A Survey of Parallel Programming Languages and Tools

Doreen Y. Cheng

Report RND-93-005 March 1993

NASA Ames Research Center
M/S 256-6
Moffett Field, CA 94035

Abstract

This survey examines thirty-five parallel programming languages and fifty-nine parallel programming tools. It focuses on tool capabilities needed for writing parallel scientific programs, not on features that explore general computer science issues. The tools are classified based on their functions and ranked with current and future needs of NAS in mind: in particular, existing and anticipated NAS supercomputers and workstations, operating systems, programming languages, and applications. The report is designed to give readers a quick grasp of the tool features, and provides tables to compare their main functions.

Introduction

Providing sufficient parallel programming tools is one key step to enable NAS users to use parallel computers. The report "A Survey of Parallel Programming Tools" (RND-91-005) has been requested by more than 100 organizations. Since then, NAS has added massively parallel supercomputers into its production computing facilities. Programming for a wide variety of parallel architectures (SIMD, shared-memory MIMD, distributed-memory MIMD, heterogeneous MIMD) demands a survey of a broader range of programming tools than the tools included in report RND-91-005. In response to the new demand, this report surveys parallel programming languages as well as tools. In the text below, they are both referred to as tools. The scope of the survey has been enlarged to include tools for all forms of parallel architectures and programming paradigms.

More than 80 tools were submitted to the survey; only a few were eliminated due to their proprietary or obsolescent nature. About a dozen entries were written based on available documents. This survey shares the same goal with the previous one: to help scientific users use the NAS parallel supercomputers. Focus is placed on those tool capabilities needed for parallel scientific programming rather than for general computer science.

This report describes 94 entries. They are grouped into five main categories based on their functions: languages, libraries, debugging and performance tools, parallelization tools, and others. The others include partitioning, scheduling and load balancing tools, network utilities, and tools for building tools. This report has five parts, one for each category of tools. A category may contain subcategories; each subcategory is organized as one section. The tools in each category/subcategory are ordered according to their usefulness to NAS users. The usefulness is rated based on the languages that a tool supports, the platforms on which it is currently available, the maturity of the tool, and the support provided. The information reported is based on the submissions by the tool developers and available documents; it reflects the status of the reported tools up until February, 1993.

The report is designed to give readers a quick grasp of the functions provided by a tool and of its usefulness in comparison with the tools in the same category. For this purpose, the description of each tool gives no details but only a brief outline of its main functions from a user point of view. References and contacts to the tool providers are given for the readers who decide to learn more about the selected tools. Similar efforts with slightly different purposes and emphasis can be found in [1] [2]

To make it easier for readers to find a specific tool in the report, a look-up table in pages 6-8 lists all the tools in alphabetic order. The "type" entry of the table indicates the tool category. Appendix A-E presents the tables that compare the tools in the same category, one for each category. The tools in these tables are also in alphabetic order. A page number is listed for each tool indicating where it is described. The tables also indicate the usefulness of a tool to NAS. The next section explains how the usefulness rating was determined.

Evaluation Criteria

This report includes evaluation results from two sources: user submissions and evaluation based on NAS needs. The only editing applied to the submissions describing user experiences is formatting. Entries with no "Evaluation" item imply that no descriptions have been submitted from their users.

The evaluation of the usefulness to NAS is based on NAS user experience (if it exists) and the literature available. A letter rating scheme is used in all the tables and the meaning of the letter rating is spelled out in each tool entry. The rating is based on NAS user feedback, the languages that a tool supports, the platforms on which it is currently available, the maturity of the tool, and the support provided. The next two paragraphs briefly introduces the computing environment and the user status at NAS.

NAS currently provides an Intel iPSC/860 and Paragon, Thinking Machine CM5, CRAY C-90 and CRAY Y-MP for computation-intensive CFD applications. More massively parallel machines are under consideration for procurement. In addition, NAS is experimenting with a more cost effective approach to parallel computing by using idle cycles of a network of workstations. The majority of the workstations at NAS are SGIs with a small number of SUNs. All platforms use UNIX or UNIX-based operating systems.

NAS users are quite familiar with vector-oriented supercomputers (CRAY-style), and are making a transition to using massively parallel machines (Intel, TMC, and a network of workstations). A small percentage of the users have been developing new parallel algorithms and applications from scratch; the majority of them have been modifying programs developed by others and/or converting programs to run on the parallel platforms. Most NAS applications are in Fortran and most users will be using Fortran for new application development in foreseeable future. A small number of users have started to use C and C++.

The remining paragraphs of this section explain the rating scheme. The rating is biased by NAS needs, and therefore may not apply to other organizations. The requirements for tools in different categories may differ. For example, supporting all NAS platforms is required by languages and libraries, but not by debugging and performance tools.

A rating "y" in the tables or "Yes" in the descriptions means that the tool is useful to NAS. Criteria 1 or 2 listed below plus 3 and 4 must be met for a tool to receive this rating: (1) The tool that has been used by NAS users and it is positively recommended by the users. (2) The tool has been evaluated at NAS and other organizations and it is positively recommended. (3) The tool supports the platforms, operating systems, and languages required at NAS. (4) The tool supplier provides support and maintenance.

A rating "m" in the tables or "Maybe" in the entries means that the tool maybe useful in its current status. Four possibilities lead to a tool to receive such a rating: (1) The functions provided by the tool match NAS user needs, but none or limited user experience or evaluation has been reported and the results are not uniformly positive. (2) The tool is supplied by a hardware vendor, and it is needed by NAS users (if the hardware is available at NAS). (3) The tool is an emerging standard supported by many organizations. (4) The tool may be able to help parallel tool development.

A rating "n" in the tables and "No" in the entries means that the tool is not useful to NAS. A rating "n,m" means that the tool is not useful to NAS users in its current status, but may become useful in the future. Five possibilities lead to a tool to receive such ratings: (1) The tool does not provide important functions needed by NAS. (2) The tool does not support the platforms, operating systems, and/or languages required by NAS. (3) The tool is based on another tool that is no longer supported. (4) Other tools in the same category provide more complete functions and/or support more platforms. (5) The tool is still in prototyping stage; more development work is needed for production use.

Index

When the type entry has more than one items connected by "+", the first one is the primary type. For example, "l+d+p" mean the tool is a language with debuggers and performance tools built to support its use.

d:	debugger
l:	language
lib:	library
m:	maybe (user experience needed, support NAS systems)
n:	no
n,m:	no at present time, maybe in the future
ns:	network support
p:	performance tool
pa:	parallelization tool
sc:	scheduler, load balancer
std:	proposed standard
y:	yes

When the type entry has more than one items connected by "+", the first one is the primary type. For example, "l+d+p" mean the tool is a language with debuggers and performance tools built to support its use.

d:	debugger
l:	language
lib:	library
m:	maybe (user experience needed, support NAS systems)
mp:	meta-tool for building performance tools
n:	no
n,m:	no at present time, maybe in the future
ns:	network support
p:	performance tool
pa:	parallelization tool
sc:	scheduler, load balancer
std:	proposed standard
y:	yes

When the type entry has more than one items connected by "+", the first one is the primary type. For example, "l+d+p" mean the tool is a language with debuggers and performance tools built to support its use.

d: debugger
l: language
lib: library
m: maybe (user experience needed, support NAS systems)
mc: meta-tool for building compilers
mp: meta-tool for building performance tools
n: no
n,m: no at present time, maybe in the future
p: performance tool
pa: parallelization tool
sc: scheduler, load balancer
std: proposed standard
y: yes

1. Parallel Languages

This part presents 35 parallel language some of which have tools developed around them. Section 1.1 presents parallel languages based on extending sequential Fortran. Section 1.2 presents languages extending C. Section 1.3 lists parallel extensions to object-oriented languages. Section 1.4 describes functional languages, logical languages, and coordination languages (the languages which tie functions written in other languages together). The languages within each section are ordered according to its availability on the platforms in which NAS is interested.

1.1 Fortran-Based Languages

1.1.1 FORTRAN 90

Functions:

- Extensions to Fortran 77 for parallel programming

- Array operations:

  - Extending arithmetic, logical, and character operations and intrinsic functions to operate on array-valued operands
  - Whole, partial, and masked array assignment (WHERE)
  - Array-valued constants and expressions
  - User-supplied array-valued functions
  - Intrinsic procedures to manipulate and construct arrays, to perform gather/scatter operations, and to support extended computational capabilities involving arrays

- Control statements and constructs:

  - SELECT and CASE, DO WHILE/ENDDO, EXIT, and CYCLE
  - Improved facilities for numerical computation
  - Portable control over numeric precision specification
  - Inquiry as to the characteristics of numeric representation
  - Improved control of the performance of numerical programs

- Intrinsic functions

  - Dot product
  - Matrix multiply
  - Matrix reduction operations

- User-defined data types

- Facilities for modular data and procedure definitions

- Pointers

- Dynamic memory allocation/deallocation

- INCLUDE facility to reduce the duplication of common declarations

- Recursive subroutine calls

- Parameterized intrinsic data types to include character sets other than English

Useful to NAS:	Maybe (Emerging standard, User experience needed)

Platforms:	The proposed standard is supported by: Convex, Cray Research, DEC, Intel, MasPar, SGI, TMC, and Tera
Operating System:	Supported by each platform
Languages Supported:	Fortran 90
Languages Used in Implementation:	Vendor dependent
Graphic User Interface:	None
Cost:	Vendor dependent
Supplier:	The specification is supplied by ISO/IEC
Contact:	ISO/IEC

See reference [3]

1.1.2 HPF (High Performance Fortran)

Functions:

- Extensions to Fortran90 for high performance parallel programming
- Directives:
  - Data alignment and distribution to increase locality of reference
  - Assertion that the statements in a particular section of code do not exhibit any sequential dependencies
  - Declaration of rectilinear processor arrangements
- FORALL statement and construct
- Pure procedures (procedures, functions, and subroutine that do not produce side effects) for elimination of undesirable consequences such as non-determinism in parallel execution
- Extended intrinsic functions and standard library:
  - Basic operations that are valuable in parallel algorithm design:
    - Reduction functions
    - Combining-Scatter functions (Performing combining operations on gathered data then scatter the results)
    - Prefix/Suffix functions
    - Sorting functions
    - Bit-manipulation functions
  - System inquiry functions:
    - Actual mapping of an array at run time
    - Number of processors and the topology of processors
- Extrinsic procedures:
  - Interface to procedures written in other paradigms (e.g. message passing)
  - Interface to other languages (e.g. C)
- Parallel I/O (same as in Fortran 90)
- Sequence and storage association

Useful to NAS:	Maybe (Emerging standard, User experience needed)
Platforms:	The proposed standard is supported by: Alliant, Convex, Cray Research, DEC, Fujitsu, HP, IBM, Intel, MasPar, Meiko, nCUBE, and TMC

Operating System:	Supported by each platform
Languages Supported:	HPF
Languages Used in Implementation:	Vendor dependent
Graphic User Interface:	None
Cost:	Vendor dependent
Supplier:	The specification is supplied by High Performance Fortran Forum
Contact:	A draft of "High Performance Fortran Language Specification" available at: titan.cs.rice.edu in public/HPFF/draft/hpf-v10.ps think.com theory.tc.cornell.edu minerva.npac.syr.edu ftp.gmd.de

See reference [4]

1.1.3 X3H5

Functions:

- Extensions to Fortran and C
- SPMD (Single Program Multiple Data), fork-join paradigm
- Shared-memory programming model
- Parallel constructs identifying a block of statements for parallel execution by one or more processes
- Worksharing constructs defining the units of work that shall be distributed among a team of processes
    - Iterative constructs to distribute entire block of statements to each process
    - Noniterative constructs to distribute several blocks of statements, one to each process
    - Ordered or unordered distribution of work
- Grouping constructs for grouping replicated code and worksharing constructs to reduce the synchronization overhead
- Synchronization:
    - Implicit at the beginning and the end of of parallel constructs, at the end of worksharing constructs, and at the end of grouping constructs
    - Explicit by using a critical section construct, a lock, an event, or a sequence
- Control not allowed to be transferred in or out of an enclosing parallel construct, worksharing construct, grouping construct, or critical section construct
- Attribute to classify a data object as not available, private, or shared

Useful to NAS: Maybe
(Emerging standard,
(User experience needed)

Platforms: Vendors participating in definition of proposed standard: CRAY Computer Co., CRAY Research In., DEC, IBM, Sun, NEC, KAI, Alliant

Operating System: Provided by each platform

Languages Supported: Fortran, C

Languages Used in Implementation: Vendor dependent

Graphic User Interface: None

Cost: Vendor dependent

Supplier: The specification is supplied by ANSI Technical Committee X3H5

Contact: anonymous ftp to lynx.cs.orst.edu in pub/x3h5

See reference [5] [6] [7]

1.1.4 ADAPTOR (Automatic DAta Parallelism TranslatOR)

Functions:

- Extensions to Fortran 77
  - Array syntax as defined by Fortran 90
  - Parallel loops (forall)
  - Data layout/distribution directives
- Libraries for message passing and for global operations of distributed arrays
- Source-to-source translation to generate Fortran 77 programs with message passing
- Support for both interactive mode and batch mode

Useful to NAS: No
(Similar to HPF,
Useful concepts are likely to be
absorbed in standardization effort in USA)

Platforms: CM5, KSR-1, iPSC/860, Alliant FX/2800,
A network of workstations using PVM,
Parsytec GCel, Meiko Concerto

Operating System: Supported by the platforms

Languages Supported: CM Fortran, Subset of Fortran 90

Languages Used in Implementation: C, GMD compiler generator

Graphic User Interface: Athena widgets, X-Windows system

Cost: None
ftp.gmd.de (129.26.8.90)
in subdirectory gmd/adapt

Supplier: GMD, I1.HR, Schloss Birlinghoven,
D-5205 St. Augustin, West Germany

Contact: Dr. Thomas Brandes
(49) 2241 - 14/2492
brandes@gmdzi.gmd.de

See reference [8]

1.1.5 P-LANGUAGES

Functions:

- Extensions to C and Fortran
- A shared memory programming model for associative and commutative binary operations on message-passing machines
- Single program multiple data model
- Source-to-source transformation
- Deadlock prevention
- Dependence analysis to achieve overlapped communication and computation
- Communication overhead reduction by consolidating many communication statements into one and hence increasing average message size

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support all NAS platforms,
Research project)

Platforms: iPSC/2, iPSC/860, Delta, KSR-1, Ncube 2

Operating System: Provided by each platform

Languages Supported: PC, Pfortran

Languages Used in Implementation: C, Lex, and Yacc

Graphic User Interface: None

Cost: $10,000
Academic discounts available
Manual available via anonymous ftp from
karazm.math.uh.edu

Supplier: Department of Mathematics
691 Phillip Guthrie Hoffman Hall
University of Houston
Houston, TX 77204--3476

Contact: L. Ridgway Scott
(713) 743-3445
Scott@UH.EDU

See reference [9]

1.1.6 FORCE

Functions:

- Extensions to Fortran for shared-memory multiprocessors
    - Statically and dynamically scheduled parallel loops
    - Parallel CASE statements
    - Barriers
    - Critical sections
    - A construct for requesting processors to execute different sections of code (functional parallelism)
    - Operations to access user-defined asynchronous variables (e.g. Produce, Consume, Void, Copy, and Isfull)
- Translation of a Force program to a Fortran 77 program with system dependent parallel construct.

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Shared memory only,
Research project)

Platforms: Y-MP, CRAY2, KSR1, Encore,
Sequent, Convex, Alliant

Operating System: Provided on each platform

Languages Supported: Force

Languages Used in Implementation: Fortran 77 and C

Graphic User Interface: None

Cost: None

Supplier: Computer Systems Design Group
Electrical and Computer Engineering
University of Colorado
Boulder, Colorado

Contact: Dr. Harry Jordan
(303) 492-7927
harry@boulder.colorado.edu

Dr. Gita Alaghband
(303) 556-2940
gita@boulder.colorado.edu

See reference [10] [11]

1.1.7 OBJECT-ORIENTED FORTRAN

Functions:

- Extensions to Fortran to support declaration, creation and management of objects
- A preprocessor to translate these constructs into standard Fortran 77
- A library of routines for message passing and object management
- An interface to C++
- Management of parallel execution of instancies of objects

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Not all NAS platforms supported
Research project)

Platforms: Intel iPSC/860, Delta, SGI Power Iris, SGI Personal Iris, IBM RS6000, Sun

Operating System: Provided by each platform

Languages Supported: Fortran77, C++

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: Engineering Research Center for Computational Field Simulation
Mississippi State University/ National Science Foundation
PO Box 6176
Mississippi State University, MS 39762

Contact: Donna Reese
(601) 325-2656
dreese@erc.msstate.edu

See reference [12]

1.1.8 VIENNA FORTRAN COMPILATION SYSTEM

Functions:

- Source-to-source translation from Vienna Fortran or Fortran 77 to Fortran with explicit message passing
- Code generation for supported target machines
- Automatic parallelization and vectorization
- A batch command language to enable the creation of batch files which may be automatically applied to a Fortran program
- A set of analysis services and a transformation catalog for interactive user input to guide the system through the parallelization process
- Automatic recording of the sequence of transformations executed and automatic execution of the sequence in the batch mode

Useful to NAS: No
(Not all NAS platforms supported,
Useful concepts are likely to be
absorbed in standardization effort in USA)

Platforms: Intel iPSC-860, SUPRENUM supercomputer,
Genesis-P machine, all distributed memory
multiprocessors on which PARMACS (Version 5.0)
runs. SUN SPARCstation with a minimum of 8
Megabytes disk storage,
(Additional disk space is required for the parallelization of user code.)

Operating System: Provided by each platform

Languages Supported: Vienna Fortran, Fortran 77

Languages Used in Implementation: GNU C

Graphic User Interface: X11R5, OSF/Motif

Cost: None

Supplier: University of Vienna,
Institute for Statistics and Computer Science
Bruenner Str. 72
A-1210 Vienna
Austria

Contact: Dr. Peter Brezany
+43-222-392647-227
brezany@par.univie.ac.at

See reference [13 14 15]

1.1.9 FORTRAN D

Functions:

- Extensions to Fortran 77 or Fortran 90:
    - A data decomposition directive for declaring an abstract problem domain (index domain) which may also be considered a virtual processor set
    - An array alignment directive for mapping arrays onto the problem domain
    - A data distribution directive for grouping elements of the decomposition and aligned arrays, and for mapping them to the parallel machine (Each dimension can be local or distributed in a block, cyclic, or block-cyclic manner.)
    - A control directive for deterministic parallel loop execution
- Compiler optimization:
    - Symbolic and dependence analysis
    - Data and computation partitioning
    - Overhead reduction by combining messages based on data dependences
    - Latency hiding by overlapping communication with computation
    - Collective communication exploitation (e.g. broadcast & reductions)
    - Parallelization of reductions and pipelined computations
    - Interprocedural reaching decompositions calculation
    - Efficient one-pass interprocedural compilation
- Translation of a Fortran D program into an SPMD Fortran 77 message-passing program

Useful to NAS: Maybe
(Technology development prototype for HPF,
Useful for tool development)

Platforms: Sun Sparc, IBM RS6000
Generates code for IPSC/860

Operating System: Provided by each platform

Languages Supported: Fortran77

Languages Used in Implementation: C and C++(g++ compiler)

Graphic User Interface: X11R4

Cost: $150 for site license

Supplier: Center for Research on Parallel Computation
Dept of Computer Science
Rice University

Contact: Theresa Chatman
(713) 527-6077
tlc@cs.rice.edu

See reference [16] [17]

1.1.10 FORTRAN M

Functions:

- Extensions to Fortran 77 for parallelism at task level
  - Constructs for explicit declaration of communication channels to plug together program modules called processes (Modularity)
  - Capability of encapsulating common data, subprocesses, and internal communication as processes (Modularity)
  - Restricted operations on channels to guarantee deterministic execution, even in dynamic computations that create and delete processes and channels (Safety)
  - A non-deterministic construct for specifying time dependent actions
- Type checking for channels at compiler time (Safety)
- Tools to specify the mapping of processes to virtual processors (Architecture Independence: separate the specification that influences only performance from those that influence correctness)
- A compiler to optimize communication and computation (Efficiency)

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support NAS platforms,
Functional parallelism only,
Research project)

Platforms: Sequent Symmetry, Sun sparc, NeXT
Distributed memory ports scheduled

Operating System: Dynix V3.1.4 with FastThreads thread library
SunOS 4.1.1, NeXTStep 2.1 or 3.0

Languages Supported: Fortran M

Languages Used in Implementation: C and Perl

Graphic User Interface: None

Cost: None

Supplier: Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Ill.

Contact: Ian Foster
(708) 252-4619
fortran-m@mcs.anl.gov

See reference [18]

1.1.11 CODE 2.0

Functions:

- A large-grain dataflow language for developing parallel programs.
    - Graphical interface for users to draw communication structure of programs
    - Nodes for sequential computations defined as calls to routines expressed in a sequential language
    - Arcs for data-flow dependences between nodes
    - User defined firing rules for nodes
    - Mechanism for controlled use of shared variables
    - User defined types.
- Support for hierarchical program development
- Support for program graphs whose topology is determined at runtime
- Automatic program performance instrumentation

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support NAS platforms,
Functional parallelism only,
Research project)

Platforms: Sequent Symmetry, Sun 4 workstations
(Plan to port to IBM RS/6000, and produce
sode for Sequents, Intel iPSC/860, and networks
of workstations)

Operating System: Provided by each platform

Languages Supported: Code 2.0 (Sequential code should be in C)

Languages Used in Implementation: C++

Graphic User Interface: X11R4 or X11R5

Cost: TBD

Supplier: Dept. of Computer Sciences
University of Texas at Austin

Contact: James C. Browne
(512) 471-9584
browne@cs.utexas.edu

Peter Newton
(512) 471-9735
newton@cs.utexas.edu

See reference [19]

1.1.12 GRIDS

Functions:

- A computing environment for grid-based numerically intensive computation
    - Declarative language for overall control of solution method
    - Topology description separated from computational algorithms
    - Grid constructs as extensions to Fortran for computational algorithms
- A runtime system which exploits knowledge of the parallelism inherent in the problem and the grid based solution methods (Explicit parallel programming not needed)
- Support for regular and irregular grids
- A preprocessor to translate a Grids code (topology, declarative part, and extended Fortran procedures) to standard Fortran

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support NAS platforms,
Research project)

Platforms: Networks of IBM RS6000

Operating System: AIX 3.2

Languages Supported: Fortran77

Languages Used in Implementation: C

Graphic User Interface: X11R5

Cost: $400
$135 for educational and research institutions

Supplier: Institute for Parallel and Distributed
High Performance Systems (IPVR)
University of Stuttgart
Breitwiesenstr. 20-22
W-7000 Stuttgart 80
Germany

Contact: Prof. Andreas Reuter
(+49) 711 7816 449
Andreas.Reuter@informatik.uni-stuttgart.de

See reference [20]

1.1.13 PCP/PFP (Parallel C/Fortran Preprocessor)

Functions:

- Extensions to C and Fortran.
- Fork-join parallel programming model on shared-memory multiprocessors
- Constructs to group processor resources into teams of processors
- Synchronization-free control constructs
- Low-overhead control constructs (On the order of a couple of local memory references)

Useful to NAS: No
(Does not support NAS platforms,
Limited programming paradigm,
Research project)

Platform: BBN TC2000

Operating System: UNIX

Languages Supported: C, Fortran77

Languages Used in Implementation: C Lex and Yacc

Graphic User Interface: None

Cost: None

Supplier: Lawrence Livermore National Laboratory L-560
P.O.Box 808 Livermore CA 94550

Contact: Brent Gorda
(510) 294-4147
brent@igor.nersc.gov

See reference [21] [22]

1.1.14 PDDP (The Parallel Data Distribution Preprocessor)

Functions:

- Extensions to Fortran
  - Data parallel programming model on shared memory systems
  - Array syntax
  - Directives for data distribution
- Library functions for global operations
- Translation of a PDDP program into a Fortran program using PFP (see entry for PCP/PFP)

Useful to NAS: No
(Does not support NAS platforms,
Limited programming paradigm,
Research project)

Platform: BBN TC2000

Operating System: UNIX

Languages Supported: Fortran77

Languages Used in Implementation: C, LEX and YACC

Graphic User Interface: None

Cost: None

Supplier: Lawrence Livermore National Laboratory L-560
P.O.Box 808 Livermore CA 94550

Contact: Brent Gorda
(510) 294-4147
brent@igor.nersc.gov

Karen Warren
(510) 422-9022

See reference [23] [24]

1.2 C-Based Languages

1.2.1 JADE

Functions:

- A declarative, data-oriented language for coarse-grain parallel programming
- Preservation of the abstractions of serial semantics and a single address space
- Constructs for specifying how a program written in a standard sequential, imperative programming language accesses data
- Translation of a C program with Jade constructs into a C program with calls to the Jade implementation
    - Dynamic interpretation of Jade specifications to determine which parts of the program can execute concurrently without violating the serial semantics
    - Generation of the data movement messages required to implement the abstraction of a single address space on distributed-memory machines

Evaluation:

(By Martin Rinard of Stanford University)

Applications:

- Water Code: Derived from the Perfect Club benchmark mdg. Simulates water in the liquid state
- String: Seismic application from the Department of Geophysics, Stanford University. Performs geophysical travel-time tomography. Reconstructs a velocity field from cross-well travel time data
- Volume Rendering: Department of Computer Science, Stanford University. Uses volume rendering to visualize CAT scan data sets.

Strong points:

- Jade's abstraction of serial semantics eliminates nondeterministic, timing-dependent bugs: all parallel executions of a Jade program deterministically generate the same result as the serial execution.
- Jade's abstraction of a single address space eliminates the need for programmers to manage the distribution of data across the parallel machine.
- Programmers can effectively use Jade to develop coarse-grain parallel programs that execute efficiently on a range of parallel architectures.

Weak points:

- No support for writing nondeterministic programs
- No user control for the low-level execution of the program for maximal efficiency

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support all NAS platforms,
Research project)

Platforms: Intel iPSC/860, Stanford DASH, SGI 4D/240, DEC, Sun, SGI

Operating System: Provided by each platform

Languages Supported: C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: Department of Computer Science
Stanford University

Contact: Martin Rinard
(415) 725-3722
martin@cs.stanford.edu

See reference [25] [26] [27]

1.2.2 CHARM

Functions:

- Extensions to C for shared-memory and message-passing systems
- Message-driven, non-blocking, execution for latency tolerance
- Reusable modules and libraries
- Information sharing abstractions
- A notation for specifying dependencies between messages and pieces of computation
- Generation of C code with machine-specific parallel constructs
- Dynamic and static load balancing
- Trace generation and visualization for performance optimization

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support all NAS platforms,
Research project)

Platforms: iPSC/860, iPSC/2 NCUBE, Sequent Symmetry,
Encore Multimax, Networks of (Unix) workstations,
(Being ported to CM5)

Operating System: Provided by each platform

Languages Supported: C (C++ soon)

Languages Used in Implementation: C

Graphic User Interface: X Motif for the performance Visualization tools.

Cost: None
Anonymous ftp with conditions

Supplier: Department of Computer Science
University of Illinois at Urbana Champaign

Contact: L.V. Kale
kale@cs.uiuc.edu

See reference [28 29]

1.2.3 DINO

Functions:

- C extensions for data parallel programming
  - Process creation, management, and termination
  - Process communication and synchronization
  - Global operations
  - Data partitioning and mapping
- SPMD Paradigm

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support all NAS platforms,
Research project)

Platforms: iPSC/2, iPSC/860,
network of Sun workstations,
(A Sun required for the front-end of the compiler)

Operating System: Sun/OS for the front end

Languages Supported: C

Languages Used in Implementation: C, Pascal

Graphic User Interface: None

Cost: None

Supplier: University of Colorado

Contact: Bobby Schnabel
bobby@cs.colorado.edu

See reference [30] [31]

1.2.4 HYPERTOOL

Functions:

- C extensions for parallel programming
  - Notation to define a procedure as an indivisible unit of computation to be scheduled on one processor.
  - Single assignment property of any parameter of a procedure.
  - Directives IN and OUT for specifying whether the parameter is read-only or read-write.
  - Dataflow firing rule for procedure scheduling. (A procedure can be executed iff all input of the procedure are available.)
- Task graph generation from the data flow between procedures
- Translation of a HyperTool code to a C code with message-passing between procedures
- Static scheduling of processes to processors on distributed-memory machines
- Performance estimates and measurements for parallel programs (speedup, efficiency, suspension time, communication time, etc)

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support all NAS platforms,
Functional parallelism only,
Research project)

Platforms: iPSC/2, iPSC/860
Tool runs on: SPARC

Operating System: Provided by each platform

Languages Supported: C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: Department of Computer Science
State University of New York
Buffalo, NY 14260

Contact: Min-You Wu
(716) 645-3185
wu@cs.buffalo.edu

See reference [32]

1.3 Object-Oriented Languages

1.3.1 PC++ (Parallel C++)

Functions:

- A data-parallel extension to C++
- Collection class for concurrent aggregates (structured sets of objects that are distributed over the processors and memories in a parallel system)
- Concurrent application of arbitrary functions to the elements of arbitrary distributed, aggregate data structures
- Collection alignment and distribution: (similar to HPF)
    - Template objects for specifying distributed collections in a given computation in relation to each other
    - An alignment object for mapping a collection to a template.
- Kernel class:
    - A global name space for the collection elements
    - Method for managing parallelism and accesses to collection elements.
- Collection library to provide a set of primitive algebraic structures that may be used in scientific and engineering computations
    - Distributed array for Fortran 90 style arrays and array operations
    - Distributed matrix and distributed vector class for BLAS-3 level operations
    - Blocked distributed matrix and blocked distributed vector for exploiting well tuned sequential class libraries for matrix vector computations
    - Grid classes for finite difference and finite element applications.
    - Dynamic structures (trees, unstructured meshes, dynamic lists, and distributed queues)
- A preprocessor that translate a pC++ code into C++ code

Useful to NAS:	No at present time Maybe in the future (User experience needed, Does not support Fortran, Does not support all NAS platforms, Research project)
Platforms:	CM-5, Paragon, Sequent, BBN TC2000, all workstations
Operating System:	Supported by each platform
Languages Supported:	pC++
Languages Used in Implementation:	C++, C
Graphic User Interface:	None
Cost:	None
Supplier:	Indiana University
Contact:	Dennis Gannon (812) 335-5184 gannon@cs.indiana.edu

See reference[33]

1.3.2 MENTAT

Functions:

- Extensions to C++ for parallel programming:
  - Data-driven computation model
  - User specifications for parallelism (by identifying the object classes whose member functions are of sufficient computational complexity to allow efficient parallel execution)
- A compiler which automatically detects the data and control dependencies between Mentat class instances involved in invocation, communication, and synchronization
- A run-time system:
  - Support for method invocation by remote procedure call (The compiler decides where and whether the caller needs to block, and generates code for required synchronization and communication.)
  - Program graph construction
  - Communication and synchronization management
  - Support for a graph-based, data-driven computation model in which the invoker of an object member function need not wait for the result of the computation, or receive a copy of the result.

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support all NAS platforms,
Research project)

Platforms: iPSC/2, iPSC/860 (gamma),
SGI Iris, Sun 3 network, Sun 4 (Sparc) network,
(In progress: Paragon, CM-5, RS/6000)

Operating System: Provided by each platform

Languages Supported: MPL - an extended C++

Languages Used in Implementation: C++

Graphic User Interface: None

Cost: None
available by ftp uvacs.cs.virginia.edu
in pub/mentat

Supplier: Department of Computer Science
University of Virginia
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903

Contact: Andrew Grimshaw
(804) 982-2204
grimshaw@virginia.edu
mentat@virginia.edu

See reference [34] [35]

1.3.3 COOL (Concurrent Object Oriented Language)

Functions:

- Extensions to C++ designed to express task-level parallelism for shared memory multiprocessors
    - Declaration of C and C++ member functions as parallel to express concurrency
    - A shared address space for communication between parallel functions
    - Monitors for synchronization between shared objects
    - Condition variables for event synchronization
    - A construct for fork-join style synchronization at task level
    - Abstractions for programmer-supplied information about the data reference patterns of a program
- A yacc-based translator that translates a COOL program into a C++ program
- A runtime system that schedules tasks and distributes the data to increase locality (based on the data reference information), and balances the load
- MTOOL for identifying memory system and other performance bottlenecks in programs
- MemSpy, a simulation-based tool, to study the memory system behavior in detail and identify the causes of poor performance
- Tango, a simulation-based tool that allows us to study the program performance under different memory hierarchies

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support NAS platforms,
Research project)

Platforms: Encore MultiMax, Stanford DASH multiprocessor, SGI workstations

Operating System: UNIX

Languages Supported: COOL

Languages Used in Implementation: C, C++

Graphic User Interface: None

Cost:	None anonymous ftp from cool.Stanford.EDU
Supplier:	Computer Systems Lab Stanford University
Contact:	Rohit Chandra (415) 725-3648 rohit@cool.Stanford.EDU

See reference [36] [37]

1.3.4 CC++ (Compositional C++)

Functions:

- Extensions to C++ for compositional parallel programming
    - Statements for creating new threads of control upon entering a block of statements (key word *par* proceeding compound C++ statements)
    - A statement for parallel threads whose number is determined at run time (*parfor*)
    - A statement for starting a new thread and returning immediately to the calling process (*spawn*)
    - A sync variable for synchronization and communication between parallel threads
    - A logical processor object for regular C++ global declarations (no shared address space outside a logical processor object)
- Data parallel, task parallel, and object parallel
- Shared memory and message passing

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support NAS platforms,
Research project)

Platform: Sequent Symmetry, Sun, SGI

Operating System: UNIX

Languages Supported: CC++

Languages Used in Implementation: C, C++

Graphic User Interface: None

Cost: None
ftp csvax.cs.caltech.edu
in comp

Supplier: California Institute of Technology

Contact: Carl Kesselman
(818) 356-6517
carl@vlsi.cs.caltech.edu

1.3.5 MELDC

Functions:

- A C-based object-oriented coordination programming language
- Support for a wide range of high-level features for programmers to cope with problems in designing open systems
- Support for investigation of the language architecture without modifying the language internals
- A MeldC variant of the gdb debugger

Useful to NAS:	No (Does not support NAS platforms, For research in language design)
Platforms:	Sun4, DecStations
Operating System:	SunOS 4.1, Ultrix 4.2
Languages Supported:	MeldC
Languages Used in Implementation:	C and assembly
Graphic User Interface:	None
Cost:	None
Supplier:	Programming System Laboratory Department of Computer Science Columbia University
Contact:	Prof. Gail E. Kaiser MeldC@cs.columbia.edu

See reference [38] [39] [40]

1.3.6 PARALLAXIS

Functions:

- Extensions to Modula-2 for data parallel (SIMD) programming
- Means to describe the virtual parallel machine, the number of identical processors with local memory, the names of communication ports, and the network topology for data exchange among PEs
- Simulators on workstations and PCs for developing and debugging parallel programs
- Compilers for massively parallel computers
- Trace generation for Parallaxis programs
- Displays for the load of the processing elements as a function of the execution time
- Displays for the connection structures between PEs in a Parallaxis program

Useful to NAS: No
(Modula-2 is not considered by NAS)

Platforms: CM2, MasPar MP-1,
Sun3, SPARCstation/Sun4, DECstation,
HP/Apollo 700, IBM RS-6000,
Apple Macintosh, IBM-PC compatibles

Operating System: SunOS Release 4.1, DEC ULTRIX V4.2,
HP-UX 8.07, IBM AIX Version 3

Languages Supported: Parallaxis

Languages Used in Implementation: C

Graphic User Interface: X11R5

Cost: None
anonymous ftp:
ftp.informatik.uni-stuttgart.de
(129.69.211.1)
in pub/parallaxis

Supplier: Institute for Parallel and Distributed Supercomputers,
Univ. Stuttgart, Breitwiesentr.
20-22, D-7000 Stuttgart 80,
Germany

Contact: Dr. Thomas Braunl
+49 (711) 781-6390
braunl@informatik.uni-stuttgart.de

See reference [41, 42]

1.3.7 MODULA-2*

Functions:

- Extensions to Modula-2
  - An arbitrary number of processes operating on data in the same, single address space
  - Synchronous and asynchronous parallel computations
  - Arbitrarily nested parallelism
  - All abstraction mechanisms of Modula-2
- Automatic process and data distribution by the compiler

Useful to NAS: No
(Modula-2 is not considered by NAS)

Platforms: MasPar MP1, a network of SUN4,
single SUN4 station
(DEC workstations soon)

Operating System: UNIX

Languages Supported: Modula-2*

Languages Used in Implementation: Modula-2 (MOCKA compiler), C
COCKTAIL compiler generation tools

Graphic User Interface: None

Cost: None
Anonymous ftp from iraun1.uka.de
in pub/programming/modula2star

Supplier: Institut fuer Programmstrukturen und Datenorganisation
Fakultaet fuer Informatik, Universitaet Karlsruhe
Postfach 6980, W-7500 Karlsruhe 1, Germany

Contact: Ernst A. Heinz
heinze@ira.uka.de

Paul Lukowicz
lukowicz@ira.uka.de

Michael Philippsen
lukowicz@ira.uka.de

++49/(0)721/6084386

See reference [43 44]

1.4 Others

1.4.1 STRAND88

Functions:

- A Prolog-like parallel processing language and development environment
- Interfaces for calling C and/or Fortran sequential routines
- Tools that monitor processor and communication load, and visualize the data
- Library of parallel services, including worker-manager task management and producer-consumer communication streams
- A symbolic, single-stepping debugger
- The Strand Abstract Machine:
    - Support for specification of user application topology
    - Automatic mapping of the application topology into the topology of the distributed-memory machine or a network of computers ("Virtual Topology" facility)

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support all NAS platforms,
Functional parallelism only)

Platforms: iPSC/860, iPSC2/386, nCube 2,
Transputer/Helios, Sequent Balance/Symmetry,
Alliant FX/2800, Encore Multimax and 91XX
Series, Sun 600MP Multiprocessing Workstation,
Sun SparcStation, Meiko Computing Surface, HP9000
MIPS RISCstation, Cogent XTM Workstation,
MacII, IBM PS/2, RS/6000, NeXT, Pyramid,
TI TMS320C40 Digital Signal Processor

Operating System: Provided by each platform

Languages Supported: Strand88, Fortran, C

Languages Used in Implementation: C, Assembly

Graphic User Interface: X11 version provided with the Sun OS

Cost: Approximately $1000/node commercial
60% educational discount

Supplier: Parallel Performance Group, Inc.
3368 Governor Drive, Suite F269
San Diego, CA 92122

Contact: Dr. Stuart Bar-On
(619) 737-973
strand@ppg.strand.com
4956839@mcimail.com

See reference [45]

1.4.2 SISAL

Functions:

- A general purpose functional language for parallel numeric computation
- Constructs to express scientific algorithms in a form close to their mathematical formulation with no explicit program control flow
- An interface that allows Sisal programs to call C and Fortran and allows C and Fortran programs to call Sisal
- Automatic exploitation of parallelism
- An optimizing compiler
- A symbolic debugger

Evaluation:

(By Chris Hendrickson and Dave Hardin of Lawrence Livermore National Laboratory)

Strengths:

- Programs can be written in Sisal faster than they can be written in conventional imperative languages.
- Programs in Sisal tend to be shorter in length.
- A Sisal program is executable on single as well as multiple processors, with no code changes needed.
- Porting between machines involves only recompilation.
- Most Sisal programs outperform equivalent Fortran programs compiled using automatic vectorizing and parallelizing tools.

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support message-passing)

Platforms: Cray X/MP, Y/MP, Cray-2, C90,
Alliant, Encore, Sequent,
Suns, Sparcs, IBM PCs, Macintoshes, Vaxes

Operating System: UNIX (BSD or AT&T)

Languages Supported: Sisal

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: Computing Research Group
Lawrence Livermore National Laboratory

Contact: John T. Feo
(510) 422-6389
feo@llnl.gov

Thomas M. DeBoni
(510) 423-3793
deboni@llnl.gov

See reference [46 47 48 49]

1.4.3 PCN (Program Composition Notation)

Functions:

- A C-like language for writing parallel programs:
    - Facilities for constructing a parallel program by combining simpler components in parallel, sequential, and choice blocks
    - Support for components written in C, or Fortran (Components can be written in PCN)
    - Constructs for specifying how computation is mapped to physical processors
    - Facilities for reusing parallel code (templates)
    - Standard libraries for I/O, mapping, etc
- A highly portable compiler:
    - Message-passing code generation for distributed memory machines
    - Shared-memory reads and writes on shared memory machines
    - An interface to the C preprocessor for macros, and conditional compilation
- Integrated source-level debugger for PCN programs
- Performance analysis tools for PCN programs
    - Upshot: event trace collection, analysis, and visualization
    - Gauge: profile collection, analysis, and visualization

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support all NAS platforms,
Functional parallelism only)

Platforms: Intel iPSC/860, Delta, Sequent Symmetry,
Sun 3, Sun 4, NeXT, IBM RS/6000, HP 9000
(series 800, 700, and 300),
ECstation 5000 (and 3100), SGI Iris

Operating System: Provided by each platform

Languages Supported: PCN, C, Fortran

Languages Used in Implementation: C

Graphic User Interface: X11R5 for performance analysis tools

Cost: None
anonymous ftp from info.mcs.anl.gov
in directory pub/pcn

Supplier: Argonne National Laboratory

Contact: Ian Foster
(708) 252-4619

Steve Tuecke
(708) 252-8711

pcn@mcs.anl.gov

See reference [50] [51]

1.4.4 LINDA

Functions:

- Language extensions to C and Fortran for parallel programming
- A coordination langage for creating parallel or distributed applications via a virtual shared memory paradigm
- Source level debugger (TupeScop)
  - Graphical user interface
  - Based on the shared memory
  - Interface for processes to attach to standard source-level debuggers (e.g. dbx)
- Consistency checking for tuple space usage
- Monitors message traffic and moves Linda run time library to reduce the traffic
- Tuple space usage visualization

Evaluation:

(By Alan Karp of HP, the work was done when he was with IBM)[52]

Application:

Linpack 100 code on 5 IBM RS/6000s over Ethernet
and 3 IBM RS/6000s over the Serial Link Adapter (SLA)
fiber optical channel. The experiment was finished in
Aug., 1991

Strengths:

- Linda does what it says it will do and does it well.
- The code is stable, does not crash the system, has only minor bugs
- Overhead of using the distributed tuple space is small in this experiment. The message latency and bandwidth are the same as measured in a program using Express (Done by H. Wang of IBM)

Weaknesses:

- The network performance over the SLA is disappointing. In the 6ms it takes to get a small tuple from another machine one can have executed 60,000 floating point operations. Even accessing a local tuple consumes the time needed to do 3,000 flops.

Useful to NAS: Maybe
(User experience needed,
Does not support all NAS platforms)

Platforms: iPSC/2, Sun, IRIS, IBM RS6000, Apollo, Encore, Sequent

Operating System: Provided by each platform

Languages Supported: C, Fortran77

Languages Used in Implementation: C, Fortran

Graphic User Interface: X11R4 for debugger

Cost: $4,995/10 workstation
$30,000 for iPSC/2

Supplier: Scientific Computing Associates Inc.

Contact: Sudy Bharadwaj
(203) 777-7442
software@sca.com

See reference [53] [54]

1.4.5 TOPSYS (TOols for Parallel SYStems)

Functions:

- Autonomous objects for parallel programming (tasks, semaphores, and mailboxes)

- A location-transparent message passing library MMK, i.e. The user needs only specify the name of the receiving object (task, mailbox, semaphore), not a physical resource (such as processor).

- Parallel debugger DETOP:

    - Inspecting objects by their names used in the source code
    - Observing and altering parallel execution at run time
    - Breakpoint types (control flow, data flow, concurrency predicates) selectable by user
    - Distributed breakpoints
    - Trace types (data traces, execution traces, concurrency traces, traces of object interaction) selectable by user
    - Display of source code and on-line help
    - Single step mode (procedure steps, statement steps)
    - Global view of distributed system
    - Monitoring of communication

- Parallel performance monitor PATOP:

    - System level, Node level, Object level
    - Specifying objects to be monitored by names used in source code
    - User interaction at run time

- Parallel program visualizer VISTOP:

    - Specifying objects to be monitored by names used in source code
    - User interaction at run time
    - A menu driven selection of objects to be animated in iconified form or deiconified with additional information
    - Scrolling
    - Automatic replay at variable speed

- Process-processor mapping according to user specification

- Dynamic load balancing

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support all NAS platforms and OS)

Platform: iPSC/2, iPSC/860, PARSYTEC SC, EDS

Operating System: NX/2, Parix, CHORUS

Languages Supported: C, Fortran77

Languages Used in Implementation: C, C++

Graphic User Interface: X windows

Cost: for iPSC systems site license $700
for Parsytec systems: license by Parsitex required

Supplier: Institut fur Informatik
Technische Universit"at Munchen
P.o.b. 20 24 20
D-8000 Munchen 2
Germany

Contact: Prof. Dr. A. Bode
++49-89-2105-8240
bode@informatik.tu-muenchen.de

See reference [55 56]

1.4.6 CAPER (Concurrent Application Programming Environment)

Functions:

- A visual programming tool to assist parallel programming in the large
  - Support for medium-grained parallel programming
  - A reusable block methodology with data flowing between blocks to encourage building-block approach to parallel programming
  - Facilities for expressing communication
  - Generic parallel algorithms to help in parallelism extraction (e.g. Sort, Prefix, Search, and Matrix Algorithms)
- A preprocessor:
  - Code generation for data distribution, parallel I/O, and distributed data restructuring
  - Code generation for communication, task invocation and synchronization
- Simple debugging facility for examining processes and communication states
- Basic performance monitoring

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support NAS platforms)

Platforms: HPC multiprocessor, NCR 3450, NCR 3600, network of SUN-3, SPARC workstations

Operating System: UNIX or VORX

Languages Supported: C, C++, Concurrent C
(Planned: HP FORTRAN)

Languages Used in Implementation: C, C++, Concurrent C

Graphic User Interface: X11 R3-R5

Cost: Available internally to AT&T since 1989
Will be available to external users
in Dec. 1993 to selected customers
at nominal cost

Supplier: AT&T/NCR

Contact: Binay Sugla
(908) 949-0850
sugla@research.att.com

See reference [57]

1.4.7 VISAGE (VISual Attributed Graph Environment)

Functions:

- A graph-based parallel programming environment for functional decomposition on distributed memory multiprocessors

- A large-grain dataflow based graphic language for prototyping:

  - Specification of task dependence graph
  - Graph annotation with parameters such as message length, messages distribution, probability of communication

- A graphical, visual editing environment

  - Specification for topology
  - Task to processors mapping

- Tools for performance prediction and execution behavior simulation

- An object-oriented, structured editor for continuous modification of the generated prototype into the actual code

- Run-time support:

  - On-line display of 3 dimensional causality graph
  - Observation of the concurrency set and dead-lock

- Post-mortem analysis:

  - Automatic program instrumentation to generate trace
  - Display of statistical information with multiple views
  - 3-dimensional manipulations of the graphs

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran,
Does not support NAS platforms,
Functional parallelism only,
Research project)

Platforms: Transputers running 3LC, Meiko, Mach
based environments (the i486)
Front end on Silicon Graphics
(SUN, iPSC2, Paragon planned)

Operating System: Meiko's CS-tools, 3LC + extensions, Mach

Languages Supported: C and its parallel extensions

Languages Used in Implementation: C, GL on Silicon Graphics

Graphic User Interface: GL

Cost:	None
Supplier:	Elec. Engineering Dept. The Technion, Israeli Institute of Technology Technion City, Haifa Israel
Contact:	Dror Zernik 972-4-294641 or 972-4-323041 dror@ee.technion.ac.il

See reference [58, 59]

1.4.8 SR (Synchronizing Resources)

Functions:

- A language for writing parallel programs with multiple threads of control connected in arbitrary fashion
- Support for multiple interprocess communication paradigms (local and remote procedure call, rendezvous, message passing, dynamic process creation, multicast, semaphores, and shared memory)
- Interface to C
- True parallel execution on multiprocessors and simulated parallelism on uniprocessors

Useful to NAS: No
(User experience needed,
Does not support Fortran,
Does not support NAS platforms,
Research project)

Platforms: Sequent Symmetry, Sun4, Sun3, DECstation,
SGI Iris, HP RISC and 9000/300, NeXT,
IBM RS/6000, DEC VAX, DG AViiON

Operating System: UNIX

Languages Supported: SR

Languages Used in Implementation: C, Yacc, Lex

Graphic User Interface: None required
Optional interface to X windows included

Cost: None
ftp from cs.arizona.edu in /sr

Supplier: Department of Computer Science
University of Arizona
Tucson, Arizona 85721

Contact: sr-project@cs.arizona.edu
(602) 621-8448

See reference [60]

1.4.9 PROLOG-D-LINDA

Functions:

- Extensions to SICStus Prolog
  - A distributed tuple space that uses unification for matching
  - Prolog style deduction in the tuple space
  - A control hierarchy that provides remote I/O facilities for client processes

Useful to NAS: No
(Prolog is not considered at NAS)

Platforms: Networks of SUN Sparc and DEC stations
(Require SICStus Prolog 0.7 or 2.1, and
NFS or equivalent transparent access to shared files)

Operating System: SUN OS and NFS, Ultrix and NFS

Languages Supported: SICStus Prolog 0.7 and 2.1

Languages Used in Implementation: SICStus Prolog and C

Graphic User Interface: None

Cost: None
ftp from ftp.cs.uwa.edu.au
in pub/prolog-linda (SICStus 0.7 version)
ftp from coral.cs.jcu.edu.au
in pub/prolog-linda (SICStus 2.1 version)

Supplier: SICStus 0.7 version :
Department of Computer Science
The University of Western Australia
Western Australia

SICStus 2.1 version :
Department of Computer Science
James Cook University
Australia

Contact: Geoff Sutcliffe
+61 77 814622
geoff@cs.jcu.edu.au

See reference [61]

1.4.10 PARALLAX

Functions:

- A language for specifying a parallel program as a hierarchical large-grain dataflow diagram
- Tools for specifying a target machine as a planar graph of processors and network links
- Estimates of speedup, processor efficiency, utilization, and critical path based on the specification
- Heuristics to schedule the program design onto the target machine
- Gantt chart, speedup graph, bar charts for efficiency and resource utilization
- Simulation and animation of the program execution

Useful to NAS: No
(Does not support NAS platforms,
Functional parallelism only,
Simulation only,
Research project)

Platform: Macintosh

Operating System: Macintosh or A/UX

Languages Supported: Parallax

Languages Used in Implementation: Pascal

Graphic User Interface: Macintosh

Cost: None for researchers and educators

Supplier: Oregon State University

Contact: Ted Lewis
(503)-737-5577
lewis@cs.orst.edu

See reference [62 63]

2. Libraries

The fifteen tools described in this part try to achieve portability by providing libraries. Most of them support parallelism within an application, except CM which only supports parallelism at job level. Application-oriented high-level abstractions are provided in Canopy, whereas the others supports programming languages such as C and Fortran with parallel extensions. A few support programming in both shared-memory paradigm and message-passing paradigm; others focus on just one. Several of them extend the support for distributed memory machines to a network of computers. Associated debugging and performance tuning tools are provided by only a few of them.

2.1 EXPRESS

Functions:

- Library functions for parallelization
- Support for data and functional decompositions, client/server, and distributed database
- Automatic loop parallelization, data distribution, and domain decomposition
- Translation of Fortran 90 source code to Fortran 77
- Support for CM2 extensions
- An interactive distributed source and assembly level debugger
- Tools for performance optimization:
    - Program instrumentation
    - Run time profile used for guidance
    - Dynamic load balancing
    - Interactive memory access visualization
    - Post-mortem communication and event analysis
    - Communication and event monitoring
- Parallel I/O
- Hardware configuration management

Evaluation:

(By Doreen Cheng of NASA Ames Research Center, through testing)

Strengths:

- Provides extensive set of tools for message-passing machines (debugging, performance monitoring, load balancing and parallelization).
- Covers a broad range of hardware, operating system, and languages.

Weaknesses:

- Debugger, profiler, parallel I/O, graphics are not available on Y-MP, nor for Intel iPSC/860 with an SGI IRIS as frontend.
- Lacks support for interactive dependency analysis.

(By Donna Bergmark of Cornell Theory Center)

- Has been working on a cluster of RS/6000 workstations since about December 1992, and is on the verge of being put into production. Experience with this latest release is that is runs much faster than the earlier experimental one that was at Cornell when the initial version of this report was being prepared.

(By Bill Pearson of University of Virginia)

Application:

- A C program compares a set of protein sequences (typically 10 - 100) to a larger set of protein sequences (2,000 - 10,000) and calculates a similarity score using several algorithms that differ in speed over a 100-fold range. Absolute communications overhead is constant but relative communications overhead varies from >50% to <5%. (Twelve Sparc 4/40 were used.)

Conclusion:

- On problems where communications cost is significant, PVM (2.4.1) imposes substantially more overhead than Express (3.2.5) (For PVM, snd()/rcv() was used. For Express exwrite()/exread() was used.

Useful to NAS: Maybe
(User experience needed)

Platforms: CRAY X-MP (UNICOS), CRAY Y-MP (UNICOS)
Intel iPSC2, iPSC/i860, iWARP, DELTA,
IBM3090 (AIX), IBM ES9000
nCUBE 2, nCUBE 2E, nCUBE 2S

A network of
HP9000/700, IBM RS6000, PVS, RS6000 with Bit3 shared memory switch, SGI, Sun SPARCstations*, SPARCServer, SPARCengine2, SPARCserverXXXMP

Transputers including Archipel i860, Inmos, Microway, Parsytec, Quintek, Transtech i860, PC's and Sun's.

Operating System: NX on iPSC/860, Paragon OSF/1 on Paragon

Languages Supported: C, Fortran77

Languages Used in Implementation: Fortran 77, Fortran 90, C and C++

Graphic User Interface: X-Windows, Sunview, Postscript

Cost:	\$3,000 per Intel iPSC/860 \$15,000 per Y-MP \$1,500 for network of Suns 20% maintenance fee per year
Supplier:	ParaSoft
Contact:	Adam Kolawa (818) 792-9941

See reference [64]

2.2 PVM (Parallel Virtual Machine)
HeNCE (Heterogeneous Network Computing Environment)

Functions:

- Library routines that permits a network of heterogeneous computers (serial, parallel, and vector computers) to appear as one large computer
  - Process management
  - Message passing
  - Data conversion between different machine representations
- A programming environment
  - Graphic interface for users to explicitly specify the parallelism of a computation
  - Tools to automate, as much as possible, the tasks of writing, compiling, executing, debugging, and analyzing a parallel computation

Evaluation:

(By Donna Bergmark of Cornell Theory Center)

- Widely used since it is difficult to write a message passing program on a network of workstations

(By Glenn Kubena, Kenneth Liao, Larry Roberts of IBM)

Strengths:

- Widely used
- Simple and easy to use

Weaknesses:

- Lack of support for fault tolerance
- Lack of load balancing
- No receipt selectivity other than by message type

(By Bill Pearson of University of Virginia)

Application:

- A C program compares a set of protein sequences (typically 10 - 100) to a larger set of protein sequences (2,000 - 10,000) and calculates a similarity score using several algorithms that differ in speed over a 100-fold range. Absolute communications overhead is constant but relative communications overhead varies from >50% to <5%. (Twelve Sparc 4/40 were used.)

Conclusion:

- On problems where communications cost is significant, PVM (2.4.1) imposes substantially more overhead than Express (3.2.5) (For PVM, snd()/rcv() was used. For Express exwrite()/exread() was used.

Useful to NAS:	Maybe (User experience needed)
Platforms:	All Unix based machines
Operating System:	Unix
Languages Supported:	C, Fortran77
Languages Used in Implementation:	C
Graphic User Interface:	None for PVM, X11R4 for HeNCE
Cost:	None send email to netlib@ornl.gov in the message type: send index from pvm send index from hence
Supplier:	Oak Ridge National Laboratory University of Tennessee
Contact:	pvm@msr.epm.ornl.gov hence@msr.epm.ornl.gov

See reference [65 66 67 68]

2.3 P4

Functions:

- Subroutine library for parallel programming
- Monitors for the shared-memory model
- Message-passing for the distributed-memory model both on heterogeneous workstation networks and on parallel machines themselves
- Monitors and message-passing for cluster model
- Trace generation for performance monitoring

Evaluation:

(By Donna Bergmark of Cornell Theory Center)

- Not installed for general use because it was redundant with PVM
- Some experiments showed that it ran slower than PVM.

(By Timothy Mattson, Craig Douglas, and Martin Schultz of Intel)

- Round trip point-to-point communication is faster than PVM, C-Linda, and CPS.

Useful to NAS: Maybe
(User experience needed)

Platforms: CM-5, Intel Delta, iPSC/860, BBN TC-2000 and GP-1000, IBM 3090, Cray X-MP, Alliant FX/8, FX/2800, and CAMPUS, Sequent Symmetry (both Dynix and PTX), nCube Sun3, Sun4, IBM RS-6000, Stardent Titan, NeXT, DEC, SGI, HP

Operating System: Provided by each platform

Languages Supported: C, Fortran77

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None
info.mcs.anl.gov
pub/p4/p4-1.2.tar.Z

Supplier: Argonne National Laboratory

Contact: Rusty Lusk
(708) 252-7852
lusk@mcs.anl.gov

See reference [69 70]

2.4 TCGMSG (Theoretical Chemistry Group Message Passing Toolkit)

Functions:

- A message-passing library for both shared-memory parallel computers and distributed-memory parallel computers
  - The programming model and interface directly modeled after (a small subset of) the PARMACS (See PARMACS entry)
  - Support for communication over network through TCP sockets
  - Support for communication through shared memory if available
- Straightforward load balancing
- Data representation conversion for Fortran integer and double precision data types and C character data

Evaluation:

(By Timothy Mattson, Craig Douglas, and Martin Schultz of Intel)

- Round trip point-to-point communication is faster than P4, PVM, C-Linda, and CPS.

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support all NAS platforms)

Platforms: Intel Delta, iPSC/860,
KSR1, Alliant FX/8/80/800/2800, ARDENT,
Convex C220, IBM R6000, HP, Sun, Dec, SGI

Operating System: KSR OS (KSR1)
Concentrix 2800 2.2 (Alliant)
Sun O/S 4.0 or above (SUN)
IRIX 4.0 (SGI)
ULTRIX (DEC)
Stardent Titan O/S 2.2 (ARDENT)
ConvexOS V8.1 (Convex)
AIX 3.1 (IBM)
HP-UX A.B8.05 (HP)

Languages Supported: Fortran, C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None
anonymous ftp from ftp.tcg.anl.gov
in pub/tcgmsg/tcgmsg.4.02.tar.Z

Supplier:	Mail Stop K1-90 Battelle Pacific Northwest Laboratory P.O. Box 999, Richland WA 99352
Contact:	Robert J Harrison (509)-375-2037 rj_harrison@pnl.gov

See reference [71] [70]

2.5 CANOPY

Functions:

- A runtime library for developing efficient grid-oriented algorithms on massively parallel MIMD systems

- SPMD programming paradigm

- Application-oriented programming concepts:

    - Grids: with connectivity along directions (Grid structure can be pre-defined, arbitrary, and user-defined.)
    - Sites on the grid: with neighboring sites defined by the connectivity
    - Fields of data: consisting of one realization of a structure on each site
    - Links: corresponding to the connections between sites along various directions, and fields defined on the links rather than the sites
    - Tasks: performing computation over a set of sites
    - Ordered sets of sites
    - Maps: to move from one defined grid to another

- A paradigm:

    - A site represents a virtual processor with fields in its local memory
    - A task implies that all the virtual processors are computing in parallel

- A Canopy program:

    - A declaration section for grids, fields, sets, and maps
    - A control part calls tasks to be executed in parallel
    - The task routine to be executed on each processor

- Automatic data and task distribution and communication

- A tool that allows an SGI host to monitor a job and display information about job execution status, time limits, and disk and tape sets mounted

- A spooler which handles the assignment of resources so that a queue of jobs can be submitted to multi-user time sharing

Evaluation:

(By Glenn Kubena, Kenneth Liao, Larry Roberts of IBM)

Strengths:

- Well suited to applications whose domain can be represented by grids
- No new language or extensions for user to learn
- Relatively mature

Weaknesses:

- Currently limited to the ACPMAPS machine at Fermilab
- User must be cognizant of and observe certain programming restrictions to write programs successfully with Canopy
- Lack of debugging and performance tuning tools

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support NAS platforms)

Platforms: Sequent, Weitek-based 5GF ACPMAPS,
50GF i860-based ACPMAPS,
(Being ported to the Intel DELTA, iPSC/860,
and Paragon) (Not suitable for vector
machines and SIMD systems)

Operating System: POSIX-like system calls

Languages Supported: C, Fortran

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None (with restrictions)

Supplier: Computing Division
Fermi National Accelerator Laboratory
Batavia IL 60510

Contact: Mark Fischler
(708) 840-4339
mf@fnal.fnal.gov

See reference [72] [73] [1]

2.6 CPS (Cooperative Processes Software)

Functions:

- A library of routines callable from Fortran or C code
    - Supports computational task distributed across a heterogenous mix of UNIX machines.
    - Explicit message passing for task parallelism
    - "Call and queue" for implicit parallelism
    - Efficient bulk data transfer
    - Process classes for grouping the processes to execute the same program on same kind of computer
    - Remote procedure call
    - Synchronization
- A job manager for the construction and execution of parallel programs
    - Starting, stopping, and monitoring processes
    - Managing queues
    - Dynamic process allocation on a per class basis
- MIMD model (within each class, a single program is run)
- Support for host-node, client-server, and input-processing-output topologies as well as user customized topologies
- XOPER: an X-window based computer operator programs which supports mount requests, messages, etc to operations.
- CPS_XPSMON mulitple process performance monitoring tool (a terminal base version is also available)
- CPS_PSMON: a terminal based performance monitoring tool
- JMDB: a distribute processing debugging tool

Evaluation:

(By B. Traversat at NASA Ames Research Center (Previously at Superconducting Super Collider (SSC)))

- Unreliable when more than 30 workstations are used in a network

Useful to NAS:	No at present time Maybe in the future (User experience needed, Does not support NAS platforms)

Platforms: SGI IRIX, IBM RS6000, DEC (VMS VAX), Sun, HP, and MIPS workstations, and ACPR3000 boards in use at Fermilab;

Interconnection networks supported include Ethernet, VME-based bus communications, and internal buses on multiple-processor workstations.

Operating System: UNIX, provided by each platform (VMS Vaxes are also currently supported)

Languages Supported: Fortran, C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None (with restrictions)

Supplier: Computing Division, Fermi National Accelerator Laboratory Batavia IL 60510

Contact: Kevin Q. Sullivan
(708) 840-8782
kevins@baja.fnal.gov
or
cps_req@fndp1.fnal.gov

See reference [74] [75] [76] [77] [78]

2.7 PARMACS

Functions:

- A message-passing programming interface for both shared-memory parallel computers and distributed-memory parallel computers
    - Macros for process management, message-passing, and synchronization
    - Macros for process/processor mapping (torus, graph, and embedded tree for global communication)
    - Macros for dynamic torus remapping and switch dimensions between 3D and 2D
- Libraries for linear algebra and for grid-based applications
- Performance analysis tools
    - Visualization of process states and communication (post-mortem)

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support all NAS platforms)

Platforms: Intel iPSC/860, Meiko CS-Tools, nCUBE 2, Parsytec GC, CRAY Y-MP, A network of DEC, IBM RS/6000, SGI, and SUN workstations

Operating System: None

Languages Supported: Fortran77, PARMACS 6.0: Fortran77, C

Languages Used in Implementation: C

Graphic User Interface: X11R3 (or later) for the performance analysis tools

Cost: DM 2,000 on a workstation
DM 10,000 on CRAY

Supplier: PALLAS GmbH
Hermuelheimer Strasse 10
5040 Bruehl
Germany

GMD
Postfach 1316
5205 St. Augustin
Germany

Contact:

Distribution, general information:
Karl Solchenbach
+49-2232-1896.0
karls@pallas-gmbh.de

Development of programming interface:
Rolf Hempel
+49-2241-14.2575/2757
Rolf.Hempel@gmd.de

See reference [79] [80] [81] [82]

2.8 PICL (A Portable Instrumented Communication Library on Intel)

Functions:

- Library routines for writing parallel programs on message-passing computers
    - Process management
    - Message passing
    - Synchronization
    - Global operations
- Trace generation for performance monitoring (visualized by ParaGraph).

Evaluation:

(Compiled by Diane Rover and Joan Francioni of Michigan State University)

- Mostly used to generate a trace for visualization using ParaGraph[83]

Useful to NAS: Maybe, in conjunction with ParaGraph
(User experience needed)

Platforms: iPSC/2, iPSC/860, Delta, Paragon
Ncube/3200, Ncube 2, Cogent, mpsim

Operating System: Provided by each platform

Languages Supported: Fortran77, C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None
available from netlib@ornl.gov

Supplier: Oak Ridge National Laboratory
Oak Ridge, TN

Contact: Pat Worley
(615) 483-8111
worley@msr.epm.ornl.gov

See reference [84]

2.9 APPL (Application Portable Parallel Library)

Functions:

- A subroutine-based library of communication primitives
- Support for shared and distributed memory MIMD machines, and networks of workstations

Evaluation:

(By Kyung Ahn, Scott Townsend, and Suresh Khandelwal of NASA Lewis Research Center)

Applications:

- MHOST: A finite element program for nonlinear analysis of aerospace propulsion system structures
- MSTAGE: A multistage viscous turbomachinery program
- PARC3D: A 3-dimensional fluid dynamics code calculating the thermodynamic properties of a fluid flow

Strengths:

- Simple to understand, easy to use
- The code portable to different platforms
- Easy to install the system

Weaknesses:

- Lack of global operations on groups
- Different definition of synchronous/asynchronous send/receive operations with that supplied by vendors

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Similar tools support more NAS platforms)

Platforms: iPSC/860, Delta, Alliant FX/80, Hypercluster
(a NASA LeRC multi-architecture test bed),
SGI, Sun Sparc, IBM RS6000 workstations.

Operating System: Provided by each platform

Languages Supported: Fortran, C

Languages Used in Implementation: C and Fortran

Graphic User Interface: None

Cost: None (with permission)

Supplier: NASA Lewis Research Center

Contact: Angela Quealy
(216) 826-6642
fsang@kira.lerc.nasa.gov

See reference [85]

2.10 PARTI (Parallel Automated Runtime Toolkit at ICASE)

Functions:

- Runtime preprocessing procedures:
  - Coordinate interprocessor data movement.
  - Manage the storage of and access to copies of off-processor data.
  - Upport a shared name space.
  - Couple runtime data and workload partitioners to user programs.
- Accessible directly by programmers

Useful to NAS: Maybe
(User experience needed)

Platforms: iPSC/860, Delta, NCUBE, CM5,
a network of workstations
(Versions of Parti are built on top of Intel,
CM-5 message passing calls, PVM, and Express)

Operating System: Provided by each platform

Languages Supported: C, Fortran

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None
ftp : hyena.cs.umd.edu
in pub/parti_distribution and
block_parti_distribution

Supplier: University of Maryland

Contact: Raja Das
(301) 405-2693
raja@cs.umd.edu

See reference [86] [87] [88] [89]

2.11 CM (Communications Manager)

Functions:

- Library routines for job-level parallel execution on a network of heterogeneous machines

- Facilities for specifying the applications to be executed, their input and output arguments, and any explicit precedence instructions (Interactive applications cannot be included unless the IO can be redirected using files.)

- Communication Services:

    - TCP/IP based application to application connection and inter-process communication library for heterogeneous platforms
    - File transfer in program-to-program space (To transfer files the nodes need not be sharing file systems via NFS or ftp or uucp.)

- Directory Services:

    - TCP/IP based directory services for groupwork
    - Support for dynamic definition, registration besides lookup of networked resources
    - Support for interaction between applications using symbolic references rather than network addresses
    - Transparent client migration when a server migrates from one host to another

- Application Management System:

    - TCP/IP and Unix based network application invocation utility
    - Automatic transport of input files and output files to and from the client site to the application site
    - Security enforcement using the Directory Services in that only registered users may execute and only from registered machines

- Task Management System:

    - Analysis of dependencies between jobs based on a user-defined task description file
    - Generation of a maximally concurrent activation data flow chart
    - Synchronization between jobs if there is input/output dependency

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Job-level parallelism only)

Platforms: Sun Sparcs, DEC2000 ,DEC3000, DEC5000, SGI workstations

Operating System: SunOS, DEC/Ultrix, SGI/IRIX OS

Languages Supported: C

Languages Used in Implementation: C, TCP/IP, UNIX/OS (mostly POSIX compliant)

Graphic User Interface: None

Cost: None

Supplier: Concurrent Engineering Research Center
West Virginia University
Morgantown, WV 26505

Contact: Raman Kannan
kannan@cerc.wvu.wvnet.edu

See reference [90] [91] [92] [93]

2.12 SPPL (The Stuttgart Parallel Processing Library)

Functions:

- A message-passing library for a heterogeneous distributed memory system
- Support for abstract data types for messages
  - Simple data types
  - Arrays and records
  - Arbitrary pointer structures

Useful to NAS: No
(User experience needed,
Does not support Fortran,
Does not support NAS platforms)

Platforms: IBM RISC System/6000
Sun, DEC workstations

Operating System: AIX 3.2, SUN OS 4.1, ULTRIX 4.2

Languages Supported: C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: $100
$35 for educational and research institutions

Supplier: Institute for Parallel and Distributed
High Performance Systems (IPVR)
University of Stuttgart
Breitwiesenstr. 20-22
W-7000 Stuttgart 80
Germany

Contact: Prof. Andreas Reuter
(+49) 711 7816 449
Andreas.Reuter@informatik.uni-stuttgart.de

See reference [94]

2.13 LMPS (The Livermore Message Passing System)

Functions:

- A library of routines that implements an efficient message passing system on the BBN TC2000
- Support for synchronous and asynchronous (blocking and non-blocking) message passing, and selective reception of messages based on type and source
- Integrated with the PCP/PFP environment (See entry for PCP/PFP)

Useful to NAS: No
(User experience needed,
Does not support NAS platforms)

Platform: BBN TC2000

Operating System: UNIX

Languages Supported: Fortran77, C

Languages Used in Implementation: C, PCP

Graphic User Interface: None

Cost: None

Supplier: Lawrence Livermore National Laboratory L-560
P.O.Box 808 Livermore CA 94550

Contact: Tammy Welcome
(510) 422-4994
tsw@mpci.llnl.gov

See reference [95]

2.14 MTASK (The MultiTASKing package)

Functions:

- A multitasking library for forking of tasks from within code that is already executed in parallel or code that is recursive in nature
- Semaphores for mutual exclusion

Useful to NAS: No
(User experience needed,
Does not support Fortran,
Does not support NAS platforms)

Platforms: Alliant/FX, Alliant/FX2800

Operating System: Concentrix (Alliant Unix)

Languages Supported: C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: CSRD
University of Illinois

Contact: Brian Bliss
(217) 244-5569
bliss@csrd.uiuc.edu

2.15 GENMP (GENeric MultiProcessor)

Functions:

- A run time library for MIMD computer architectures
- Dynamic load balance and interprocessor communication
- Designed for particle methods and for uniform mesh methods that apply computational effort non-uniformly over the mesh.

Useful to NAS: No
(User experience needed,
Does not support all NAS platforms,
For particle methods only)

Platforms: iPSC/860, Cray Y-MP, Sparcstation

Operating System: Supported by each plotform

Languages Supported: Fortran77

Languages Used in Implementation: Fortran 77

Graphic User Interface: None

Cost: None
anonymous ftp cs.ucsd.edu
in pub/baden/genmp

Supplier: University of California at San Diego

Contact: Prof. Scott B. Baden
(619) 534-8861
sbaden@ucsd.edu

Scott Kohn
(619) 534-5913
skohn@ucsd.edu

See reference [96]

3. Debugging and Performance Tuning Tools

Twenty three tools described in this part are for debugging parallel programs and tuning their performance. Section 3.1 presents three environments integrating both types of tools. Section 3.2 describes the tools that are for debugging only. Section 3.3 lists the tools that are designed for performance tuning. As a result of visualizing program behavior, however, many tools in the last section can also assist debugging.

The order of presentation in each section is based on their usefulness to NAS (platforms, and languages supported), their relative maturity, and the amount of support from the suppliers. For tools with the same rating, they are listed in alphabetic order.

3.1 Integrated Systems

3.1.1 PRISM

Functions:

- Integrated graphical debugger, performance analysis tool, and data visualizer
- Automatic and consistent update for displays of debugging data, performance data, and source code
- Command alias
- System resource control (e.g. attach/detach sequencers, boot)
- Dynamic linking of programs
- Source-level debugger:
  - Breakpoints
  - Single step
  - Watch points for events
  - Expression evaluation
  - Trace
  - Stack, memory, and register examination
- Performance analysis:
  - Procedure level and statement level performance statistics for a specified resource or subsystem
  - Advices to assist isolating performance bottleneck
- Visualization:
  - Graphical display for data values or ranges
  - Performance statistics for resources

Evaluation:

(By Al Globus at NASA Ames Research Center)

- The best feature is is that it incorporates regular field visualization techniques to examine large arrays quickly.

Useful to NAS: Yes

Platforms: CM2, CM200, CM5

Operating System: CMOS, CMOST

Languages Supported: C, Fortran (TMC version)

Languages Used in Implementation: C

Graphic User Interface: X Motif

Cost: Bundled with system

Supplier: Thinking Machine Co.
245 First St.
Cambridge, MA 02142-1264

Contact: (617) 234-4000
(617) 876-1111
customer-support@think.com

See reference [97]

3.1.2 MPPE (MasPar Programming Environment)

Functions:

- Integrated graphical debugger, performance profiler, and visualizer, with client-server architecture for remote debugging

- Static analysis of parallel programs and graphical display of run-time profile information

- Automatic and consistent update for displays of stack, source code, and data after each skip, step, and continue during debugging

- Source-level debugging of parallel C and Fortran 90 code:

  - Step, skip, continue
  - Conditional breakpoints
  - Data inspectors
  - Evaluation of parallel C and Fortran 90 expressions
  - Graphical display of variable history
  - Animation of step, skip, continue (automatic repetition of commands)
  - Debugging optimized code

- Performance analysis:

  - Statement level and routine level profiling
  - Compiler-generated information relating to performance
  - Graphical display of profile histograms with source code
  - Summary pages for statement and routine level profiles
  - Profiling of optimized code

- Visualizer:

  - Processor array state
  - 2D visualization of variables, expressions

Useful to NAS: Maybe
(User experience needed
if NAS provides MP-1, MP-2)

Platforms: User interface on DecStation and SPARC
The program to be debugged on MP-1 or MP-2

Operating System: DecStation- Ultrix 4.2, SPARC- SunOS 4.1.1

Languages Supported: C, Fortran77, MasPar Fortran, MPL

Languages Used in Implementation: Smalltalk and C++

Graphic User Interface: DecStation- Motif, SPARC- OpenWindows

Cost:	One concurrent use license free with MasPar system. Additional licenses $2500 per concurrent use.
Supplier:	MasPar Computer Corporation 749 N. Mary Avenue Sunnyvale, CA 94086
Contact:	Helen Asher (408) 736-3300

See reference [98]

3.1.3 PARASPHERE

Functions:

- An integrated environment with OSF/Motif-like graphical interface

- High Performance Fortran with interface to C and Fortran 77

- DECmpp Programming Language (DPL), a C-like programming language

- System level commands for accessing the DECmpp 12000 Data Parallel Unit (DPU)

- An interactive parallel source-level debugger

    - Expression evaluation (DPF or DPL syntax)
    - Conditional breakpoints
    - State log and replay

- A profiler for analyzing program runtime statistics

    - Hierarchical and static profiling

- A call graph browser for viewing call relationships between program functions, files, and directories in graphic form

- A cross-referencer for determining where names are declared, defined, or referenced in a program

- DECmpp VAST-2 that translates Fortran 77 source code to DECmpp High Performance Fortran

    - Data dependency analysis
    - Safe loop transformation
    - Splitting of common blocks and separate scales from arrays
    - User directives and switches for interactive control of transformation
    - Examination of EQUIVALENCE statements to detect hidden recursion
    - Subroutine and function inlining

Useful to NAS: Maybe (User experience needed if NAS provides DECmpp)

Platforms: DECmpp 12000/Sx, 12000

Operating System: ULTRIX V4.2A

Languages Supported: C, Fortran, DPL, DECmpp High Performance Fortran

Languages Used in Implementation: C and C++

Graphic User Interface: Motif interface XIPD in late 1993

Cost: Bundled with system

Supplier: Digital Equipment Corporation
146 Main Street, MLO1-3/B11
Maynard, Massachusetts 01754

Contact: Mike Fishbein
fishbein@rdvax.enet.dec.com

See reference [99]

3.2 Debuggers

3.2.1 TOTALVIEW

Functions:

- Multi-process debugger
- Less-intrusive (It does not require any special libraries to be linked and allows code and conditional breakpoints to run as compiled code.)
- Ability to patch with compiled code
- Expression evaluation facility
- Evaluated breakpoints
- Multi-process breakpoints
- Integration with GIST for event logging (see BBN Performance Tools)
- Graphical interface for program control and breakpoint setting
- Ability to attach to running processes
- Ability to attach to processes created by fork
- Ability to debug a program running on a remote workstation
- Ability to communicate with the target system through shared memory, TCP/IP, or a serial line
- Ability to download code to an embedded system from a host for cross development purposes

Useful to NAS: Maybe
(User experience needed
if NAS provides CRAY MPP)

Platforms: BBN GP1000 and TC2000
Tadpole TP885v, FASP, Motorola 680x0, 88100
Sun SPARC, AT&T DSP32C
(In process of porting to CRAY MPP)

Operating System: Implementations available for SunOS, pSOS,
Lynx Realtime O/S, BBN nX, and hardware
with no operating system

Languages Supported: C, C++, Fortran

Languages Used in Implementation: C++

Graphic User Interface: X11R4

Cost: Embedded system solution typically $16K for the license plus the cost of porting to the given system

Pricing for the Sparc version announced in the future

Supplier: BBN Systems and Technologies

Contact: David Rich
(617) 873-2634
drich@bbn.com

See reference [100]

3.2.2 UDB (KSR symbolic debugger)

Functions:

- Breakpoints
- Examining and displaying data
- Examining the stack
- Specifying and examining source files
- Alias and user-defined commands
- Editing the command line
- Window status and control

Evaluation:

(By Donna Bergmark of Cornell Theory Center)

- Functional and helpful

Useful to NAS: Maybe
(User experience needed
if NAS provides KSR1)

Platform: KSR1

Operating System: KSR OS

Languages Supported: Fortran, C

Languages Used in Implementation: C

Graphic User Interface: X11R5

Cost: Bundled with system

Supplier: Kendall Square Research
170 Tracer Lane
Waltham, MA 02154

Contact: Steve Breit
(800) 669-1577
sbreit@ksr.com

See reference [101]

3.2.3 IPD

Functions:

- Source-level parallel debugger
  - Breakpoints
  - Watchpoints
  - Data display and modification
  - Source listing
  - Register display
  - Disassembler
  - Stack traceback
  - Debugger environment variables and command aliases
  - Run-time instrumentation of programs for profiling and event tracing (No special compile or link options necessary)
- Extensions to support distributed-memory parallel programs
  - Message queue display
  - All commands applicable for one or multiple processes
  - Control for separating debugger I/O from application I/O
- Menu-driven graphical interface (release 1.1)
- Integration with the performance analysis tools (release 1.1)
- Debug session logging facility

Useful to NAS: Maybe
(User experience needed)

Platforms: iPSC/860, planned for Paragon

Operating System: NX on iPSC/860, Paragon OSF/1 on Paragon

Languages Supported C, Fortran77

Languages Used in Implementation: C and C++

Graphic User Interface: Motif interface XIPD in late 1993

Cost: Bundled with system

Supplier: Intel Supercomputer Systems Division

Contact: Intel SSD Support
1-800-421-2823
support@ssd.intel.com

See reference [102]

3.2.4 XAB

Functions:

- Preprocessors and libraries for instrumenting PVM programs
- A monitoring process for collecting trace information as the program executes
- Graphic display of events and PVM calls
- A script for converting xab tracefiles to PICL tracefiles for use with Paragraph

Useful to NAS:	Maybe (User experience needed)
Platforms:	Unix based system where PVM runs
Operating System:	UNIX
Languages Supported:	C, Fortran77
Languages Used in Implementation:	C, CPP, m4, awk
Graphic User Interface:	X11R4, Athena widgets
Cost:	None
Supplier:	School of Computer Science and the Pittsburgh Supercomputer Center Carnegie Mellon University
Contact:	Adam Beguelin (412) 268-7866 adamb@cs.cmu.edu

See reference [103]

3.2.5 XPDB

Functions:

- A graphic debugger for programs using SPPL (see SPPL entry)
  - Examining message passing
  - Viewing hierarchical dataflow graph
  - Invoking sequential source code debuggers
- Facilities to print the data in selected messages
- Automatic selection of the appropriate print format depending on the abstract data type of the message

Useful to NAS: No
(User experience needed,
Does not support Fortran,
Does not support NAS platforms,
Limited to programs using SPPL)

Platforms: IBM RISC System/6000
Sun, DEC workstations

Operating System: AIX 3.2, SUN OS 4.1, ULTRIX 4.2

Languages Supported: C

Languages Used in Implementation: C

Graphic User Interface: X Window System X11R4 (or higher)

Cost: $200
$65 for educational and research institutions

Supplier: Institute for Parallel and Distributed
High Performance Systems (IPVR)
University of Stuttgart
Breitwiesenstr. 20-22
W-7000 Stuttgart 80
Germany

Contact: Prof. Andreas Reuter
(+49) 711 7816 449
Andreas.Reuter@informatik.uni-stuttgart.de

See reference [94]

3.2.6 EXECDIFF

Functions:

- Library routines for specifying the data objects to be monitored during execution
- Value Monitoring for the specified data objects in two versions of a program
- Specification of a tolerance for the difference in floating point numbers
- Value comparison for the data objects generated by executing the two versions to assist debugging

Useful to NAS: No
(User experience needed,
Does not support NAS platforms,
Limited to debugging programs
evolved from a correct version)

Platforms: Alliant Computers

Operating System: Unix (Berkeley or System V)

Languages Supported: C, Fortran

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: Center for Supercomputing Research & Development
University of Illinois

Contact: Brian Bliss
(217) 244-5569
bliss@csrd.uiuc.edu

3.3 Performance Tools

3.3.1 ATEXPERT

Functions:

- Performance monitoring and visualization
- Instrumentation of parallel regions indicated by the Autotasker and the serial sections of code (with 10%-20% overhead)
    - Program
    - Subroutines
    - Parallel regions
    - Parallel loops
- Graphical displays of performance data
    - Time spent in program segments
    - Number of processors on which a code segment is running
- Prediction for speedups on a dedicated system from data collected from a single run on a nondedicated system

Evaluation

(By Robert J. Bergeron of NASA Ames Research Center)

Strengths:

- Easy to use and provides flexible operation.
- Provides a detailed insight into parallel execution on Cray architectures.
- Text identifies many specific regions and causes of poor parallel performance.
- Visual display allows user to form their own judgements.

Weaknesses:

- Requires a strong understanding of Cray parallel processing and online help is insufficient.
- Basis for text judgements of poor parallel performance is not available (would be available on a true "expert" system).
- Emphasizes predictive capability whereas its main value is providing insight.

Useful to NAS:	Yes
Platforms:	CRAY Y-MP, X-MP EA, X-MP, CRAY-2 Display on SGI and Sun
Operating System:	UNICOS 6.0 FMP of the CF77 compiler release 4.03 or above

Languages Supported:	Fortran, C
Languages Used in Implementation:	C
Graphic User Interface:	X Windows (ASCII available)
Cost:	Bundled with system
Supplier:	Cray Research Inc. 2360 Pilot Knob Rd. Mendota Heights, MN 55120
Contact:	(612) 681-5907

See reference [104]

3.3.2 INTEL PERFORMANCE ANALYSIS TOOLS (release 1.1)

Functions:

- Based on ParaGraph
- Motif-based menu-driven graphical interface
- Animation of the execution of parallel applications derived from trace information gathered during program execution
- Graphical summaries and statistical analysis of overall program behavior
- Pause/resume, single-step, slow down, or restart an animation from any point.
    - Processor usage
    - Frequency, volume and overall pattern of inter-processor communications
    - Critical Path displays
    - Task displays relating processors with the part of the executing parallel code

Useful to NAS: Maybe
(User experience needed)

Platforms: iPSC/860, Paragon

Operating System: NX on iPSC/860, Paragon OSF/1 on Paragon

Languages Supported: C, Fortran77

Languages Used in Implementation: C and C++

Graphic User Interface: Motif interface XIPD in late 1993

Cost: Bundled with system

Supplier: Intel Supercomputer Systems Division

Contact: Intel SSD Support
1-800-421-2823
support@ssd.intel.com

See reference [105]

3.3.3 PARAGRAPH

Functions:

- Trace-based performance visualization of message-passing parallel programs (trace data generated by PICL, see entry for PICL)

- Dynamic, graphical depiction of processor utilization, communication traffic, load balance, and other aspects of program behavior and performance

Evaluation:

(By Diane Rover and Joan Francioni of Michigan State University)[106]

Applications:

- Teaching parallel computing and programming to novices pre-college students
- Teaching parallel computing and programming to advanced graduate students
- SLALOM: solves a radiosity problem in which the walls of a room are decomposed into patches. Computation time typically is dominated by the solution of a symmetric matrix using Gaussian elimination and back substitution techniques (on nCUBE-2, 128 nodes)

Strengths:

- Widely used
- Achieved considerable success when used to introduce parallel computing and programming to novices.
- Provided a common introduction and foundation for similar tools, and students could readily gain hands-on experience with it in self-paced laboratory exercises.
- PICL and Paragraph provided valuable assistance in studying and optimizing the SLALOM program.

Weaknesses:

- Considerable effort is required to selectively trace large programs or view only parts of a large trace file.
- Considerable effort is required to selectively reduce the trace data generated for runs with many processors.
- Movement through the trace file during ParaGraph simulation is constrained.
- There is no corresponding view of the source program.
- Invoking user-/application-specific views requires creating separate executables of ParaGraph.
- Comparison of data from multiple trace files requires running multiple instances of ParaGraph.

Suggestions based on experience:

- Prototyping views using the commercially-available and general-purpose tools AVS and MatLab
- Integrate the tested views into ParaGraph.

Useful to NAS: Maybe
(User experience needed)

Platforms: Unix workstation with X Windows

Operating System: UNIX

Languages Supported: C, Fortran77

Languages Used in Implementation: C

Graphic User Interface: X Windows (Xlib, no toolkit)

Cost: None

Supplier: Oak Ridge National Laboratory
and University of Illinois

Contact: Michael Heath
(217) 333-6268
heath@ncsa.uiuc.edu

See reference [107]

3.3.4 KSR PERFORMANCE TOOLS

Functions:

- Prof for displaying profile data produced by the monitor subroutine
  - Percentage of time spent in a subroutine
  - Number of times called
  - Number of milliseconds per call
  - A summary of profile (for multiple profile files)
- Gprof for displaying call graph profile data
- A performance monitoring library for per-thread performance data
  - Performance data from the event monitor, hardware registers, and the kernel (user-time, wall-clock-time cache hits/misses, thread migration, etc.)
  - Functions for accessing performance data
  - Functions for timing

Evaluations:

(By Donna Bergmark at Cornell Theory Center)

Strengths:

- The timer is good.

Weaknesses:

- Does not measure elapsed time across parallel jobs.

Useful to NAS: Maybe
(User experience needed
if NAS supports KSR1)

Platform: KSR1

Operating System: KSR OS

Languages Supported: Fortran77, C

Languages Used in Implementation: C

Graphic User Interface: None

Cost: Bundled with system

Supplier: Kendall Square Research
2102 Business Center Dr.
Waltham, MA 02154-1379

Contact: Steve Breit
(800) 669-1577
sbreit@ksr.com

See reference [101]

3.3.5 BBN PERFORMANCE TOOLS

Functions:

- GIST: An event logging and display tool
    - Events logging from within the user application
        - Kernel events (i.e. page swaps)
        - User defined events through subroutine calls
    - Graphic displays for the events and/or states
- ProfView: a statistical profiler
    - An extension of the standard Unix prof/gprof utilities
    - Profiling data collection at the subroutine, source line or instruction level
    - Data display along with a graph of time spent in each area
    - Multi-threaded program profiling and very light-weight profiling which produces only histograms

Useful to NAS: Maybe
(User experience needed
if NAS provides KSR1)

Platforms: BBN "Butterfly" series
KSR1
(GIST being ported to Sun Sparc)

Operating System: BBN nX, SunOS, KSR OS

Languages Supported: C, C++, Fortran

Languages Used in Implementation: C, C++

Graphic User Interface: X11R4
X11R5 for KSR1

Cost: $16,000 for GIST + ProfView on the TC2000 / GP1000. On KSR-1, the cost is bundled with the machine. Pricing for Sun and HP will be released.

Supplier:	BBN Systems and Technologies and Kendall Square Research
Contact:	David Rich (617) 873-2634 drich@bbn.com Steve Breit (800) 669-1577 sbreit@ksr.com

See reference [100]

3.3.6 AIMS (The Ames InstruMentation System)

Functions:

- A source code instrumentor which automatically inserts event recorders into program source code before compilation
- A run-time performance monitoring library which collects performance data
- A visualization tool-set which reconstructs program execution based on the data collected
- Being incorporated into the run-time environments of various parallel testbeds to evaluate their impact on user productivity

Evaluation:

(By Diane Rover and Joan Francioni of Michigan State University)[106]

Strengths:

- Compared with ParaGraph, AIMS offers greater control over the simulation replay, and a more flexible user interface

Weaknesses:

- More complicated than ParaGraph

Useful to NAS: Maybe
(User experience needed)

Platforms: Monitors the execution of applications
on the iPSC/860 and iPSC/Delta
The graphical interface on SunSparc and SGI

Operating System: NX

Languages Supported: Fortran, C

Languages Used in Implementation: C

Graphic User Interface: X11R5 and Motif

Cost: None

Supplier: NASA Ames Research Center

Contact: Jerry Yan
(415) 604-4381
jerry@ptolemy.arc.nasa.gov

See reference [108]

3.3.7 PABLO PERFORMANCE ANALYSIS ENVIRONMENT

Functions:

- A Motif-based interface for the specification of source code instrumentation points (both trace and count data)
- A C parser that can generate instrumented application source code
- A performance data trace capture library for single processor Unix systems and for the Intel iPSC/2 and iPSC/860 hypercubes
- A self-documenting data metaformat and associated tools that can be used to describe and process diverse types of data
- A graphical performance analysis environment

Useful to NAS:	Maybe (User experience needed)
Platforms:	Trace generation on Intel iPSC/2 and iPSC/860, Sun, (working on CM5) Visualization on SPARC2-GX
Operating System:	SunOS 4.1.2 for visualization
Languages Supported:	C, (working on Fortran)
Languages Used in Implementation:	C (GNU g++/gcc 2.3.1 or ATT Cfront version 3.0.1) perl 3.0
Graphic User Interface:	X11R5 (patch level 19), Motif release 1.2.1
Cost:	None (license required for commercial use) ftp bugle.cs.uiuc.edu (128.174.237.148)
Supplier:	University of Illinois at Urbana Champaign
Contact:	Daniel A. Reed pablo@bugle.cs.uiuc.edu

See reference [109]

3.3.8 IPS-2

Functions:

- Performance monitoring, analysis, and visualization
- Critical path analysis and visualization
- Support for the analysis of multiple applications and multiple runs of the same application in a single measurement session
- Support for dynamic on-the-fly user selection of what performance data to collect with decision support to assist users with the selection and presentation of performance data

Useful to NAS:	Maybe (User experience needed)
Platforms:	Y-MP, Sequent Symmetry, Sun (SunOS 4.1, Solaris 2.0), DECstation
Operating System:	UNIX
Languages Supported:	Fortran, C
Languages Used in Implementation:	C
Graphic User Interface:	X11
Cost:	$300 for source No charge to universities
Supplier:	Univ. of Wisconsin
Contact:	Barton P. Miller (608) 263-3378 bart@@cs.wisc.edu

See reference [110] [111] [112]

3.3.9 FALCON

Functions:

- A tool for on-line monitoring and visualization of programs using a parallel cthreads library on shared-memory machines
- A view specification language:
    - Specifying sensors
    - Predefined collection of probes, sensors, and views
- Interactive program instrumentation:
    - Software sensors for generating trace data synchronously with the program execution
    - Software probes for generating trace data only in response to an asynchronous request by the user or the monitoring system
- Trace Visualization:
    - Animated graphical displays of the program run-time performance and behavior (generated with the POLKA program animation system, see entry for POLKA)
    - Built-in graphical views
    - User-defined views

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support Fortran)

Platforms: KSR-1, Sequent Symmetry, GP-1000 BBN Butterfly, Silicon Graphics multiprocessor, SPARC

Operating System: SUN OS, Mach1000

Languages Supported: C

Languages Used in Implementation: C and C++

Graphic User Interface: X11R4, Open windows

Cost: None

Supplier: College of Computing
Georgia Institute of Technology

Contact:

Weiming Gu
(404) 894-3982
weiming@cc.gatech.edu

Karsten Schwan
(404) 894-2589
schwan@cc.gatech.edu

See reference [113]

3.3.10 VOYEUR

Functions:

- Library routines for trace generation and inspection
- Compiler option for parallel program profiling
- Run time statistics for performance analysis:
    - Amount of code running in parallel and serial
    - Performance prediction for a program on other Convex systems

Useful to NAS: No
(User experience needed,
Does not support NAS parallel platforms)

Platforms: All Convex Series of Supercomputers

Operating System: CONVEXOS

Languages Supported: Fortran, C

Languages Used in Implementation: Fortran

Graphic User Interface: None

Cost: None

Supplier: Convex Computer Corp.
European Headquarters
Randalls Research Park
Randalls Way
Leatherhead, Surrey
KT22 7TS
United Kingdom

Contact: Ronald W Gray
(44) 372-386696
gray@convex.com

3.3.11 UPSHOT

Functions:

- Analysis of execution trace produced by parallel programs
- Graphical tool for visualizing parallel program behavior
    - Individual events and process states on parallel time lines
    - Dynamic histogramming of state durations
    - Primitive proportional-time animation
    - Summary and detail information
    - Zooming in and out and scrolling back and forth

Useful to NAS: Maybe
(User experience needed)

Platforms: Sun, SGI, RS-6000, DEC

Operating System: Supported by the platforms

Languages Supported: C, Fortran, Aurora Parallel Prolog, PCN

Languages Used in Implementation: C, with Athena widgets

Graphic User Interface: X11R4 or later

Cost: None
ftp info.mcs.anl.gov
directory pub/upshot,
files upshot.tar.Z and alog.tar.Z

Supplier: Argonne National Laboratory

Contact: Rusty Lusk
(708) 252-7852
lusk@mcs.anl.gov

See reference [114]

3.3.12 MARITXU

Functions:

- A set of tools for visualizing massive amounts of trace data
- A trace-file format converter that converts an existing trace to the format acceptable by Maritxu
- Animation of dynamic evolution through time
  - Usage of processor resources (CPU load, queues, memory, link communication)
  - Current communication
  - Topology of the network/subnetwork
  - Information discrimination based on importance
- Highly scalable visualization for shared-memory and message-passing multiprocessors, and for distributed computing (by applying recent psychology discoveries in human perception to data presentation)
- A Set-up Manager for the user to set preferences, define icons, map data, determine topology, define statistics, and for defining the interface between the monitoring tool and Maritxu

Useful to NAS: Maybe
(User experience needed)

Platforms: Mips RS3260, MIPS 2030, SGI Indigo, SGI 4D/340S

Operating System: UNIX

Languages Supported: Supported by Transputers and CM2

Languages Used in Implementation: C

Graphic User Interface: X11R4, Motif 1.1

Cost: TBD

Supplier: Computer Systems Engineering
Dept. of Electronics
University of York
Heslington - York YO1 5DD
England

Contact: Eugenio Zabala
+ 44 904 432381
e-mail: ez@ohm.york.ac.uk

See reference [115] [116] [117]

3.3.13 GPMS (General Parallel Monitoring System)

Functions:

- Trace generation, filtering and transportation
- Visualization with a customized version of ParaGraph
  (see entry ParaGraph)

Useful to NAS: No
(User experience needed,
Does not support NAS platforms)

Platforms: Transputer networks
Transputer+i860 networks

Operating System: Trollius

Languages Supported: C, Fortran77

Languages Used in Implementation: C

Graphic User Interface: X11R4

Cost: None

Supplier: Trollius
Ohio supercomputer center,
OH, USA

ARCHIPEL
PAE des glaisins
74940 Annecy le vieux
+33 5064 0666

Contact: gdburns@tbag.osc.edu

Bernard Tourancheau
Bernard.Tourancheau@lip.ens-lyon.fr

4. Parallelization Tools

Nine tools presented in this part are to assist in converting a sequential program to a parallel program. Three different approaches to the conversion process are used. Section 4.1 lists the automatic conversion tools with user directives accepted for batch processing. Section 4.2 presents the interactive conversion tools which interact with users during a conversion process. The tools in the above sections use compiler analysis and transformation techniques to locate/create parallelism in a sequential code. The tools described in Section 4.3 provide means for the user to specify parallelism in a program; the tools do not attempt to discover parallelism.

4.1 Automatic Compilation

4.1.1 CRAY/fpp

Function:

- Automatic DO-loop parallelization
- Code transformation to take advantage of CRAY architecture

Evaluation:

(By Doug Pase and Katherine Fletcher through use)

- User interface is batch oriented.
- Uses static program analysis only.

Useful to NAS: Yes

Platforms: CRAY machines

Operating System: UNICOS, COS

Language Supported: Fortran

Graphic User Interface: None

Cost: Bundled with system

Supplier: Cray Research

See reference [118] [119]

4.1.2 KAP/CRAY

Function:

- Automatic DO-loop parallelization
- Code transformation to take advantage of CRAY architecture

Evaluation:

(By Doug Pase and Katherine Fletcher through use)

- User interface is batch oriented.
- Uses static program analysis only.

Useful to NAS:	No (Provides similar functions as provided by fpp)
Platforms:	Y-MP, X-MP, Sun, Vax
Operating System:	UNICOS, COS, UNIX, Ultrix
Languages Supported:	Fortran
Graphic User Interface:	None
Cost:	First copy $7,500/yr Add'l copy $3,750/yr Site license $15,000/yr
Supplier:	Kuck and Associates
Contact:	Davida Bluhm (217) 356-2288

See reference [119]

4.2 Interactive Parallelization Tools

4.2.1 FORGE 90

Functions:

- A tool set for interactively parallelizing Fortran loops
- Analysis tools:
  - Intra-procedural and inter-procedural dependency analysis
  - Flow analysis
- Tools for viewing the results provided by the analysis tools:
  - Reference tracing of variables and constants (use-def and def-use chains)
  - Exposing COMMON and EQUIVELENCE aliasing and COMMON block inconsistencies
  - Control and data flow from a global perspective
  - In/out data dependencies between routines, basic code blocks, or arbitrary blocks
- Parallelization of do-loops with/without subroutine calls
- Parallel/vector directives insertion for shared-memory parallel machines
- Support for interactive user data decomposition (HPF)
- Generation of Fortran programs with message passing for distributed-memory machines
- Conversion of a Fortran 77 program to a Fortran 90 program
- Automatic program instrumentation and parallelization guided by run time statistics
- A database of static information of a program

Evaluation:

(By Donna Bergmark of Cornell Theory Center)

Strengths:

- Very useful for traipsing through old dusty deck Fortran codes
- Its timing profile down to the loop level is very helpful
- Reliable and very responsive vendor support
- Highly recommended

Weaknesses:

- Does not parallelize certain loops that can be parallelized by PAT

(By Doreen Cheng of NASA Ames Research Center)

Applications:

NAS Benchmarks on dedicated Y-MP/8.
Forge version 7.01, 1990

Strengths:

- The interactive nature provides convenient access to the tools and to information about the program.
- Useful in helping understanding a program developed by others
- Useful in achieving speedup without rewriting a program
- The most robust system in its kind.
- The user response time is reasonably fast.
- Responsive vendor support.

Weaknesses:

- Requires familiarity of compiler analysis terminology.

Suggestions:

- Code generation must take advantage of target architectures.
- Guidance to dependence elimination, code transformation and parallelization will be very helpful.

Useful to NAS:	Yes
Platforms:	Generates code for: CRAY Y-MP, Intel iPSC/860, Delta, Paragon, CM2, CM5, Clustered Workstations using PVM or Express. Runs on all Suns, IBM RS/6000, SGI, HP workstations, IBM Power/4

Operating System: Unix any flavor

Languages Supported: Fortran 77 and Fortran 90, HPF

Languages Used in Implementation: C

Graphic User Interface: Sunview, X-window (native, motif), Openlook

Cost: Start at
$1850 for source code browser
$4250 for shared memory parallelizer
$6250 for distributed memory parallelizer
$7600 for batch HPF parallelizer

Supplier: Applied Parallel Research, Inc.
550 Main Street, Suite I.
Placerville, CA 95667

Contact: John Levesque
(916) 621-1600
levesque@a.psc.edu

Jim Dillon
(916) 621-1600
jed@netcom.com

See reference [120]

4.2.2 PARASCOPE

Functions:

- An environment for development of parallel scientific programs written in a shared-memory parallel dialect of Fortran 77.

- Program analysis tools:

    - Data dependence analysis
    - Control dependence analysis
    - Control flow analysis
    - Global value numbering
    - Static single assignment
    - Basic Fortran static semantic analysis
    - Interprocedural analysis

- Program transformation tools:

    - Reordering transformations:

        - Loop distribution, interchange, skewing, fusion, reversal
        - Statement interchange

    - Dependence breaking transformations:

        - Privatization, scalar expansion, array renaming
        - Loop splitting, peeling, alignment

    - Memory optimizing transformations:

        - Strip mining, loop unrolling, scalar replacement, unroll and jam

    - Others:

        - Sequential to parallel loops, statement addition and deletion, loop bounds adjustment

- The ParaScope Editor (PED):

    - Text editing
    - Template-based structure editing of Fortran modules
    - Access to the results generated by the program analysis tools
    - User-guided program transformation to exploit and reveal parallelism

- A debugging system:

  - Automatic instrumentation of parallel Fortran programs for shared-memory multiprocessors to detect data races
  - Support for parallelism in the form of nested parallel loops
  - Interprocedural analysis to guide placement of instrumentation to minimize run-time overhead
  - Data race report in the form of a pair of references highlighted in the source code

- The prototype Fortran D compiler:

  - Translates Fortran programs annotated with data layout and distribution directives to Fortran program with message passing for distributed memory multiprocessors.

Evaluation:

(By Donna Bergmark of Cornell Theory Center)

Strengths:

- Useful in finding data dependencies in parallel loops

Weaknesses:

- Difficult for users to figure out what to do if there are dependencies
- Does not generate parallel code for our platforms

Useful to NAS: Maybe for tool development

Platforms: ParaScope is supported on the Sun4 and RS6000 platforms.
The Fortran D compiler generates code for the IPSC/860.
PED generates code for the Sequent and the Cray.

Operating System: Sun OS 4.x on Sun4’s , AIX 3.2 on RS6000’s

Languages Supported: Fortran 77 + extensions for expressing parallel loops

Languages Used in Implementation: C, C++, Yacc, Lex

Graphic User Interface: X11R4

Cost: $150 for site license

Supplier:	Rice University CITI/SDC P.O. Box 1892 Houston, TX 77251-1892 (713) 527-6077
Contact:	Send email messages with subjects "send license" and "send catalog" to softlib@cs.rice.edu to receive information on how to get ParaScope

See reference [121] [122] [123]

4.2.3 PAT (Parallelization AssistanT)

Functions:

- A tool set for interactively parallelizing loops in Fortran programs

- Program analysis:

  - Intra-/inter-procedural control dependency analysis
  - Intra-/inter-procedural data dependency analysis
  - Flow analysis
  - Loop analysis

- Program transformation:

  - Alignment, replication, expression substitution
  - Code/declarations generation for parallel loops
  - Explicit synchronization insertion (events and locks)

- Interactive parallelization:

  - Display of program structure (subroutines and loops)
  - Loop parallelizability classification
  - Sequential to parallel loop conversion

- Program instrumentation:

  - Timing calls insertion around selected loops
  - Trace generation
  - Display for statistics of loop counts and runtimes

Evaluation:

(Donna Bergmark of Cornell Theory Center)

Strengths:

- Useful in finding data dependencies in parallel loops
- Generate parallel code from serial Fortran code.
- X-Window based user interface is convenient.
- Easy to learn
- Recommended

Weaknesses:

- Requires understanding of dependency analysis.
- GUI not robust enough

Useful to NAS: Maybe
(User experience needed)

Platforms:	Pat runs on SUN, IBM, DEC, HP, SGI workstations Generates code for KSR-1, IBM 3090, Sequent, Cray Y-MP
Operating System:	UNIX
Languages Supported:	Full Fortran 77, with KSR directives and IBM Parallel Fortran.
Languages Used in Implementation:	C and C++
Graphic User Interface:	X11R4
Cost:	Free for non-commercial use includes reuseable data structure class components. anonymous ftp from: ftp.cc.gatech.edu in directory pub/pat
Supplier:	Georgia Institute of Technology
Contact:	Bill Appelbe (404) 894-6187 bill@cc.gatech.edu

See reference [124] [125] [126]

4.2.4 TINY

Functions:

- A research/educational tool for experimenting with array data dependence tests and reordering transformations

- Dependency analysis:

    - Induction variable recognition
    - Choice of: Omega test, Power test, Lambda test, Banerjee's inequalities
    - With Omega test:

        - Reduction dependences
        - Array kill analysis
        - Analysis of when assertions can eliminate a dependence

- Program transformations:

    - Array & scalar expansion and privatization
    - Scalar forward substitution
    - Storage classes
    - Loop interchange/skewing/distribution/fusion

- A graphic user interface for browsing the results of analysis and transformations

- A framework for reordering transformations which can be used to optimize code fragments for parallelism and/or locality

Useful to NAS: Maybe for tool development

Platforms: Unix platforms

Operating System: UNIX

Languages Supported: Tiny (A toy language developed by Michael Wolfe of OGI)

Languages Used in Implementation: C

Graphic User Interface: Xterm & curses interface

Cost: None
anonymous ftp from ftp.cs.umd.edu
in pub/omega

Supplier: Department of Computer Science
University of Maryland

Contact: omega@cs.umd.edu

See reference [127] [128] [129]

4.3 User-Specifying Parallelism

4.3.1 SCHEDULE

Function:

- A large-grain dataflow language for expressing dependencies between code blocks written in Fortran
- A graphic user interface for specification and visualization
- Performance tools:
  - Program profiling
  - Execution monitoring
  - Memory access visualization
  - Program execution visualization
  - Critical path determination
- Task scheduling
- Dynamic load balancing

Evaluation:

(By Donna Bergmark at Cornell National Supercomputer Facility)

- Obsolete and being decommissioned
- Received very little use

Useful to NAS: No
(User experience needed,
Does not support NAS platforms)

Platform: CRAY-2

Operating System: UNIX

Languages Supported: Fortran

Languages Used in Implementation: C

Graphic User Interface: X11R4

Cost: None

Supplier: University of Tennessee

Contact: Jack Dongarra
(615) 974-8295
dongarra@cs.utk.edu

See reference [130 131]

4.3.2 PYRROS

Functions:

- A task graph language for specifying the operations of tasks and the data dependencies between them. (The tasks are written in C.)
- A graphical interface for displaying task dependencies
- A scheduler for mapping tasks with arbitrary precedence to processors and ordering their execution
- A graphical interface for displaying the schedule
- A code generator for producing parallel C codes with communication primitives based on the schedule

Useful to NAS: No
(User experience needed,
Does not support Fortran,
Does not support NAS platforms)

Platforms: SUN Generates code for nCUBE-II
(working on Intel iPSC/860)

Operating System: UNIX or others that support C

Languages Supported: C

Languages Used in Implementation: C

Graphic User Interface: X window is optional, graph displayer needs proprietary AT&T DAG system

Cost: None

Supplier: Department of Computer Science
Rutgers University

Contact: Tao Yang
(908)-932-0050
tyang@cs.rutgers.edu

Apostolos Gerasoulis
(908)-932-2725
gerasoulis@cs.rutgers.edu

See reference [132]

4.3.3 ENTERPRISE

Functions:

- A graphic user interface for specifying large-grain (functional) parallelism in C programs (in terms of "assets" such as departments, services, workers, and divisions)
- A preprocessor to generate (from the graphical specification) a C program with parallel functions and message passing between them
- Support for non-blocking function calls until the caller accesses a result that needs to be returned by the callee (futures)
- Automatic source control
- Limited debugging/performance monitoring
- Animation of a program execution at message-passing level

Useful to NAS:	No (User experience needed, Does not support Fortran, Does not support NAS platforms)
Platform:	Sun
Operating System:	UNIX
Languages Supported:	C
Languages Used in Implementation:	Smalltalk, C, C++
Graphic User Interface:	Smalltalk running under X windows
Cost:	In alpha test stage
Supplier:	Department of Computing Science University of Alberta Canada
Contact:	Jonathan Schaeffer (403) 492-3851 jonathan@cs.ualberta.ca

See reference [133 134]

5. Others

Ten tools are included in this part. Section 5.1 presents the meta-tools (tools for building tools). Section 5.2 lists the tools to make it more effective to compute on a network of workstations. Section 5.3 describes the tools for grid partitioning, task scheduling and load balancing.

5.1 Tools for Building Tools

5.1.1 SAGE

Functions:

- A programming language transformation toolkit used to design restructuring source-to-source compilers and instrumentation packages

- A complete parser for Fortran 77, Fortran 90, C (ansi and k&r), C++ Att 2.0 and 3.0

- A common internal representation, a library of C functions, and a C++ class library to access and restructure the internal representation and to generate the output code

- A comment based annotation language to allow transformation access to user assertions

Useful to NAS: Maybe for tool development

Platforms: Sun, HP, NeXT, SGI and DEC workstations (Porting on CM-5, Paragon, KSR, n-Cube, Mieko, Tera, Cray)

Operating System: UNIX

Languages Supported: Fortran77, Fortran 90, C, C++

Languages Used in Implementation: C and C++

Graphic User Interface: None

Cost: None (gnu rules apply)

Supplier: Indiana University

Contact: Dennis Gannon
(812) 335-5184
gannon@cs.indiana.edu

See reference [135]

5.1.2 IMPROV (Integrated Manipulation of PROgram Visualization)

Functions:

- A meta-tool for fast prototyping of complex visualizations of parallel software behavior (a next generation of the PARADISE, see next entry)
    - Support for a formal and complete generalization of the program visualization procedure, which divides the task into fundamental, independent modules for specifying the relationships among basic concepts (Events, Behavior, and Graphics).
    - Facilities for users to specify details in each module in terms of objects called "entities", and to define the visualization by indicating relationships among the entities in different modules
    - A textual language for specifying Events and Behavior entities
    - A graphical editor for specifying Graphics (It can also be specified by text.)
    - Rules for specifying entity relationships
    - Hierarchical entity construction
    - Framework for relating entities from different modules
- Scalability to massively parallel software through various abstract reduction operations.
- Wide range of graphical manipulations based on fundamental graphical objects, such as points, lines, squares, circles, polygons, etc.
- Syntax-directed editing
- Support for traces of arbitrary formats, event formats integrated with event entity specification

Useful to NAS:	No at present time Maybe in the future (User experience needed, Does not support NAS platforms)
Platforms:	Sun3 and Sun4/Sparc workstations
Operating System:	UNIX
Languages Supported:	No specific language requirements or support Trace formats defined by the user with the Event entities, ascii and binary

Languages Used in Implementation: C

Graphic User Interface: X11R4 (or later)

Cost: None (with license agreement)

Supplier: Department of Electrical & Computer Engineering
University of Iowa

Contact: James Arthur Kohl
(319) 335-6432
kohl@hitchcock.eng.uiowa.edu

Thomas L. Casavant
(319) 335-5953
tomc@hitchcock.eng.uiowa.edu

See reference [136 137 138]

5.1.3 PARADISE (PARallel Animated DebuggIng and Simulation Environment)

Functions:

- A meta-tool for designing program visualization tools for debugging and performance tuning of parallel software

- Facilities for users to designs an abstract visual model of parallel program behavior

    - Defining "visual objects" using programmed modules that describe their functionality and interfaces.
    - Connecting visual objects using a graphical interface to form the overall structure of the visual model.

- Tools for replaying parallel program execution traces

- Tools for simulating program behavior and animating it via the visual model

- Variable animation and simulation speeds

- Point-and-click to identify internal states of visual objects and their interconnections (e.g. details of messages, time stamps, and the actual data being transferred)

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support NAS platforms)

Platforms: Sun3 and Sun4/Sparc workstations

Operating System: UNIX

Languages Supported: No specific language requirements. Trace formats are defined by the user to allow arbitrary traces to be used (ascii only).

Languages Used in Implementation: C

Graphic User Interface: Sunview or X11R4 (or later)

Cost: None (with license agreement)

Supplier: Department of Electrical & Computer Engineering
University of Iowa

Contact: James Arthur Kohl
(319) 335-6432
kohl@hitchcock.eng.uiowa.edu

Thomas L. Casavant
(319) 335-5953
tomc@hitchcock.eng.uiowa.edu

See reference [139] [140] [141]

5.1.4 POLKA

Functions:

- A graphics library for building animations of parallel programs and their executions
  - Primitives for creating smooth, color, continuous program visualizations and animations
  - Primitives for explicitly building animations with concurrent actions, thereby helping to illustrate parallelism
  - Focused on ease of learning and use by graphics non-experts

Useful to NAS: Maybe for tool development

Platforms: Unix workstations

Operating System: UNIX

Languages Supported: C++

Languages Used in Implementation: C++

Graphic User Interface: X Windows and Motif or OLIT

Cost: None

Supplier: Graphics, Viz., and Usability Center
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Contact: John Stasko
(404) 853-9386
stasko@cc.gatech.edu

See reference [142]

5.2 Tools for Parallel Computing on A Network

5.2.1 DQS (Distributed Queuing System)

Functions:

- A distributed queuing system for a network of heterogeneous computers
- Support for single and multi-node loosely-coupled batch processing
- Facilities for users to request resource that meet some minimum performance or architecture specifications
- Facilities for configuring, modifying, deleting, and querying the queues
- Facilities for submitting, monitoring, querying, and terminating jobs on a single computer or a cluster of computers
- Load balancing for jobs submitted to *group* queues
- Optional keyboard/mouse activity monitoring (queue suspended if active)
- Parallel jobs using PVM supported in dedicated mode

Useful to NAS:	Maybe (User experience needed)
Platforms:	SUN, IBM, SGI, DEC, HP workstations
Operating System:	SunOS 4.1.x, AIX 3.2.x, IRIX 4.0.x, OSF/1,MACH, HPUX, ULTRIX
Languages Supported:	All languages supported by platforms
Languages Used in Implementation:	C
Graphic User Interface:	X11
Cost:	None ftp at ftp.scri.fsu.edu pub/DQS//DQS.REV_2.0.tar.Z
Supplier:	Supercomputing Research Institute Florida State University
Contact:	Thomas P. Green (904) 644-0190 green@scri.fsu.edu

See reference [143] [144]

5.2.2 FUNNEL

Functions:

- A run time support for SIMD program execution on a network of heterogeneous workstations
- Manages the data stream for multiple instances of the application
- Fault tolerant in cases of machine crash, no swap space, disk full, or ethernet down
- Large grain parallelism achieved by providing each instance of a single-processor application with an environment mimicking the original

Useful to NAS:	Maybe (User experience needed)
Platforms:	SGI, DECstations
Operating System:	UNIX
Languages Supported:	Interfaces to executables, not code
Languages Used in Implementation:	C
Graphic User Interface:	None
Cost:	None Demo available via anonymous ftp at zebra.desy.de
Supplier:	ZEUS Collaboration Deutsches Elektron-Synchrotron (DESY) Hamburg, Germany
Contact:	Burkhard Burow 49-40-8998-3053 burow@vxdesy.cern.ch

5.2.3 DJM (The Distributed Job Manager)

Functions:

- A job control system that manages the execution of user applications on Connection Machines
- Facilities to submit, monitor, and terminate jobs
- Facilities to query the status of jobs
- Commands for specifying the number of processors and the amount of memory and disk space required
- Scheduling jobs to achieve balanced the load on CM partitions
- Scheduling jobs to achieve balanced the load on CM front-end
- Facilities for switching the direction of input/output streams of a job between screen/keyboard and a file

Useful to NAS: Maybe
(Under evaluation at NAS
User experience needed)

Platforms: CM2, CM5, SUN

Operating System: Provided by the platforms

Languages Supported: Supported by the platforms

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None
ftp ec.msc.edu
/pub/LIGHTNING/djm_0.9.11.Z

Supplier: Minnesota Supercomputer Canter, Inc.

Contact: Alan Klietz
(612) 626-1737
alan@msc.edu

See reference[145]

5.2.4 CONDOR

Functions:

- A distributed batch system for recovering idle cycles of a network of workstations
- Job-level parallelism only
- Monitoring the activities of participating workstations
- Scheduling jobs to idle workstations
- Suspending/migrating foreign jobs when the owner of a workstation starts using it
- Fault-tolerance (with restrictions):
    - Checkpointing
    - Process migration
- Protection of owner's files from executing foreign jobs
- Direct NFS access and remote system calls for executing programs to access remote files
- Redirection of file I/O to the submitting machine

Useful to NAS: Maybe
(User experience needed,
Mechanisms to handle parallel
applications needed)

Platforms: Suns, SGI, Snakes(hp), DecStations, 6K

Operating System: UNIX BSD 4.2 and 4.3

Languages Supported: All languages supported by the platforms

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None
ftp ftp.cs.wisc.edu

Supplier: Computer Science Department
University of Wisconsin, Madison

Contact: Miron Livny
(608) 262-0856
miron@cs.wisc.edu

See reference[146 147]

5.2.5 MNFS (Modified NFS)

Functions:

- A modified NFS system for supporting shared-memory programs on a network of workstations:
  - Supports shared memory for mapped files
  - Support for user-managed consistency of mapped pages
  - An additional address space with 32-byte pages for efficiency
  - Additional extensions to support efficient data sharing and program synchronization in the high-latency environments found in computer networks.
- Better performance than NFS
- Provided as a mod-loaded module, i.e. it runs as part of the kernel but is dynamically loaded after the OS is booted. No OS recompilation or rebuilding is required. (A modified NFS daemon is also provided, as is an automounter that mounfs MNFS file systems.)

Useful to NAS: No at present time
Maybe in the future
(User experience needed,
Does not support all NAS platforms)

Platforms: Sun workstations (including MPs)
(386BSD, IRIX in progress)

Operating System: SunOS 4.1.1, 4.1.2, 4.1.3

Languages Supported: C, Fortran77,
Any language that can open a file and do mmap()

Languages Used in Implementation: C

Graphic User Interface: None

Cost: None

Supplier: Supercomputing Research Center,
17100 Science Drive,
Bowie, MD. 20716

Contact: Ron Minnich
(301)-805-7451
rminnich@super.org

See reference [148 149 150]

5.3 Partitioners and Schedulers

5.3.1 TOP/DOMDEC (DOMain DEComposition)

Functions:

- A tool for mesh partitioning in parallel processing
- Algorithms for initial automatic mesh partitioning
    - The Greedy Algorithm
    - The RCM algorithm
    - A recursive RCM algorithm
    - Slicing (Principal Inertia) algorithms (Ix, Iy, Iz)
    - Recursive versions of the Slicing algorithms (RIx, RIy, RIz)
    - Frontal Algorithm for 1D Topology partitions
    - Recursive Graph Bisection
    - Recursive Spectral Bisection
- Support for optimizing the items listed below over the initial partitioning
    - Size of the interface
    - Frontwidth of the subdomain (for direct frontal solvers)
    - A product of the above two items
    - Node-wise load balancing
    - Element-wise load balancing
    - Edge-wise load balancing
- Algorithm used in optimizing the component of the mesh that is neighboring the interface
    - Tabu Search
    - Simulated Annealing
    - Stochastic Evolution
- Support for decision making in selecting the best mesh partition for a given problem and a given multiprocessor
- Support for evaluation of load balancing, network traffic and communication costs
- Generation of parallel data structures needed for local computations and for message passing
- Object oriented high-speed graphics for user interface

Useful to NAS:	Maybe (User experience needed)
Platforms:	Runs on SGI, IBM R6000 with the GL card Generates partition for CRAY Y-MP, KSR, iPSC-860, and CM-2

Operating System: UNIX

Languages Supported: Fortran, C++

Languages Used in Implementation: C++ and GL

Graphic User Interface: GL

Cost: $250 for source license

Supplier: PGSoft, 5212 Pinehurst Drive
Boulder, CO 80301

Contact: Charbel Farhat
Phone (303) 492-3992
charbel@alexandra.Colorado.EDU

See reference [151]

5.3.2 GANG SCHEDULER

Functions:

- Support for fair sharing of time and space on the BBN TC2000
- Allocation of processors and requests for benchmarking time
- Four priority queues

Useful to NAS: No
(User experience needed,
Does not support NAS platforms)

Platform: BBN TC2000

Operating System: UNIX

Languages Supported: C, Fortran, PCP, PFP, Uniform Systems, Zipcode

Languages Used in Implementation: C

Graphic User Interface: vt100 display

Cost: None

Supplier: Lawrence Livermore National Laboratory L-560
P.O.Box 808 Livermore CA 94550

Contact: Brent Gorda
(510) 294-4147
brent@igor.nersc.gov

See reference [152]

5.3.3 BLOBS

Functions:

- A visualization tool for interactive evaluation of alternative load balancing strategies through simulation
- Designed for computations involving particles only
- Facilities for users to control number of processors, partitioning method, and partitioning frequency
- Tools to estimate and display the problem partitioning, interprocessor communication costs, workload distribution across the processors, and overall parallel efficiency based on the user input and a sample trace of a execution

Useful to NAS: No
(User experience needed,
Does not support all NAS platforms,
For particle methods only)

Platforms: Sparcstation running openwindows

Operating System: UNIX

Languages Supported: Accepts a blobs-specific trace format.

Languages Used in Implementation: C

Graphic User Interface: Xview and OpenWindows libraries

Cost: None
anonymous ftp cs.ucsd.edu
in pub/baden/blobs

Supplier: University of California at San Diego

Contact: Prof. Scott B. Baden
(619) 534-8861
sbaden@ucsd.edu

Scott Kohn
(619) 534-5913
skohn@ucsd.edu

See reference [153]

5.3.4 PREP-P (PREProcessor for Poker)

Functions:

- Static partitioning, placing, routing and scheduling for function-parallel programs
- Simulation of parallel execution using Poker (Poker is no longer supported by University of Washington)

Useful to NAS:	No (User experience needed, Does not support Fortran, Does not support all NAS platforms)
Platform:	Sun3 and Sparc workstations
Operating System:	UNIX
Languages Supported:	XX (a simple abstraction of C)
Languages Used in Implementation:	C, assembly language
Graphic User Interface:	Required by Poker
Cost:	None
Supplier:	Parallel Computation Lab Department of Computer Science and Engineering University of California, San Diego
Contact:	Francine Berman (619)534-6195 berman@cs.ucsd.edu

See reference [154]

Acknowledgements

I would like to express my gratitude to all the people who submitted descriptions of their tools and/or shared with me their experience of using tools. I also would like to thank reviewers Tom Woodrow, Louis Lopez and Bernard Traversat for their comments.

References

1. Glenn Kubena, Kenneth Liao, and Larry Roberts, *White Paper on Massively Parallel Programming Languages,* IBM, Dec. 3, 1992.
2. Thomas Sterling, Paul Messina, Marina Chen, Frederica Darema, Geoffrey Fox, Michael Heath, Ken Kennedy, Robert Knighten, Reagon Moore, Sanjay Ranka, Joel Saltz, Lew Tucker, and paul Woodard, ‘‘System Software and Tools for High Performance Computing Environments,’’ *A Report on the Findings of the JPL Pasadena Workshop*, April, 1992.
3. ISO/IDE, ‘‘Information technology -- Programming languages -- Fortran,’’ *International Standard, Reference number ISO/IEC 1539 : 1991 (E)*, 1991.
4. High Performance Fortran Forum, *High Performance Fortran Language Specification*, Version 1.0 Draft, Jan. 25, 1993.
5. ANSI Technical Committee X3H5, *Parallel Processing Model for High Level Programming Languages*, June 1992.
6. ANSI Technical Committee X3H5, *Parallel Fortran Standard*, 1992.
7. ANSI Technical Committee X3H5, *Fortran Binding -- Data Model Section*, 1992.
8. T. Brandes, ‘‘Efficient Data Parallel Program without Explicit Message Passing for Distributed Memory Multiprocessors,’’ *GMD Technical Report, TR92-4*, 1992.
9. L. Ridgway Scott, ‘‘Pfortran: a parallel dialect of Fortran,’’ *Fortran Forum 11*, vol. No. 3, pp. 20-31, Sept.1992.
10. Harry Jordan, ‘‘The Force,’’ *ECE Tech. Report 87-1-1*, Jan. 1987.
11. Harry F. Jordan, Muhammad S. Benten, Norbert S. Arenstorf, and Aruna V. Ramanan, *Force User’s Manual*, March 1989.
12. Donna Reese, ‘‘Object-Oriented Fortran for Portable, Parallel Programs,’’ *The Third IEEE Symposium on Parallel and Distributed Processing*, pp. 608-615, December 1991.
13. B.Chapman, P.Mehrotra, and H.P.Zima, ‘‘Vienna FORTRAN - A Fortran Language Extension for Distributed Memory Multiprocessors,’’ in *Compilers and Runtime Software for Scalable Multiprocessors*, ed. J. Saltz and P. Mehrotra, Elsevier, Amsterdam, 1991.
14. P.Brezany, B.Chapman, and H.Zima, ‘‘Automatic Parallelization for GENESIS,’’ *Austrian Center for Parallel Computation, Technical Report ACPC/TR 92--16*, November 1992.
15. B.M.Chapman, P.Mehrotra, and H.Zima, ‘‘Programming in Vienna Fortran,’’ *Scientific Programming*, vol. Vol.1, No.1, 1992.
16. Seema Hiranandani, Ken Kennedy, and Chau-Wen Tseng, ‘‘Compiling Fortran D for MIMD Distributed-Memory Machines,’’ *Communications of the ACM*, vol. 35(8), pp. 66-80, August 1992.
17. Geoffrey Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Ulrich Kremer, Chau-Wen Tseng, and Min-You Wu, ‘‘Fortran D Language Specification,’’ *Dept. of Computer Science Technical Report TR90-141, Rice University*, December 1990.
18. I. Foster and K. M. Chandy, ‘‘Fortran M: A Language for Modular Parallel Programming,’’ *Preprint MCS-P327-0992, Argonne National Laboratory, Argonne, Ill*, 1992.
19. P. Newton and J. C. Browne, ‘‘The CODE 2.0 Parallel Programming Language,’’ *Proc. ACM International Conf. on Supercomputing*, July 1992..
20. A. Reuter, U. Geuder, M. Haerdtner, B. Woerner, and R. Zink, ‘‘The GRIDS Project,’’ *Technical Report, University of Stuttgart*, 1992.

21. Eugene Brooks, Brent Gorda, and Karen Warren, "The Parallel C Preprocessor," in *Scientific Programming*, vol. vol 1, Number 1, John Wiley & Sons, Inc., New York.

22. Brent Gorda, Karen Warren, and Eugene D. Brooks III, "Programming in PCP," *Technical Report, Lawrence Livermore National Laboratory, UCRL-MA-107029*, April 1991.

23. Brent Gorda, "Data Parallel Programming," *Spring Proceedings, 1992 Cray User Group*.

24. Eugene D. Brooks et al., "The 1992 MPCI Yearly Report: Harnessing the Killer Micros," *Lawrence Livermore National Laboratory, UCRL-ID-107022-92*, March 1991.

25. M. C. Rinard, D. S. Scales, and M. S. Lam, "Heterogeneous Parallel Programming in Jade," *Proceedings of Supercomputing '92*, pp. 245-256, Nov. 1992.

26. M. S. Lam and M. C. Rinard, "Coarse-Grain Parallel Programming in Jade," *Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 94-105, April, 1991.

27. M. C. Rinard and M. S. Lam, "Semantic Foundations of Jade," *Record of the Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 105-118, Jan., 1992.

28. L.V. Kale, "The Chare Kernel Parallel Programming Language and System," *in: Proc. of the International Conference on Parallel Processing*, Aug. 1990.

29. L.V. Kale, "A Tutorial Introduction to Charm," *Parallel Programming Laboratory Internal report*, 1992.

30. Matthew Rosing, Robert B. Schnabel, and Robert P. Weaver, "The DINO Parallel Programming Language," *Tech. Report CU-CS-457-90, CS Dept. Univ. of Colorado at Boulder*, April 1990.

31. Thomas M. Derby, Elizabeth Eskow, Richard Neves, Matthew Rosing, Robert B. Schnabel, and Robert P. Weaver, "DINO 1.0 User's Manual," *Tech Report CU-CS-501-90, CS Dept. Univ. of Colorado at Boulder*, April 1990.

32. Min-You Wu and Daniel D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1 No. 3, pp. 330-343, July 1990.

33. Francois Bodin, Peter Beckman, Dennis Gannon, Srinivas Narayana, and Shelby Yang, "Distributed pC++: Basic Ideas for an Object Parallel Language," *Proceedings of Supercomputing 91 (Albuquerque, Nov. 1991), IEEE Computer Sciety and ACM SIGARCH*, pp. 273-282.

34. A. S. Grimshaw, "Easy to Use Object-Oriented Parallel Programming with Mentat," *to appear in IEEE Computer*, May 1993.

35. A. S. Grimshaw, W. Timothy Strayer, and Padmini Narayan, "The Good News About Dynamic Object-Oriented Parallel Processing," *University of Virginia, Computer Science Report TR-92-41,*, 1992.

36. Rohit Chandra, Anoop Gupta, and John L. Hennesy, "Integrating Concurrency and Data Abstraction in the COOL Parallel Programming Language," *Technical Report CSL-TR-92-511, Computer Systems Lab, Stanford University,*, February 1992.

37. Rohit Chandra, Anoop Gupta, and John L. Hennesy, "Data Locality and Load Balancing in COOL," *To Appear in the Symposium on Principles and Practices of Parallel Programming (PPoPP)*, May 1993.

38. Gail E. Kaiser, Wenwey Hseush, Steven S. Popovich, and Shyhtsun F. Wu, "Multiple Concurrency Control Policies in an Object-Oriented Programming System," *2nd IEEE Symposium on Parallel and Distributed Processing, Dallas TX*, pp. 623-626,

December, 1990.

39. Steven S. Popovich, Shyhtsun F. Wu, and Gail E. Kaiser, "An Object-Based Approach to Implementing Distributed Concurrency Control," *11th International Conference on Distributed Computing Systems, Arlington TX*, pp. 65-72, May, 1991.
40. Wenwey Hseush, James C. Lee, and Gail E. Kaiser, "MeldC Threads: Supporting Large-Scale Dynamic Parallelism," *Technical Report CUCS-010-92, Columbia University*, March, 1992.
41. Thomas Braunl, "Structured SIMD Programming in Parallaxis," *Structured Programming Journal*, vol. 10/3, 1998.
42. Thomas Braunl, " Parallel Programming," in *An Introduction Textbook*, Prentice-Hall, Summer 1993.
43. Walter F. Tichy and Christian G. Herter, "Modula-2*: An Extension of Modula-2 for Highly Parallel, Portable Programs," *Technical Report KA-INF, No. 4/90*, Jan., 1990.
44. Michael Philippsen and Walter F. Tichy, "Modula-2* and its Compilation," *First International Conference of the Austrian Center for Parallel Computation*, pp. 169-183, Springer Verlag, September 30 - October 2, 1991.
45. Ian Foster and Stephen Taylor, in *Strand New Concepts in Parallel Programming*, Prentice Hall, 1989.
46. D. Cann, "Retire Fortran? A Debate Rekindled," *Communications of the ACM*, vol. Vol 35, Number 8, August, 1992.
47. J. T. Feo, D.C. Cann, and R. R. Oldehoeft, "A Report on the SISAL Language Project," *Journal of Parallel and Distributed Computing*, vol. Vol 12 No. 10, pp. 349-366, December 1990.
48. J. R. McGraw and et. al., "Sisal: Streams and iterations in a single-assignment language, Language Reference Manual, Version 1.2," *Lawrence Livermore National Laboratory Manual M-146 (Rev. 1)*, March 1985.
49. Lawrence Livermore National Laboratory, "Proceedings of the Second Annual Sisal Users Conference," *Lawrence Livermore National Laboratory Technical Report UCRL JC112593*, October 1992.
50. I. Foster, R. Olson, and S. Tuecke, "Productive parallel programming: The PCN approach," *Scientific Programming*, vol. 1(1), pp. 51-66, 1992.
51. K. M. Chandy and S. Taylor, *An Introduction to Parallel Programming,* Jones and Bartlett, 1991.
52. Alan H. Karp, "Some Experience with Network Linda," *The International Journal for High Speed Computing (to appear)*, 1993.
53. D. Gelernter, N. Carriero, S. Chandran, and S. Chang, "Parallel Programming in Linda," *Proceedings of the 1985 International Conference on Parallel Processing*, pp. 255-263, 1985.
54. Sudhir Ahuja, Nicholas Carriero, and David Gelernter, "Linda and Friends," *IEEE Computer*, pp. 26-34, Aug. 1986.
55. T. Bemmerl and A. Bode, "An Integrated Tool Environment for programming distributed memory multiprocessors," in *Distributed memory computing, Lecture Notes in Computer Science*, ed. A. Bode, vol. Vol. 487, pp. 130 - 142, Springer-Verlag, 1991.
56. "Manuals on: MMK, TOPSYS, DETOP, PATOP, VISTOP," *Technical Report, Technische Universit"at M"unchen*, 1991.
57. Binay Sugla, John Edmark, and Beth Robinson, "An Introduction to the CAPER Concurrent Application Programming Environment," *IEEE Conference on Parallel*

Processing, 1989.

58. Doug Kimeleman and Dror Zernik, "On-the-Fly Topological Sorting for Interactive Debugging and Live Visualization of Parallel Programs," *To appear in the Third ACM ONR Workshop on Parallel and Distributed Debugging*, may, 1993.
59. Leonid Gluhovsky and Dror Zernik, "ILGA - A Little Language For Processing Graphs," *Technical report No. 872. Electrical Engineering Faculty, Technion*.
60. Gregory R. Andrews and Ronald A. Olsson, *The SR Programming Language: Concurrency in Practice,* ISBN 0-8053-0088-0, Benjamin/Cummings Publishing Company, 1993.
61. G. Sutcliffe and J. Pinakis, "Prolog-D-Linda: An Embedding of Linda in SICStus Prolog," *Technical Report 91/7, Department of Computer Science, The University of Western Australia, Perth, Australia*.
62. H. El-Rewini and T. Lewis, "Scheduling and Performance Evaluation Tool for Parallel Computing," *Proceedings of 4th Annual Symposium on Parallel Processing, Fullerton, CA.*, p. 60, April 1990.
63. H. El-Rewini and T. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *Journal of Parallel and Distributed Computing*, vol. Vol 9, pp. 138-153, June 1990.
64. Parasoft Co., *Express C User's Guide, Version 3.0*, 1990.
65. J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, "A Users' Guide to PVM," *Technical Report No. ORNL/TM-11826, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831-6367*, July 1991.
66. V. S. Sunderam, "PVM : A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience*, vol. Vol. 2 No. 4, pp. 315--339, Dec. 1990.
67. J. J. Dongarra, G. A. Geist, R. Manchek, J. Plank, and V. Sunderam, "HeNCE: A User's Guide Version 1.2," *Technical Report, Computer Science Department, CS-92-157*, February 1992.
68. J. J. Dongarra, G. A. Geist, R. Manchek, K. Moore, R. Wade, and V. S. Sunderam, "HeNCE: Graphical Development Tools for Network-Based Concurrent Computers," *Proceedings of the Scalable High Performance Computing Conference IEEE Computer Society Press*, pp. 129-136, April 1992, Williamsburg.
69. Ralph Butler Ewing Lusk, "User's Guide to the p4 Parallel Programming System," *Technical Report, Argonne National Laboratory, Mathematics and Computer Science Division, ANL-92/17*, Oct. 1992.
70. Timothy G. Mattson, Craig C. Douglas, and Martin H. Schultz, "A Comparison of CPS, LINDA, P4, POSYBL, PVM, and TCGMSG: Two Node Communication Times," *Yale University Technical Report YALEU/DCS/TR-975*, 1993.
71. Robert J. Harrison, "Moving Beyond Message Passing: Experiments With A Distributed-Data Model," *Technical Report, Argonne National Laboratory*, 1991.
72. M. Fischler, G. Hockney, and P. Mackenzie, "Canopy 5.0 Manual," *Technical Report, Fermilab*.
73. M. Fischler, "The ACPMAPS System - A Detailed Overview," *Fermilab publication FERMILAB-TM-1780*.
74. Fausey, Rinald, Wolbers, Potter, Yeager, Ullfig, "CPS & CPS Batch Reference Guide," *Fermi Computing Division #GA0008*.
75. Fausey, Rinaldo, Wolbers, Potter, Yeager, Ullfig, "CPS User's Guide," *Fermi Computing Division #GA0009*.
76. Frank Rinaldo and Stephan Wolbers, "Loosely-Coupled Parallel Processing at Fermilab," *To be published in 'Computers in Physics'*, March-April 1993.

77. Matthew R. Fausey, ‘‘CPS and the Fermilab Farms,’’ *FERMILAB-Conf-92/163*, June 1992.

78. Tom Nash, ‘‘High Performance Parallel Local Memory Computing at Fermilab,’’ *Proceedings of WHP92 on Heterogeneous Processing, Beverly Hills, Calif*, March 23, 1992.

79. L. Bomans, R. Hempel, and D. Roose, ‘‘The Argonne/GMD macros in Fortran for portable parallel programming and theirimplementation on the Intel iPSC/2,’’ *Parallel Computing, North-Holland*, vol. Vol. 15, pp. 119-132, 1990.

80. R. Hempel, ‘‘The ANL/GMD Macros (PARMACS) in Fortran for Portable Parallel Programming using the Message Passing Programming Model,’’ *User's Guide and Reference Manual, Version 5.1*, Nov. 1991.

81. R. Hempel, H.-C. Hoppe, and A. Supalov, ‘‘PARMACS 6.0 Library Interface Specification,’’ *GMD internal report*, Dec. 1992.

82. R. Hempel and H. Ritzdorf, ‘‘The GMD Communications Library for Grid-oriented Problems,’’ *GMD Arbeitspapier No. 589*.

83. Diane T. Rover and Joan M. Francioni, ‘‘A Survey of PICL and Paragraph Users (1992),’’ *Technical Report, Department of Electrical Engineering, Michigan State University (TR-MSU-EE-SCSL-01193)*, Feb. 1993.

84. G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, ‘‘PICL, A Portable Instrumented Communication Library, C Reference Manual,’’ *ORNL/TM-11130*, July, 1990.

85. A. Quealy, G. L. Gole, and R. A. Blech, ‘‘Portable Programming on Parallel/Networked Computers Using the Application Portable Parallel Library (APPL),’’ *to be published as a NASA TM*, 1993.

86. R. Ponnusamy, R. Das, J. Saltz, and D. Mavriplis, ‘‘The Dybbuk Runtime System,’’ *Compcon, San Francisco*, February 1993.

87. C. Chase, K. Crowley, J. Saltz, and A. Reeves, ‘‘Parallelization of Irregularly Coupled Regular Meshes,’’ *Sixth International Conference on Supercomputing, Washington DC*, June 1992.

88. R. Das, D. J. Mavriplis, J. Saltz, S. Gupta, and R. Ponnusamy, ‘‘The Design and Implementation of a Parallel Unstructured Euler Solver Using Software Primitives (AIAA-92-0562),,’’ *Proceedings of the 30th Aerospace Sciences Meeting, Reno NV*, 1992.

89. A. Sussman, J. Saltz, R. Das, S. Gupta, D. Mavriplis, and R. Ponnusamy, ‘‘PARTI Primitives for Unstructured and Block Structured Problems,’’ *Computing Systems in Engineering (Proceedings of Noor's Flight Systems conference*, pp. 73-86, 1992, 3, 1.

90. R. Kannan, et. al., ‘‘Software Environment for Network Computing,’’ *Workshop on "Heterogeneous Network-Based Concurrent Computing" SCRI/Florida State University*, October 16-18, 1991 also appears in the newsletter for the IEEE Technical Committee on Operating Systems and Application Environments, Vol. 6, No. 1, 1992. Also available as a technical report (CERC-TR-RN-91-007) from CERC, WVU, Morgantown, WV 26505..

91. R. Kannan, C.L.Chen, Michael Packer, and Hawa Singh, ‘‘Directory Service for Group Work,’’ *CSCW 92 Tools & Technologies Workshop, Toronto Canada. Also available as a technical report (CERC-TR_RN-91-004) from CERC, WVU, Morgantown, WV 26505*.

92. Vangati R. Narender and R. Kannan, ‘‘Dynamic RPC for Extensibility,’’ *IEEE International Phoenix Conference on Computers and Communications-92, Phoenix, Arizona*, April 1 -3, 1992.

93. V. Jagannathan, K. J. Cleetus, R. Kannan, J. Toth, and V. Saks, "Application Message Interface," *IEEE International Phoenix Conference on Computers and Communications-92, April 1 -3, 1992, Phoenix, Arizona.*.

94. Roland Zink, "The Stuttgart Parallel Processing Library SPPL and the X Windows Parallel Debugger XPDB," *To be presented at the Seventh International Parallel Processing Symposium Parallel Systems Fair, Newport Beach CA*, April 1993.

95. Tammy Welcome, "Programming in LMPS," *Technical Report, Lawrence Livermore National Laboratory, UCRL-MA-107031*, March 1991.

96. Scott B. Baden and Scott Kohn, "The Reference Guide to GenMP -- The Generic Multiprocessor," *Technical Report, University of California, San Diego, Dept. of Computer Science and Engineering, CS92-243*, June, 1992.

97. Thinking Machine Co., *Prism User's Guide version 1.1*, Dec. 1991.

98. MasPar Computer, *MPPE User Guide*.

99. Digital Equipment Corporation, *PARASPHERE User's Guide*.

100. BBN Systems and Technologies, Inc., *Using the Xtra(TM) Programming Environment*.

101. Kendall Square Research, *KSR Manual*.

102. Intel Supercomputer Systems Division, *iPSC/2 and iPSC/860 Interactive Parallel Debugger Manual*, April 1991.

103. Adam Beguelin, "Xab: A Tool for Monitoring PVM Programs," *To appear HP'93, Newport CA, April 1993. Also available as CMU tech report CMU-CS-93-105*.

104. Cray Research Inc., *UNICOS Performance Utilities Reference Manual SR-2040 6.0*, pp. 199-234, 1991.

105. Intel Supercomputer Systems Division, *Performance Analysis Tools Manual*.

106. Diane Rover and Joan Francioni, "In process of compiling a report on the experience of using parallel programming tools," Michigan State University, 1993.

107. Michael T. Heath and Jennifer A. Etheridge,, "Visualizing the Performance of Parallel Programs," *IEEE Software*, vol. Vol. 8, No. 5, pp. 29-39, September 1991.

108. Charlies Fineman, Philip Hontalas, Sherry Listgarten, and Jerry Yen, *A User's Guide to AIMS, Version 1.1*, May, 1992.

109. Daniel A. Reed, Ruth A. Aydt, Tara M. Madhyastha, Roger J. Noe, Keith A. Shield, and Bradley W. Schwartz, "The Pablo Performance Analysis Environment," *Technical Report, University of Illinois*.

110. Barton P. Miller, Morgan Clark, Jeff Hollingsworth, Steven Kierstead, Sek-See Lim, and Timothy Torzewski, "IPS-2: The Second Generation of a Parallel Program Measurement System," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1 No. 2, April 1990.

111. Jeffrey K. Hollingsworth and bart@cs.wisc.edu, "Dynamic Control of Performance Monitoring on Large Scale Parallel Systems," Technical Report, Univ. of Wisconsin-Madison Comp Sci Dept.

112. R. Bruce Irvin and Barton P. Miller, "Multi-Application Support in a Parallel Program Performance Tool," *Univ. of Wisconsin-Madison Comp Sci Dept Tech Rep #1135*.

113. Weiming Gu and Karsten Schwan, "Falcon: A Monitoring and Visualization System for Parallel and Distributed Systems.," *Technical Report GIT-CC-93/11, Georgia Institute of Technology. January 1993. Draft.*.

114. Virginia Herrartc and Ewing Lusk, "Studying Parallel Programming Behavior with Upshot," *Argonne National Laboratory Technical Report ANL-91/15*.

115. Eugenio Zabala and Richard Taylor, "Maritxu: Generic visualisation of highly parallel processing," in *Programming Environments for Parallel Computing*, ed. N. Topham, R. Ibbett, and T. Bemmerl, pp. 171-180, Noth-Holland, 1992.

116. Eugenio Zabala and Richard Taylor, "Process and processor interaction: architecture independent visualisation schema," *Environments and Tools for Parallel Scientific Computing, 7-8 September, Saint Hilaire du Touvet, France*, 1992.

117. Eugenio Zabala and Richard Taylor, "Maritxu: Visualising the run-time behaviour of transputer networks," in *Parallel Computing: from theory to sound practice, Proceedings from EWPC'92, Barcelona, Spain*, ed. Wouter Joosen and Elie Milgrom, pp. 100-103, IOS Press, March 1992.

118. Cray Research, Inc., *CF77 Compiling System Volume 4: Parallel Processing Guide*.

119. Douglas M. Pase and Katherine E. Fletcher, "Automatic Parallelization: A Comparison of CRAY fpp and KAI KAP/CRAY," *NASA Ames NAS Technical Report, RND-90-010*, Nov. 1990.

120. Applied Parallel Research, *FORGE 90 Version 8.0 Baseline System User's Guide*, April 1992.

121. Alan Carle, Keith D. Cooper, Robert T. Hood, Ken Kennedy, Linda Torczon, and Scott K. Warren, "A Practical Environment for Scientific Programming," *IEEE Computer*, Nov. 1987.

122. C. David Callahan, Keith D. Cooper, Robert T. Hood, Ken Knennedy, and Linda Torczon, "Parascope: A Parallel Programming Environment," *International Journal of Supercomputer Applications*, vol. 2 No. 4, pp. 84-99, Winter, 1988.

123. K. Kennedy, K. S. McKinley, and C. Tseng, "Interactive Parallel Programming Using the ParaScope Editor," *TOPDS*, vol. 2 No. 3, pp. 329-341, July 1991.

124. Bill Appelbe and Kevin Smith, "Start/Pat: A Parallel-Programming Toolkit," *IEEE Software*, pp. 29-38, July 1989.

125. Kevin Smith, Bill Appelbe, and Kurt Stirewalt, "Incremental Dependence Analysis for Interactive Parallelization," *ICS*, pp. 330-341, June 1990.

126. Bill Appelbe, Kevin Smith, and Kurt Stirewalt, "PATCH -- A New Algorithm for Rapid Incremental Dependence Analysis," *ICS*, pp. 424-432, June 1991.

127. Michael Wolfe, "The Tiny Loop Restructuring Research Tool," *Proc of 1991 International Conference on Parallel Processing,*, pp. II-46 - II-53, 1991.

128. William Pugh, "The Omega test: a fast and practical integer programming algorithm for dependence analysis," *Communications of the ACM 8*, pp. 102--114, August 1992.

129. William Pugh and David Wonnacott, "Eliminating False Data Dependences using the Omega Test," *Tech. Report CS-TR-2993, Dept. of Computer Science, Univ. of Maryland, College Park.*

130. J. J. Dongarra and D. C. Sorensen, "SCHEDULE: Tools for Developing and Analyzing Parallel Fortran Program," *Tech. Memo 86, Argonne National Laboratory*, Nov. 1986.

131. J. J. Dongarra and D. C. Sorensen, "SCHEDULE Users Guide," *Argonne National Laboratory*, June 1987.

132. T. Yang and A. Gerasoulis, "PYRROS: Static scheduling and code generation for message passing multiprocessors," *Proc. of 6th ACM Inter. Conf. on Supercomputing, Washington D.C.*, pp. 428-437, July 1992.

133. Greg Lobe, Paul Lu, Stan Melax, Ian Parsons, Jnathan Schaeffer, Carol Smith, and Duane Szafron, "The Enterprise Model for Developing Distributed Applications,"

TR 92-20, Department of Computing Science, University of Alberta, 1992.

134. Duane Szafron, Jonathan Schaeffer, Pok Sze Wong, Enoch Chan, Paul Lu, and Carol Smith, "The Enterprise Distributed Programming Model," *Programming Environments for Parallel Computing*, pp. 67-76, Elsevier Science Publishers, 1992.

135. Dennis Gannin, Jenq Kuen Lee, Bruce Shei, Sekhar Sarukkai, Srinivas Narayana, Neelakantan Sundaresan, Daya Attapatu, and Francois Bodin, "Sigma II: A Tool Kit for Building Parallelizing Compilers and performance Analysis Systems," *Proceedings 1992, IFIP Edinburgh Workshop on Parallel Programming Environments. April, 1992 in Programming Environments for Parallel Computing, IFIP Transactions A-11, N. Topham, R. Ibbet, T. Bemmerl, eds., North-Holland Press*, pp. 17-36.

136. T. L. Casavant and J. A. Kohl, "The IMPROV Meta-Tool Design Methodology for Visualization of Parallel Programs," *Invited Paper, International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, January 1993.

137. J. A. Kohl and T. L. Casavant, "Methodologies for Rapid Prototyping of Tools for Visualizing the Performance of Parallel Systems," *Presentation at Workshop on Parallel Computer Systems: Software Tools, Santa Fe, New Mexico, October 1991.*

138. J. A. Kohl, "The Construction of Meta-Tools for Program Visualization of Parallel Software," Technical Report Number TR-ECE-920204, Department of ECE, University of Iowa, Iowa City, IA, 52242, February 1992.

139. J. A. Kohl and T. L. Casavant, "Use of PARADISE: A Meta-Tool for Visualizing Parallel Systems," *Proceedings of the Fifth International Parallel Processing Symposium (IPPS), Anaheim, California*, pp. 561-567, May 1991.

140. J. A. Kohl and T. L. Casavant,, "A Software Engineering, Visualization Methodology for Parallel Processing Systems," *Proceedings of the Sixteenth Annual International Computer Software & Applications Conference (COMPSAC), Chicago, Illinois*, pp. 51-56, September 1992.

141. T. L. Casavant, J. A. Kohl, and Y. E. Papelis, "Practical Use of Visualization for Parallel Systems," *Invited Keynote Address Text for 1992 European Workshop on Parallel Computers (EWPC), Barcelona, Spain*, March 23-24, 1992.

142. Stasko, John T. and Kraemer, Eileen, "A Methodology for Building Application-Specific Visualizations of Parallel Programs," *Technical Report Graphics, Visualization, and Usability Center, Georgia Institute of Technology, GIT-GVU-92-10, also to appear in Journal of Parallel and Distributed Computing, May 1993*, Jan. 1992 $K polka-report.

143. Thomas P. Green and Jeff Snyder, "DQS, A Distributed Queuing System," DQS Documents, March 1992.

144. Louis S. Revor, "DQS Users Guide," DQS Documents, Sept. 1992.

145. Alan Klietz, "The Distributed Job Manager," *User's Guide of DJM*, 1992.

146. Allan Bricker, Michael Litzkow, and Miron Livny, "Condor Technical Summary," Condor Documents, Sept. 1991.

147. Michael Litzkow and Miron Livny, "Supporting Checkpointing and Process Migration Outside the UNIX Kernel," *Usenix Winter Conference, San Francisco, CA*, 1992.

148. Ron Minnich and Dave Farber, "Reducing Host Load, Network Load, and Latency in a Distributed Shared Memory," *10th ICDCS, 1990.*

149. Ron Minnich, "Mether: A Memory System for Network Multiprocessors," Ph.D. Thesis, U. Penn., 1991.

150. Ron Minnich and Dan Pryor, ‘‘A Radiative Heat Transfer Simulation on a SPARC-Station Farm,’’ *HPDC-1, 1992*.

151. C. Farhat and M. Lesoinne, ‘‘Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics,’’ *International Journal for Numerical Methods in Engineering*, vol. Vol. 36, No. 5, pp. 745-764, 1993.

152. Brent C. Gorda and Eugene D. Brooks III, ‘‘Gang Scheduling a Parallel Machine,’’ Lawrence Livermore National Laboratory UCRL-JC-107020 Rev. 1.

153. Scott R. Kohn and Scott B. Baden, ‘‘Blobs: Visualization of Particle Methods on Multiprocessors,’’ CSE Technical Report Number CS92-241, May, 1992.

154. *Experience With an Automatic Solution to the Mapping Problem in The Characteristics of Parallel Programs,* MIT Press, 1987.

Appendix A: Comparison of Parallel Languages

The table on the next page compares the languages described in Part 1. It indicates whether there is a compiler/translator for the language, and whether there are debuggers and performance tuning tools associated. It also indicates if the language is useful to NAS at this time.

Parallel Languages					
Name	Compiler Preprocessor	Debugger	Performance Tool	Useful To NAS	Page Number
Adaptor	y	n	n	n	16
Caper	y	y	y	n,m	56
CC++	y	n	n	n,m	41
Charm	y	n	y	n,m	31
Code 2.0	y	n	y	n,m	24
Cool	y	n	y	n,m	39
Dino	y	n	n	n,m	32
Force	y	n	n	n,m	18
Fortran 90	y	n	n	std	10
Fortran D	y	y	n	m	21
Fortran M	y	n	n	n,m	23
Grids	y	n	n	n,m	26
HPF	n	n	n	std	12
HyperTool	y	n	y	n,m	33
Jade	y	n	n	n,m	29
Linda	y	y	y	m	52
MeldC	y	y	n	n	42
Mentat	y	n	n	n,m	37
Modula-2*	y	n	n	n	45
O-O Fortran	y	n	n	n,m	19
P-Languages	y	n	n	n,m	17
Parallax	n	n	y	n	62
Parallaxis	y	y	y	n	43
PC++	y	n	n	n,m	35
PCN	y	y	y	n,m	50
PCP/PFP	y	n	n	n	27
PDDP	y	n	n	n	28
P-D Linda	y	n	n	n	61
Sisal	y	y	n	n,m	48
SR	y	n	n	n	60
Strand88	y	y	y	n,m	46
Topsys	y	y	y	n,m	54
Vienna Fortran	y	n	n	n	20
Visage	y	n	y	n,m	58
X3H5	n	n	n	std	14

O-O: Object-Oriented
P-D: Prolog-D
m: may be (user experience needed and/or support NAS platforms)
n: no
n,m: no at present time, maybe in the future
std: proposed standard
y: yes

Appendix B: Comparison of Libraries

The table below compares the libraries described in Part 2. The entry "Machine" lists the type of machines which the library supports: shared-memory, message-passing, and networks of computers. The entries "Debugger" and "Performance Tools" indicate if there are debugging and performance tools associated with the library.

Libraries					
Name	Machine	Debugger	Performance Tool	Useful to NAS	Page Number
APPL	msg	n	n	n,m	79
Canopy	sh,msg,net	n	n	n,m	72
CM	net	n	n	n,m	82
CPS	net	y	y	n,m	74
Express	sh,msg,net	y	y	m	64
GenMP	sp	n	n	n	87
LMPS	msg	n	n	n	85
Mtask	sh	n	n	n	86
P4	sh,msg,net	n	y	m	69
Parmacs	sh,msg,net	n	y	n,m	76
Parti	msg,net	n	n	m	81
PICL	msg	n	y	m	78
PVM	net	y	y	m	67
SPPL	net	n	n	n	84
TCGMSG	sh,msg	n	n	n,m	70

m: may be (user experience needed)
msg: message passing
n: no
n,m: no at present time, maybe in the future
net: network of computers
sh: shared memory
sp: special purpose
y: yes

Appendix C:

Comparison of Debugging and Performance Tuning Tools

The table on the next page compares the main features of the tools described in Part 3. Many tools in this part provide visualization of execution trace for performance tuning. Visualizing program behavior also helps debugging, although debugging might not be the focus of the design of the tool. The "tr" in the "Debugger" entry below reflects this side effect. The entry "Trace Generation" indicates whether a tool generates trace and the method of trace generation if it does. The entry "Visualization" indicates the main features of the visualization.

Debuggers and Performance Tuning Tools					
Name	Debugger	Trace Generation	Visualization	Useful To NAS	Page Number
AIMS	tr	au,u	g,an,sm,p	m	111
Atexpert	n	au	g,st,sm	y	102
BBN-Perf	tr	au,u	g,st,p	m	109
ExecDiff	d	u	tx	n	101
Falcon	tr	u	g,an	n,m	114
GPMS	tr	u	g,an,sm,p	n	120
Intel-Perf	tr	u	g,an,sm,p	m	104
IPD	sl,tl	n	tx	m	98
IPS-2	tr	y	y	m	113
KSR-Perf	n	au,u	tx	m	107
Maritxu	tr	n	g,an	m	118
MPPE	sl	au,u	g,an,sm,p	m	91
Pablo	tr	u	g,an	m	112
ParaGraph	tr	u	g,an,sm,p	m	105
ParaSphere	sl	au	g,st,sm,p	m	93
Prism	sl	au	g,an,sm,p	y	89
TotalView	sl	u	tx	m	95
UDB	sl	n	n	m	97
Upshot	tr	n	g,an,sm,p	m	117
Voyeur	n	u	tx,p	n	116
XAB	tr	au	g,an	m	99
Xpdb	tl	n	n	n	100

an: animation-based visualization
au: automatic source instrumentation
d: debugging based on data comparison
g: graphical
m: may be (user experience needed, and/or if we have the machine)
n: no
n,m: no at present time, maybe in the future
p: profile
sl: source-level debugging
sm: summary
st: static visualization
tl: task-level debugging
tr: trace-visualization-based debugging (focus is performance tuning)
tx: text only
u: user-directed source instrumentation
y: yes

Appendix D: Comparison of Parallelization Tools

The table below compares the functions of the tools described in Part 4. It indicates whether a tool performs static code analysis such as dependency analysis, whether it transforms a code to increase parallelism, and whether it generates code (machine code or source code) for target machines. It also indicates whether there are debugging and performance tuning tools associated with the tool.

Parallelization Tools

Name	Analysis/ Transform.	Code Gen.	Debugger	Performance Tool	Useful to NAS	Page Number
Enterprise	n	y	y	y	n	136
Forge 90	y	y	n	y	y	124
fpp	y	y	n	n	y	122
KAP	y	y	n	n	n	123
ParaScope	y	y	y	n	m	127
PAT	y	y	n	y	m	130
Pyrros	n	y	n	n	n	135
Schedule	n	y	y	y	n	134
Tiny	y	n	n	n	m	132

Transform.: Transformation
Gen.: Generation
m: may be (user experience needed, or for tools development)
n: no
n,m: no at present time, maybe in the future
y: yes

Appendix E: Summary of Other Tools

The table below summarizes the tools described in Part 5. The entry "Meta-Tool" indicates if a tool is for the design and development of other tools. The entry Scheduler/Load Balancer indicates if the tools is for task scheduling and/or load balancing. The entry "Network Support" indicates if the tool is to facilitate parallel computation on a network of computers.

Other Tools

Name	Meta-Tool	Scheduler/ Load Balancer	Network Support	Useful to NAS	Page Number
Blobs	n	sim	n	n	153
Condor	n	y	y	m	147
DJM	n	y	y	m	146
DQS	n	n	y	m	144
Funnel	n	y	y	m	145
Gang	n	y	n	n	152
Improv	p	n	n	n,m	139
MNFS	n	n	y	n,m	149
Paradise	p	n	n	n,m	141
Polka	p	n	n	m	143
Prep-P	n	sim	n	n	154
Sage	c	n	n	m	138
TopDomDec	n	y	n	m	150

c: for designing compiler front-end
m: may be (user experience needed)
n: no
n,m: no at present time, maybe in the future
ns: network support
p: for performance tuning tools
sim: through simulation
y: yes