte Carlo, sorting, Scalar products, etc.
Kallstrom (Occam) [PROG 18]	Parallel Programming	iPSC's transputers & balance (c) - review of environments, division of execution, data sharing, synchronization of events
Hey [PROG 19]	Reconfigurable Parallel Program	Transputers - extracting parallelism, geometric parallelism (data distributed), algometric parallelism, control grain, load balancing/ communication overload
[PROG 20]	Cover: 88 Int. Conf. on Computer Language	
Bagrodia [PROG 21]	Parallel Programming	Enhanced C for the CM (SC) derived from UNITY, data parallel style (SIMD or MIMD), primative & data structures
McBryan [PROG 22]	PDE Parallel Programming	SOR, multigrid conjugate gradient
Bershad [PROG 23]	Object Oriented Parallel Programming	PRESTO, predefined object types: threads, synchronization objects, written in C++ Sequent Dynix, run time system
Brandes [PROG 24]	Parallel Programming	Parallelization tool/ knowledge based
Zima [PROG 25]	Parallel Programming	Semi-automatic parallelization for SUPRENUM multigrid
Percus [PROG 26]	Parallel Algorithm	Random number generator

Author/Label	Primary Topic	Supporting Topics
Storøy [PROG 27]	Parallel Programming	Monitor synchronization, highest level concurrency, algorithm design
Karp and [PROG 28]	Parallel Programming	Two styles: Fork-Join simple program multiple data
Polychronopoulos [PROG 29]	Vector Programming	Scheme for arbitrarily nested loops
Oldenhoeft [PROG 30]	Functional Programming	Applicative, functional definitions, data dependencies constrain evaluation, streams & interation in a single assignment language (SISAL)
. [PROG 31]	Cover: 1st Int. Conf. of Supercomputing - Athens, Greece	
Solchenbach PDE [PROG 32]	PDE	Multigrid methods for SUPRENUM
Dongarra [PROG 33]	Parallel Environment	Transportable numerical software SCHEDULE package (environment)
Jayasimha [PROG 34]	Parallel Programming	Performance estimation for synchronization
Francis [PROG 35]	Procedure Calls	Parallelism programming systems medium grain
Eisenstadter [PROG 36]	Locality of Reference	Software optimization
Weihl [PROG 37]	Abstract Data Types	Atomic actions
Bastani [PROG 38]	Parallel components	Abstract data type, functional, interface, control server, client
Martin [PROG 39]	Parallel procedure calls	UNIX

Author/Label	Primary Topic	Supporting Topics
Parkinson [PROG 40]	Speedup	Computation costs in multiprocessor summation operator, global operator
Terrano [PROG 41]	parallel compiler	distributed memory multiprocessor - reconfigurable
Preiss [PROG 42]	dataflow partitioning	data flow graphs
McGregor [Prog 43]	Comm. ACM Special Issue	
Ammar [SIM 1]	Speedup	Time cost, 5 categories of parallel structures
Bain [SIM 2]	HyperSim	Simulation of hypercubes on hypercubes
Yoder architecture [SIM 3]	Word Recognition	SIMD & Array word recognition, parallel algorithms
Ramamoorthy [SIM 4]	Petri Nets	Simulation method
Yaw [SIM 5]	Petri Nets	Cycle time simulation
Chung [SIM 6]	Petri Nets	Task simulation
Krauss [SIM 7]	Petri Nets	Process net
Stotts [SIM 8]	Execution Scematics	*Parallel Flow Graphics* (PFG), graphical programming, petri nets
Hura [SIM 9]	Petri net environment	PNSOFT
Bray [SIM 10] optimization	concurrency detection	architecture modeling architecture
Lopriore [SUP 1]	User interface for debugging & measurement	Block-oriented languages

Author/Label	Primary Topic	Supporting Topics
Martin [SUP 2]	Performance Evaluation	Practices of supercomputer performance evaluation
Gupta [SUP 3]	Symbolic Debugging of Parallel Code	Trace scheduling compiler /debugger
Mills [SUP 4]	Debugger for tree-oriented parallel computer	Network server, window manager, parallel debugger
Reeves [SUP 5]	Performance Meas. through fundamental algorithms	High-level language impacts
Hough [SUP 6	Pattern-oriented Debugger	Interprocessor control and data flow patterns time-stamped event stream
Burkhart [SUP 7]	Monitoring Facilitier	Multiprocessing losses, performance measure tool
Callahan [SUP 8]	Test Suite	Vectorizing compiler (100 Fortran Loops)
Eager [SUP 9]	Speedup and Efficiency Measures	
Pan [SUP 10]	Concurrent Debugger (DECON)	PSC /2, C & FORTRAN, lost messages
Bohm [SUP 11]	Performance Evaluation	Single assignment, data flow
:Flower [SUP 12]	CrOS Toolset (Comfort)	Communications, UNIX, PLOTIX, NDB, Hypercube, NCUBE
McGuire [SUP 13]	Concurrency Measurement	Cache Miss effects, FX/8
So [SUP 14]	Speedup Analyzer	
Allen [SUP 15]	Fortran Developer	Nondeterminism
Feo [SUP 16]	Parallel Complexity	Livermore Loops

Author/Label	Primary Topic	Supporting Topics
Griffin [SUP 17]	Fortran Analyzer & Debugger	GRAY-X-MP Multitasking library syntax
Bemmerl [SUP 18]	Visualization, Debugging, Performance Analysis	Portable tool design, integration into programming environment, multi-architecture
Goldberg protocol [SUP 19]	Concurrently Executing Processes	Load management, For transparent process cloning
Kumar [SUP 20]	Concurrency detection	Large Fortran codes
Kumar [SUP 20]	Concurrency Measurement	Large Numerical Programs
Whelan [SUP 21]	Matrix decomposition	Shared memory machines
Miller Debugger [SUP 22]	Debugging and incremental Tracing	Parallel Program (PPG)
McCreary [SUP 24]	Graph partitioning	Automatic grain size
Cheng [SUP 25]	Machine independent Parallel programming	DRAM model of machines
Stone [SUP 32]	Debugging via reply	Speculative reply concurrency map
Bell [TECH 1]	Future of high performance computing	

APPENDIX C - KEY WORDS AND AUTHOR ALPHABETICAL LISTING

Part 2: Author Alphabetical Listing

Author/Label	Primary Topic	Supporting Topics
Ahuja [LANG 22]	Linda	Tuple space
Alexander [ALG 11]	Multidimensional (N-D) Signal processing	State space model, linear infinitive state machine
Allen [EXT.LANG. 18]	IBM Parallel Fortran Translator	3090/VF
Allen [SUP 15]	Fortran Developer	Nondeterminism
Ammar [SIM 1]	Speedup	Time cost, 5 categories of parallel structures
Angus [EXT OS. 7]	Parallel I/O Facility (C & FORTRAN)	Hypercube
Appelbe [ENV 7]	Parallelizing Assistant Tool (PAT)	Fortran 8X, dependency graphs, Cray
Armstrong [ALG 16]	High Performance One-D FFTS; Matched Vector	
Arya [ALG 13]	Performance Estimator Tool	Algorithm efficiency on varying interconnection architectures prior to code
Baalbergen [EXTOS 18]	Parallel make	
Bagrodia [PROG 21]	Parallel Programming	Enhanced C for the CM (SC) derived from UNITY, data parallel style (SIMD or MIMD), primative & data structures
Bailey [HCI 6]	Graphical Views	Icon, Vector, Simulator Views
Bain [EXT.OS. 5]	Concurrent Programming Toolkit	Intel IPSC
Bain [SIM 2]	HyperSim	Simulation of hypercubes on hypercubes
Baldwin [LANG 5]	Parallel Constraint Language	Implicit parallelism via compilerConsul

Author/Label	Primary Topic	Supporting Topics
Bastani [PROG 38]	Parallel components	Abstract data type, functional, interface, control server, client
Beck [EXT.OS. 2]	Parallel Operating System Support	Extensions to UNIX process model language extension, runtime library, ADA & CTT, sequent balance
Bell [MAP 27]	Data Parallelism	Data management independent large block partitioning
Kruatrachue [MAP 28]	Automatic Grain Size Determinations	Conflict between load balancing and communications miniturization grain packing with schedule optimizer.
Bell [TECH 1]	Future of high performance computing	
Bemmerl [SUP 18]	Visualization, Debugging, Performance Analysis	Portable tool design, integration into programming environment, multi-architecture
Berman [MAP 4]	Parallel Algorithms	Mapping into parallel architecture
Bershad [PROG 23]	Object Oriented Parallel Programming	PRESTO, predefined object types: threads, synchronization objects written in C++ Sequent Dynix, run time system
Bisiani [ENV 1]	Tool Coordination Tool	Planner to sequence tools/shell developer
Bisiani [MAP 38]	Multilanguage	(Agora) heterogeneous machine shared memory abstractions
Bohm [SUP 11]	Performance Evaluation	Single assignment, data flow
Bokhari [ALG 3]	Module Assignment	Multiple structure
Bradley [OS 1]	Picasso	Hypercube operating system

Author/Label	Primary Topic	Supporting Topics
Brandes [PROG 24]	Parallel Programming	Parallelization tool/ knowledge based
Bray [SIM 10] optimization	Concurrency detection	Architecture modeling architecture
Bronnenberg [ARCH 9]	Symbolic/object-oriented machine <u>DOOM</u>	
Brown [HCI 7]	Algorithm Animator	Executive monitor with graphical I/O
Browne portability, [PROG 3]	Computer Oriented Display Environment	Abstraction, Visual programming, computation units, dependency relations, independence from architecture
Burkhart [SUP 7]	Monitoring Facilitier	Multiprocessing losses performance measure tool
Callahan [EXT.LANG. 19] passing	Dist. Mem. Compiler Issues	Virtual machine, efficient message
Callahan [SUP 8]	Test Suite	Vectorizing compiler (100 Fortran Loops)
Carle [ENV 4]	Scientific multiprocessor env. (R^n)	Fortran development and maintenance, loop analysis
Carlton [EXT.LANG. 2]	Distributed Prolog	Message passing for AND parallelism
Casavant [ARCH 4]	Panel on reconfigurable architecture	
Cavano [PROG 6]	Parallel Programming	No "systems" approach to parallel
Cavano [PROG 7]	Parallel Programming	No "systems" approach to parallel
Chandy	Architectural Independence	Specification notation,

Author/Label	Primary Topic	Supporting Topics
[PROG 10]		UNITY, functional vs. imperative programming vs. logic
Chandy [PROG 13]	Speedup	Cost of storage & communications, time, space
Chen [ALG 5]	Data Dependency Graph	Parallel algorithm design, forest, and multistages
Chen [EXT.LANG. 5]	Functional Language	Mathematical notation/ lambda calculus parallel program optimization
Chen [EXT.OS. 1]	Dynamic Memory Management	Data dependency history, Connection machine
Chen [MAP 26]	Graph-oriented mapping	Approximation algorithms Greedy mapping, hypecube
Cheng [SUP 25]	*Machine independent* parallel programming	DRAM model of machines
Cherkassky [MAP 5]	Matrix Operation	Hypercube
Chung [SIM 6]	Petri Nets	Task simulation
Clapp [LANG 18]	Ada on a Hypercube	Run-time system
Clark [LANG 7]	PARLOG	Systems programming & object-oriented programming applications
Colin [MAP 39]	Task Allocation	Graph model dependency graph, virtual distributed system
Dally [EXT.LANG. 12]	Object-oriented Concurrent Programming	CST (parallel Small talk -80) distributed objects (state us dist. across many nodes)
Dart	Environment Review	Language, structure,

Author/Label	Primary Topic	Supporting Topics
[ENV 2]		toolkit, & method environments, environment list
DeForest [LANG 19]	Hyperflow Declarative Language	Hypercube Lucid Nonsequential language
deJong [MAP 41]	Matrix bounds	Symbolic bounds checking and correction
DeMarco [PROG 5]	Parallel Programming	Data flow, reusability, object oriented,
DiNitto [LANG 1]	Next Century Languages	Inertia of dusty deck software; very high level languages
Dinning [ARCH 1]	Methods survey	MMD Synchronization
Dongarra [ENV 13]	Parallel FORTRAN	SCHEDULE, portability, hiding machine dependence
Dongarra [ENV 15]	SCHEDULE	
Dongarra [LIB 1]	LINPACK	LAPACK matrix-matrix
Dongarra [PROG 33]	Parallel Environment	Transportable numerical software SCHEDULE package (environment)
Dubois [MAP 9]	Synchronization	Event ordering on multi-Processor
Eager [SUP 9]	Speedup and Efficiency Measures	
Eisenstadter [PROG 36]	Locality of Reference	Software optimization
Ellis [EXT.OS. 3]	Dynamic Storage Allocation	Global shared memory
Engstrom [ALG 7]	Systolic Development Tools	Executable notation, intermediate language
Ercal	Task Allocation	Recursive minicut

Author/Label	Primary Topic	Supporting Topics
[MAP 25]		bipartitioning, hypercube
Ertel [ENV 10]	PSC/2	Multiple users, file access, workstation
Felten [EXT.LANG. 11]	Coherent Parallel C	Data parallel model, elimination of domain boundary checks, transparent process distribution, hypercube
Feng [ALG 15]	Communicating Sequential Processes	Communication and synchronization
Feo [SUP 16]	Parallel Complexity	Livermore Loops
Fisher [EXT.LANG. 8]	SIMD Optimization	Abstraction of communication, compiler
Fleckenstein [EXTOS 17]	Parallel make	
Flower [SUP 12]	CrOS Toolset (Comfort)	Communications, UNIX, PLOTIX, NDB, Hypercube, NCUBE
Fox [ALG 14]	Matrix algorithm-multiplication	Hypercube
Fox [MAP 20]	Neural Network Modeling Algorithm	Hypercube
Fox [MAP 21]	Load Balancing	Neural networks (simulated annealing)
Fox [PROG 9]	Parallel Programming	Grain control, loosely synchronous close; virtual machine; neutral simulated annealing decomposition
Fox [PROG 17]	Parallel Algorithms	Hypercube matrix, FFT, Monte Carlo, sorting, Scalar products, etc.
Francis [PROG 35]	Procedure Calls	Parallelism programming systems medium grain
Frieze [ALG 6]	Quadratic assignment	DAP SIMD

Author/Label	Primary Topic	Supporting Topics
Gait [EXT.OS. 9]	Scheduling in a 2-tier Memory Hierarchy	Shared & distributed memory combined
Gannon [HCI 4]	Program Restructuring	Performance predictor & statistics, pop-down !
Garg [EXT.OS. 16]	Comm. Primatives	Analyzer of comm. structure Concurrent C
Gehani [EXT.LANG. 1]	Concurrent C++	Two supersets of C with data abstraction and parallel programming
Gelernter Parallel [LANG 11]	Compiler vs. Language Issues	CSP, Occam, Ada, Lisp, Concurrent Prolog, Functional
Girkar [LANG 16]	Compilers	Data dependencies
Gokhale analysis, [PROG 1]	Data Flow Language	Data dependency data flow language
Goldberg [SUP 19]	Concurrently Executing Processor	Load management, protocol for transparent process cloning
Goldman [LANG 13]	Qlisp-parallel lisp	Futures, spawn
Greenbaum [MAP 15]	Synchronization Costs	Barrier vs. free forms
Griffin [SUP 17]	Fortran Analyzer & Debugger	CRAY-X-MP multitasking library syntax
Grossman [EXT.LANG. 6]	English Language for Math Program	Matrix arithmetic
Guarna [ENV 6]	Edit, debug & tune env. (FAUST)	X-window & UNIX, project manager & database, Fortran & C
Guarna [ENV 9]	(Faust)	Window manager, loop restructuring
Gupta	Symbolic Debugging of	Trace scheduling compiler

Author/Label	Primary Topic	Supporting Topics
[SUP 3]	Parallel Code	/debugger
Guzzi [LANG 15]	FORTRAN-Vector & Parallel	Multiprocessor & vector
Hach [ARCH 6]	General purpose parallel computing	
Halstead [EXT.LANG. 10]	Futures, Symbolic Computing	Multilisp
Hammarling [LIB 3]	NAG Library	Linear algebra routines
Hansen [LANG 9]	Communicating Sequential Processor	Concurrent agents w/comm. via unbuffered channels
Harp [ARCH 8]	ESPRIT Project 1085-	Reconfigurable Transputer
Hey [PROG 19]	Reconfigurable Parallel Program	Transputers - extracting parallelism, geometric parallelism (data distributed), algometric parallelism, control grain, load balancing/ communication overload
Hough [SUP 6]	Pattern-oriented Debugger	Interprocessor control and data flow patterns time-stamped event stream
Hsai [HCI 8]	Pictorial programming	Visualization and animation through tuples with both scenes and films
Hudak [PROG 15]	Functional Programming	Separate specification and implementations "parafunctional programming", declarative program execution, algorithm dependent, "ParaAlfl"
Hudak [PROG 16]	See [PROG 15]	
Hura [SIM 9]	Petri net environment	PNSOFT

Author/Label	Primary Topic	Supporting Topics
Hwang [ARCH 7]	Supercomputing architecture review	
Jamieson [ALG 4]	Mapping Parameters	Algorithm/architecture relationships
Jayasimha [PROG 34]	Parallel Programming	Performance estimation for synchronization
Jordan [LANG 21]	Language concepts	<u>Force</u> task assignment structural, non-sequential methods
Kallstrom (Occam) [PROG 18]	Parallel Programming	IPSC's transputer & balance (c) - review of environments, division of execution, data sharing, synchronization of events
Kapenga [MAP 37]	Task Allocation	Adaptive task partitioning - MIMD high level macros portable, to MIMD 2-d integration method
Karp [LANG 10]	Parallel FORTRAN dialects (12)	Alliant FX/8, BB&N Butterfly Cray X-MD, Elxsi 6400, Encore Multimax, Flex 32, IBM 3090-VF, Intel iPSC, Sequent Balance
Karp and [PROG 28]	Parallel Programming	Two styles: fork-join Simple program multiple data
Katsef [MAP 42]	Data partitioning	Message passing machine assembler
Kim [MAP 13]	Architectural Independence	Graph representation mapping linear clusters
Kramer [MAP 31]	Task Allocation	Comparison of optimum with random solution
Krauss [SIM 7]	Petri Nets	Process net

Author/Label	Primary Topic	Supporting Topics
Krishnamoorthy [HCI 2]	Algorithm Animation	Graphics primatives
Krumme [OS 2]	SIMPLEX	NCUBE hypercube operating system
Kruskal [MAP 2]	Granularity	Definitions (many)
Kumar [MAP 7]	Systolic Mapping	Mapping 2-d systolic arrays into 1-d arrays, improved matrix
Kumar [SUP 20]	Concurrency Measurement	Large Numerical Programs
Kumar [SUP 20]	Concurrency detection	Large Fortran codes
Lager [ALG 1]	Gen. Purp. Signal Processing	Customizable processing, analysis, & display; Library of operations; Separate specification from implementation; Communication structure support
Lake [LANG 20]	Language & model of computation	
LeBlanc [EXT.OS. 12]	Set of Cooperating Processors	Balanced binary tree
Lin [ALG 12]	Matrix Inversion	Dynamic communication structure, target for re-configurable multi-processes
Livingstone [MAP 24]	Reference Distribution Methods	Hypercube
Lopriore [SUP 1]	User interface for debugging & measurement	Block-oriented languages
Lunberg [LANG 24]	Parallel Ada realtime system	MIMD multiprocessor task execution scheme
Malony [EXT.OS. 13]	Message passing facility	Portable C Library

Author/Label	Primary Topic	Supporting Topics
Martin [ARCH 5]	Performance evaluation	
Martin [PROG 39]	Parallel procedure calls	UNIX
Martin [SUP 2]	Performance Evaluation	Practices of supercomputer performance evaluation
McBryan [ARCH 2]	Review of state-of-art in parallel architecture	
McBryan [PROG 22]	PDE Parallel Programming	SOR, multigrid conjugate gradient
McCreary [SUP 24]	Graph partitioning	automatic grain size
McCrosky [ALG 8]	Array Manipulation	SIMD algorithms, data structures
McDowell [MAP 3]	Static Analysis	Program Analysis
McDowell [MAP 14]	Static Anaysis	Concurrency history graph, all possible parallel states
McGuire [SUP 13]	Concurrency Measurement	Cache Miss effects, FX/8
Mehrotra [EXT.LANG. 20]	Block Structured Scientific Language abstrations for portability	Array arithmetic,
Miller Debugger [SUP 22]	Debugging and incremental Tracing	Parallel Program (PPG)
Mills [SUP 4]	Debugger for tree-oriented parallel computer	Network server, window manager, parallel debugger
Missirlis [MAP 36]	PDE Programming	Successive over relation (SOR)
Moreno [MAP 6]	Partitioning Algorithms into Systolic Arrays	Coalescing, cut & pile, decomposition into systolic arrays
Muhlenbein	Evolution Approach	Assignment problem

Author/Label	Primary Topic	Supporting Topics
[MAP 11]		SUPRENUM, competition, cooperation-better results than known heuristics
Mundie [LANG 12]	Ada Parallel Processing	Ada tasking model
Musciano [MAP 32]	Task Allocation Dynamic Scheduling	Simultaneous PASCAL thnead management profile
Myers [HCI 5]	Tools for User-Interface	User-interface survey Tools
Neves [PROG 4]	Parallel Programming	Problems of mapping vs. computer developers claims
Nicol [MAP 1]	Speed up	Problem size and optimal speedup (dup)
Nicol [PROG 12]	Speedup	Relation between problem size and architecture
O'Hallaron [ALG 10]	Kalman Filer	Warp computer
Oldenhoeft [PROG 30]	Functional Programming	Applicative, functional definitions, data dependencies constrain evaluation, streams & interation in a single assignment language (SISAL)
Pan [SUP 10]	Concurrent Debugger (DECON)	PSC /2, C & FORTRAN, lost messages
Parkinson [PROG 40]	Speedup	Computation costs in multiprocessor summation operator, global operator
Peir	Recurrence Programming	Partitioning recurrences
[MAP 34]		Linear recurrence minimum distance method totally independent computation

Author/Label	Primary Topic	Supporting Topics
Percus [PROG 26]	Parallel Algorithm	Random number generator
Perrott [LANG 2]	Array and Vector Process Pascal	Optimizing compiler
Pettey [MAP 23]	Simulated Annealing	Process Placement
Pienze [ENV 12]	Concurrent Programming	SUPRENUM, vector node parallel programming make library (mapping & communications
Pierce [OS 4]	NX/2	Hypercube operating system
Pike interfaces [ENV 16]	Concurrent windows	Communications
Polychronopoulos [LANG 14]	Compiler Optimization	Barrier sync., data dependencies, run-time dependence
Polychronopoulos [PROG 29	Vector Programming	Scheme for arbitrarily nested loops
Pratt [ENV 14]	Scientific parallel programming	UNIX, Virtual Machine, multiple target machines
Preiss [PROG 42]	Dataflow partitioning	Dataflow graphs
Purtilo [ALG 2]	Design Method	Separate specification from implementation, communication structure support
Purtilo [PROG 2]	Environment Design System	Communication structure Provided from specification
Rahgoza [EXT.OS. 15]	Semantic Language of transactions	Distributed data, concurrency control mechanisms
Ramamoorthy [SIM 4]	Petri Nets	Simulation method

Author/Label	Primary Topic	Supporting Topics
Ramanujam [MAP 29]	Task Allocation	Simulated annealing
Reddy [MAP 8]	Mapping I/O	Hypercube
Reeves [ENV 3]	SIMD Programming Environment	Parallel Pascal, MPP, Scientific programming
Reeves [SUP 5]	Performance Meas. through fundamental algorithms	High-level language impacts
Rosenberg [MAP 40]	SIMD simulation	SIMD to vector mapping BLITZEN machine for NASA
Rosing [EXT.LANG. 13]	Modified C for Distributed Memory Mach.	Better process and communication control
Ruppelt [EXT.LANG. 9]	Object-oriented Specification parallel prog.	PDE spec. language, SUPRENUM
Russell [PROG 8]	Parallel Programming	Scheme for re-working of Existing FORTRAN; diagram of environment
Sabot [PROG 14]	Paralation Model	Parallel relation, algorithm description data structure (1) and operator (3)
Salmon [MAP 22]	Scattered Decomposition	Costs and speedup possible
Salmon [OS 3]	MOOSE	Hypercube operating system
Schroder [EXT.OS. 10]	Process Execution and Communication Environmment	SUPRENUM

Author/Label	Primary Topic	Supporting Topics
Schwan [EXT.OS. 8]	Arbitrary Communication Graphs	Global Function, Intel
Shapiro [LANG 4]	Process-oriented Language Concurrent Prolog	Data flow synchronization, guarded-command indeterminancy control
Shibayama [EXT.LANG. 3]	Object-based Parallel Computing	Transformational rules, merging & splitting concurrent objects
Smith [ENV 5]	Parallelizing Assistant Tool (PAT)	Fortran development, loop analysis
Snelling [LIB 2]	Parallel Library	Data control, debug effects, hardware affinity, complexity (ease of use), issuer of libraries, (SPLIB by Snelling portable parallel library
Snyder [ARCH 3]	Taxonomy	
So [SUP 14]	Speedup Analyzer	
Sobek interface, [PROG 11]	Architectural Independence	CODE graphical Encapsulation strategy
Solchenbach PDE [PROG 32]	PDE	Multigrid methods for SUPRENUM
Stevenson [EXT.LANG. 4]	Analysis of Sequential Prog. to Determine Concurrency	Compliler optimization methods-discover data grouping, operations, communications, control flow
Stevenson [EXT.OS. 11]	System Language Design	Distribution of virtual programs and large arrays Holistic merge of op. syst. & language
Stone [ALG 9]	Parallel Database Query	Data level parallelism
Stone	Debugging via reply	Speculative reply

Author/Label	Primary Topic	Supporting Topics
[SUP 32]		concurrency map
Storøy [PROG 27]	Parallel Programming	Monitor synchronization, highest level concurrency, algorithm design
Stotts [SIM 8]	Execution Scematics	Parallel Flow Graphics (PFG), graphical programming, petri nets
Stout [MAP 16]	Vision Algorithm	Mapping to parallel architectures variations among architectures
Swarztrauber [ALG 17]	Multiprocessor FFTS	Hypercube
Terrano [PROG 41]	Parallel compiler	Distributed memory multiprocessor - reconfigurable
Tick [LANG 6]	Parallel Logic Programming Architecture	Prolog based
Tolle [EXT.OS. 6]	UNIX Utilies	NCUBE
Tomboulian [HCI 3]	Schematic Programming	Graphical editor, visualization, validity checks
Treleaven [ARCH 10]	Review of state-of-art in parallel architecture	
Triolet [EXT.LANG. 16]	Parallelization with CALLS present	Restructuring compiler
Tripathi [LANG 3]	Object-oriented Langugage <u>SINA</u>	Concurrent and distributed programming data abstraction, concurrency, synchronization, interobject communication; inheritance reuseability, delegation
Vomberger [EXT.OS. 14]	Parallel Prolog on LAN	Async multiprocessor arch.
Watson [LANG 25]	Concurrent language	

Author/Label	Primary Topic	Supporting Topics
Weihl [PROG 37]	Abstract Data Types	Atomic actions
Weiss [MAP 33]	Loop Allocation Dynamic Scheduling	DOALL FORK-JOIN
Welch [LANG 17]	Transputer/Occam	Abstraction, structuring & information hiding
Whelan [SUP 21]	Matrix decomposition	Shared memory machine
Whiteside [LANG 23]	Linda	LAN supercomputing
Wholey [EXT.LANG. 7]	Connection Machine Lisp	Fine-grained, data-oriented style xappings (arrays/hash tables)
Wolfe	Synchronization in	Large job turnaround on
[EXT.LANG. 14]	Multiprocessor	Shared memory machir.s
Wolfstahl [EXT.OS. 4]	System Calls (mapping directive)	Signal changes in communication structure or occurance of mapping-related events
Wu [MAP 10]	Scheduling and Synchronization	Hypercube programming aid
Yamazaki [LANG 8]	Object-oriented Language	High to low level parallelism small-talk 80 virtual machine
Yau [HCI 1]	Visual Languages	Visualization, Software specifications
Yaw [SIM 5]	Petri Nets	Cycle time simulation
Yoder architecture [SIM 3]	Word Recognition	SIMD & Array Word recognition, parallel algorithms
Zima [PROG 25]	Parallel Programming	Semi-automatic parallelization for SUPRENUM multigrid
Zorn	Extension to Common Lisp-	Multiprocessing

Author/Label	Primary Topic	Supporting Topics
[EXT.LANG. 17]	Spur Lisp	Extensions

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 1990	3. REPORT TYPE AND DATES COVERED
4. TITLE AND SUBTITLE LITERATURE SURVEY ON TOOLS		5. FUNDING NUMBERS C: N66001-87-D-0039
6. AUTHOR(S) C. G. Murphy		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Science Applications International Corporation 10240 Sorrento Valley Road , Suite 202 San Diego, CA 92121		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Ocean Systems Center San Diego, CA 92152-5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER NOSC TD 1853
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This literature survey of parallel programming tools, including more than 200 references, is designed to allow continued browsing and probing of different specific areas of interest. Therefore, the first look at parallel programming tools is a broad one, where each reference article is briefly described. Key words (or phrases) were assigned and provided in sorts by author and by assignment to a hierarchy of high performance computing technology. This hierarchy ranges from algorithms through tools environments to architectures and technology. A table of commercial programming tools is also provided.

14. SUBJECT TERMS multiprocessor parallel programming algorithm signal processing			15. NUMBER OF PAGES 143 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAME AS REPORT

SUPPLEMENTARY

INFORMATION

AD-A226 933

NAVAL OCEAN SYSTEMS CENTER
SAN DIEGO, CALIFORNIA 92152-5000

ERRATA

24 September 1990

NOSC Technical Document 1853
Literature Survey of Parallel Processing Tools
By C. G. Murphy, Science Applications International Corporation
Dated July 1990

Literature Change

1. Replace the covers of NOSC TD 1853 mailed to you on 14 September 1990.

Naval Ocean Systems Center
San Diego, CA 92152-5000

Technical Document 1853
July 1990

Literature Survey of Parallel Processing Tools

C. G. Murphy

Science Applications International Corporation

ADMINISTRATIVE INFORMATION

This report was prepared by Science Applications International Corporation (SAIC), under contract N66001-87-D-0039 for Code 733 of the Naval Ocean Systems Center.

Released by
D. K. Barbour, Head
Signal Processing
Technology Branch

Under authority of
J. A. Roese, Head
Signal and Information
Processing Division

JJ