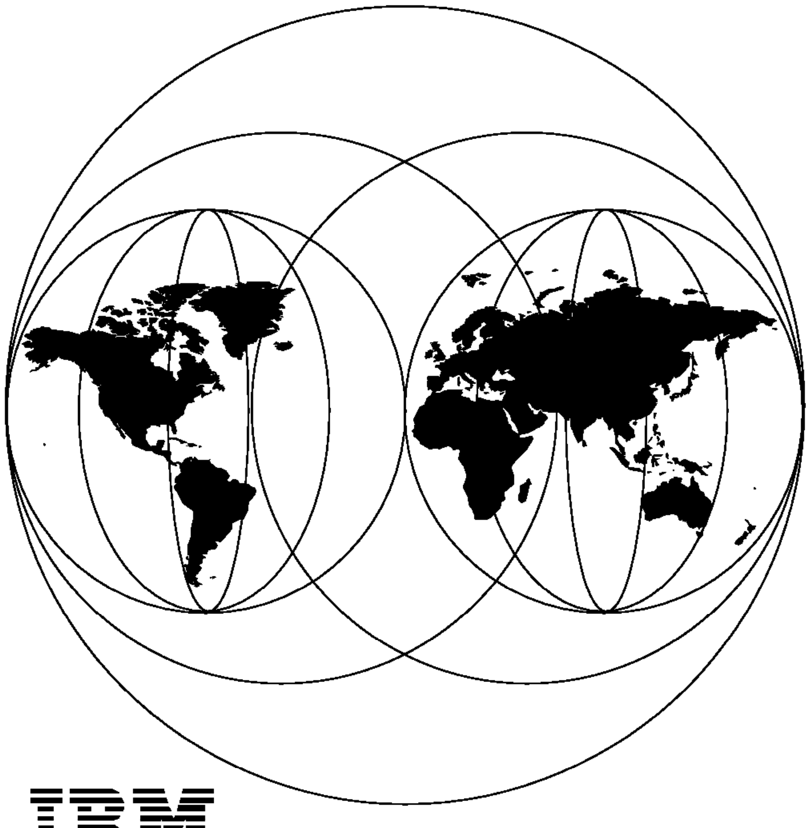
SG24-4541-01

RS/6000 Scalable POWERparallel System: Scientific and Technical Computing Overview

October 1996



International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

SG24-4541-01

RS/6000 Scalable POWERparallel System: Scientific and Technical Computing Overview

October 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, "Special Notices" on page 161.

Second Edition (October 1996)

This edition applies to:

IBM Parallel Environment Version 2 Release 1 for AIX
IBM PVMe Version 2 Release 1
IBM Parallel ESSL Version 1 Release 1 for AIX Version 4
IBM Parallel OSL Version 1 Release 1 for AIX
IBM XL High Performance Fortran Version 1 Release 1 for AIX

for use with IBM AIX Version 4 Release 1.3 and PSSP Version 2 Release 1.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

Contents

Preface . . . v
How This Redbook Is Organized . . . v
The Team That Wrote This Redbook . . . vi
Comments Welcome . . . vi

Chapter 1. Introduction . . . 1
1.1 IBM Software for NIC Applications . . . 2
1.2 Amdahl′s Law Theory . . . 4
  1.2.1 First Case: k=constant . . . 5
  1.2.2 Second Case: k Linear . . . 6
1.3 Remarks . . . 7

Chapter 2. Message Passing Interface . . . 9
2.1 Overview . . . 11
  2.1.1 MCPI Definition . . . 13
  2.1.2 MPI Definition . . . 14
2.2 Message Passing Layers . . . 15
2.3 MPI Enhancement vs MPL . . . 16
  2.3.1 Blocking Communications . . . 18
  2.3.2 Nonblocking Communications . . . 20
  2.3.3 Persistent Communications . . . 23
  2.3.4 Process Group Management . . . 25
  2.3.5 Communicators . . . 27
  2.3.6 Predefined Datatypes . . . 30
  2.3.7 Derived Datatypes . . . 31
2.4 Collective Communications . . . 37
2.5 Reduction Operations . . . 39
2.6 Topologies . . . 40
  2.6.1 Graph Topology . . . 40
  2.6.2 Cartesian Topology . . . 43
  2.6.3 Environmental Management . . . 45
2.7 IBM MPI Enhancement vs MPI Standard . . . 47

Chapter 3. PVMe V2 . . . 49
3.1 Covered Topics . . . 51
3.2 PVMe Overview . . . 52
  3.2.1 PVMe and PVM . . . 54
  3.2.2 PVMe versus PVM . . . 56
  3.2.3 The PVMe Daemon . . . 58
  3.2.4 PVMe Structure . . . 61
  3.2.5 PVMe: New Features . . . 64
3.3 PVMe 2.1 Installation . . . 66
3.4 Writing a PVMe Program . . . 67
  3.4.1 Managing Tasks and Hosts . . . 68
  3.4.2 Packing and Unpacking Messages . . . 70
  3.4.3 Exchanging Messages and Signals . . . 71
  3.4.4 Groups and Collective Communications . . . 72
3.5 Running a PVMe Program . . . 74
  3.5.1 Examples . . . 75
  3.5.2 Options to the PVMe Daemon . . . 78
  3.5.3 PVMe Environment Variables . . . 79

3.5.4 Running on RS/6000 and RS/6000 SP 80

Chapter 4. Parallel ESSL and Parallel OSL 83
4.1 Parallel ESSL 84
4.1.1 Announcement Summary 85
4.1.2 Parallel ESSL Highlights 86
4.1.3 PESSL Design Objectives 87
4.1.4 PESSL Application Support 91
4.1.5 PESSL Operating Environments 92
4.1.6 Parallel ESSL - New Routines 94
4.1.7 What is BLACS? 103
4.1.8 Steps for Using PESSL 105
4.1.9 PESSL/BLACS Call Examples 107
4.2 Parallel OSL 111
4.2.1 Why use OSLp? 112
4.2.2 OSLp Operating Environment 113
4.2.3 OSLp - New Routines 114
4.2.4 Creating OSLp Code from Serial OSL Code 116

Chapter 5. High Performance Fortran 121
5.1.1 Acknowledgements 122
5.2 Topics Covered 123
5.2.1 HPF Announcement (December 5, 1995) 124
5.2.2 HPF - Operating Environment 126
5.2.3 HPF Restrictions 127
5.2.4 HPF vs XLF 128
5.2.5 New HPF Directives 134
5.2.6 HPF Features - SUMMARY 135
5.2.7 ALIGN Directive 137
5.2.8 Distributing Data 141
5.2.9 INDEPENDENT Directive 146
5.2.10 SEQUENCE and Combined Directives 148
5.2.11 FORALL Statement 150
5.2.12 PURE and Extrinsic Procedures 153
5.2.13 Intrinsic Procedures 154
5.2.14 HPF Compiler Options 156

Appendix A. Special Notices 161

Appendix B. Related Publications 163
B.1 International Technical Support Organization Publications 163
B.2 Redbooks on CD-ROMs 163
B.3 Other Publications 163

How To Get ITSO Redbooks 165
How IBM Employees Can Get ITSO Redbooks 165
How Customers Can Get ITSO Redbooks 166
IBM Redbook Order Form 167

List of Abbreviations 169

Index 171

Preface

This redbook provides detailed coverage of the scientific and technical software available on IBM RS/6000 Scalable POWERparallel systems for numeric intensive computing (NIC) applications.

The redbook discusses the following software programs in depth.

IBM Parallel Environment Version 2 Release 1 for AIX.
The new Message Passing Interface (MPI) subroutines.
IBM PVMe Version 2 Release 1 (which is now externally compatible with the public domain PVM 3.3.7).
IBM Parallel ESSL Version 1 Release 1 for AIX Version 4.
IBM Parallel OSL Version 1 Release 1 for AIX.
IBM XL High Performance Fortran Version 1 Release 1 for AIX.

This redbook is of value to IBM specialists and customer specialists who will be developing and administering end-user education. It is in technical presentation format, with foils and related notes to the speaker included, and can also be used as a student handout.

Some knowledge of the AIX 4.1.3 operating system, RISC/6000 SP architecture, and parallel programming application is assumed.

How This Redbook Is Organized

This redbook contains 174 pages. It is organized as follows:

- Chapter 1, "Introduction"

  This chapter provides some information about the parallel programs designed to be executed on parallel machines.

- Chapter 2, "Message Passing Interface"

  The message passing interface (MPI) is the new standard for message passing libraries designed for parallel programs running on distributed memory machines. MPI is now available in IBM Parallel Environment for AIX Version 2. This chapter is an overview of MPI and presents examples of MPI usage.

- Chapter 3, "PVMe V2"

  This chapter describes IBM PVMe Version 2, which IBM developed to take advantage of the RS/6000 SP distributed architecture together with the performances offered by the high-performance switch, and to be externally compatible with the Oak Ridge National Lab. Parallel Virtual Machine (PVM) Version 3.3.7.

- Chapter 4, "Parallel ESSL and Parallel OSL"

  This chapter is a presentation of numeric intensive computing (NIC) libraries developed by IBM:

  – IBM Parallel ESSL for AIX Version 4, which is a set of parallelized ESSL subroutines

  - IBM Parallel OSL for AIX Version 4, which includes a set of parallelized OSL subroutines

- Chapter 5, “High Performance Fortran”

  This chapter describes the new High Performance Fortran (HPF) standard and gives information about the IBM XL HPF compiler.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Poughkeepsie Center.

This project was designed and managed by:

Endy Chiakpo	ITSO, Poughkeepsie Center
Michel Perraud	ITSO, Poughkeepsie Center

The authors of this document are:

Giulia Caliari	IBM Italy
Henry Altaras	IBM Israel
Alexandre Blancke	IBM France
Mario Bono	IBM Australia
Franz Gerharter-Lueckl	IBM Austria

Thanks to the following people for their invaluable contributions to this project:

Rob Clark	IBM Poughkeepsie
Joanna Kubasta	IBM Toronto Laboratory
John Martine	IBM Poughkeepsie
Dave Reynolds	IBM Poughkeepsie
George Wilson	IBM Poughkeepsie
Henry Zongaro	IBM Toronto Laboratory

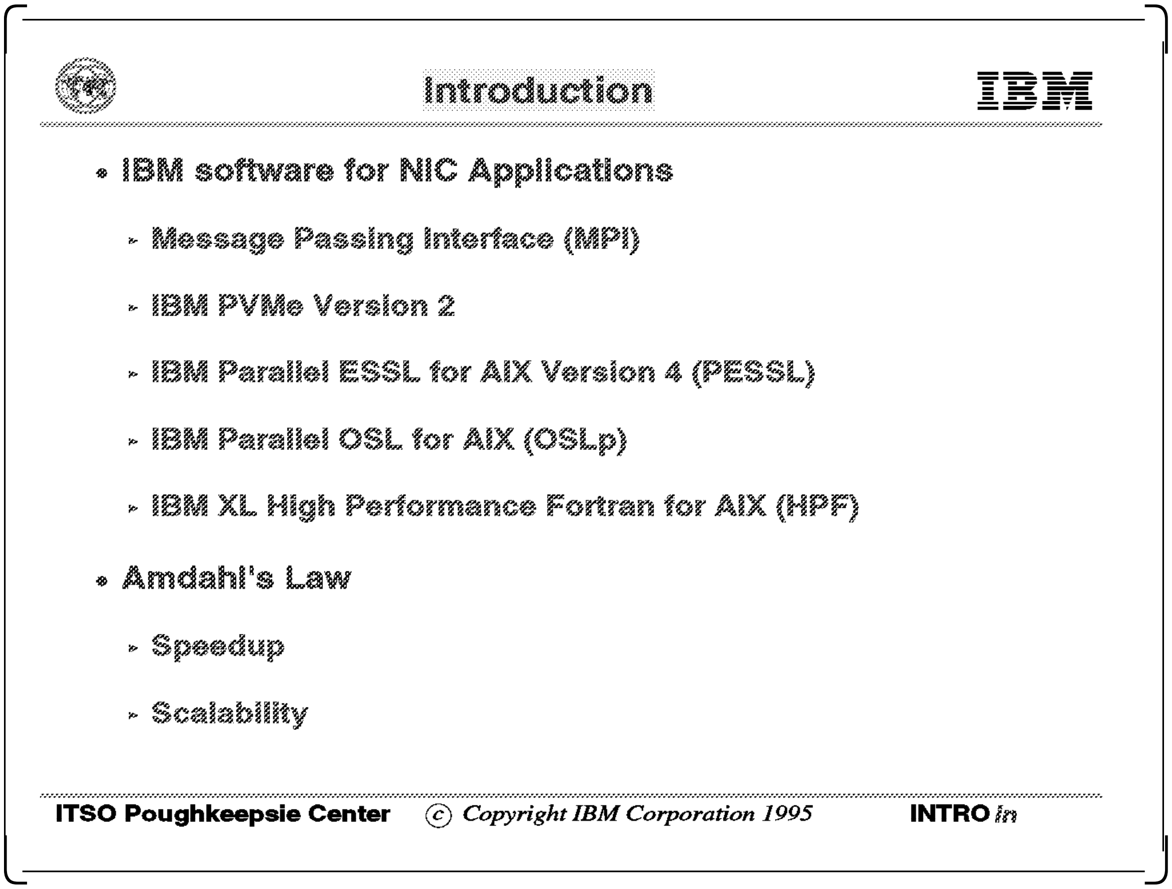
Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Introduction



Introduction

- IBM software for NIC Applications
  - Message Passing Interface (MPI)
  - IBM PVMe Version 2
  - IBM Parallel ESSL for AIX Version 4 (PESSL)
  - IBM Parallel OSL for AIX (OSLp)
  - IBM XL High Performance Fortran for AIX (HPF)
- Amdahl's Law
  - Speedup
  - Scalability

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 INTRO in

In June 1995, IBM announced a set of new versions for RS/6000 SP software:

- IBM Parallel System Support Programs (PSSP) Version 2

  The new features included in PSSP Version 2 are presented in *PSSP Version 2 Technical Presentation*.

- IBM Parallel Environment Version 2 for AIX, which includes the new message passing interface (MPI) library.
- IBM PVMe Version 2, on which message passing subroutines can use the IP protocol on the HPS, and which allows users to run their parallel jobs on a mix of RS/6000 SP nodes and clustered RS/6000 workstations.
- IBM Parallel ESSL for AIX Version 4
- IBM Parallel OSL for AIX

On December 5, 1995, IBM announced the IBM XL High Performance Fortran for AIX.

This book is a technical presentation of this software available for scientific users and NIC applications developers who are working on RS/6000 SP machines. This book is primarily intended for specialists and SEs who have to present the new IBM software and teach users and developers. It includes mid-size foil pictures and the related technical information. This book can also be used as student handout.

The introduction is organized as follows:

- Section 1.1, “IBM Software for NIC Applications” is an overview of IBM software.
- Section 1.2, “Amdahl′s Law Theory” presents some information about Amdahl′s law, which is useful to explain how resources are consumed by a parallel program and which rules determine the parallel program speedup.

1.1 IBM Software for NIC Applications

The IBM software catalog for scientific users and technical computing developers includes the following new features:

Message Passing Interface (MPI)
IBM PE Version 2 provides this new message passing library, which is becoming the de facto standard in scientific and technical computing centers. With this set of message passing subroutines, scientific developers are developing efficient code for complex applications. As it is developed, the code is portable on every platform that provides a library consistent with the MPI standard. On RS/6000 platforms, the MPI subroutines may coexist with subroutines from the MPL library, the former message passing library provided by IBM with PE. So, users may plan their migration to the MPI standard without any break. MPI is presented in Chapter 2, “Message Passing Interface” on page 9.

IBM PVMe Version 2 (PVMe)
IBM PVMe Version 2 is source compatible with the public domain PVM 3.3.7. It allows users to port their PVM applications on RS/6000 SP systems and to obtain better performance with the high performance switch. With PVMe Version 2, users can include in the parallel program node list some RS/6000 SP nodes and complement them with clustered RS/6000 workstations. PVMe Version 2 has been improved to create trace files readable by `xpvm`, the public domain post morten monitor for PVM parallel programs. IBM PVMe Version 2 is described in Chapter 3, “PVMe V2” on page 49.

IBM Parallel ESSL for AIX version 4 (PESSL)
For users who used to use the ESSL subroutines either on mainframes with ESSL/370 or RS/6000 workstations with ESSL/6000, Parallel ESSL for AIX provides a way to migrate applications to parallel environments on RS/6000 SP systems and clustered RS/6000 workstations. A subset of most frequently used ESSL/6000 subroutines have been parallelized. They are described in Section 4.1, “Parallel ESSL” on page 84.

IBM Parallel OSL for AIX (OSLp)
OSLp is now available on RS/6000 SP systems running AIX 3.2.5 and PSSP V1.2 (OSLp 1.2.0), and on RS/6000 SP systems running AIX 4.13 and PSSP V2.1. OSLp provides users with a linear programming solver (LP), a mixed-integer programming solver (MIP), and a solver for problems with a quadratic optimize equation. Problem solving performance is improved when such problems are executed on parallel machines. OSLp is presented in Section 4.2, “Parallel OSL” on page 111.

IBM XL High Performance Fortran for AIX (HPF)

On December 5, 1995, IBM announced the IBM XL HPF compiler. HPF is designed to generate SPMD parallel programs. The compiler generates MPI subroutines calls according to directives included in the source code. These directives show which parts of the code must be parallelized and provide the developer with the means to determine the way data is distributed between the parallel program processes. Users developing their applications with HPF create a portable source code because HPF is going to be available on most parallel and distributed systems. IBM XL HPF is described in Chapter 5, “High Performance Fortran” on page 121.

These products are supported on RS/6000 SP systems by AIX Version 4.1.3 together with PSSP Version 2. MPI, PESSL, OSLp, and HPF are supported on RS/6000 clusters, but these libraries and the HPF compiler imply better performance when the resulting executable runs on RS/6000 SP equipped with the HPS. These software products are developed to take advantage of the RS/6000 SP distributed architecture. This architecture is powerful for scientific and numeric intensive computing (NIC) applications. In fact, according to Amdahl′s law, the speedup users can expect when they run their applications on distributed parallel machines will depend on two factors:

The choice of parallelizable algorithms

The choice of parallelizable algorithms, which implies that you can distribute the computation time to independent tasks that can run simultaneously on different processors.

It generally means that the source code of existing applications must be rewritten to assume the distribution of both data and execution time between independent processors externally connected through a communication path, which can be either a standard network or a specific high-speed connection, such as the RS/6000 SP high-performance switch (HPS).

Parallel programs running on distributed parallel machines may be developed using several models. The models preferred by developers are:

- The SPMD model (single program, multiple data)

  In this model, there is only one executable and each process runs a copy of this single executable. Of course, this executable may include logic that tests the process range in the set of processes, and algorithms are set up according to this range.

- The MPMD model (multiple programs, multiple data)

  In this model, the developer designs the application as a set of different processes generally specialized to execute a specific algorithm. One of the processes, named the master process, is started first, manages the slave process loading on distributed processors, distributes the data to slaves, and gathers the results computed by slaves.

The IBM parallel environments support either SPMD programs or MPMD programs.

Because each process is designed to be executed on different processors, processes must generally exchange data when the parallel program is executed. This is still true when parallel environments, such as the public domain PVM, allow the user to load several processes per processor at execution time.

Data exchange, together with other communications between processes is made easy through ad hoc functions, available as callable subroutines that are included in message passing libraries.

Three message passing libraries are provided by IBM software:

- The PVM library available with PVMe
- The MPL library, included in PE and developed for RS/6000 SP systems before the MPI standard availability
- The MPI library, now available with PE Version 2

PE also includes tools that help developers debug and tune their parallel application using MPL or MPI.

The availability of a fast communication path

The availability of a fast communication path between processes when master and slave processes exchange data or when the master process gathers the results.

On RS/6000 SP, this fast communication path has the HPS, which executes data transfer between nodes, using either the IP protocol or a *user space* protocol specifically developed to optimize the message passing communication through the HPS.

Section 1.2, “Amdahl′s Law Theory” gives a simplified example of Amdahl′s law when applied to a parallel program running on a parallel distributed system.

The speedup function is computed with respect to two variables:

- The number of processor nodes used to execute the parallel program
- The communication function, which represents the elapsed time consumed to communicate between processes that compose the parallel program.

1.2 Amdahl′s Law Theory

Let′s suppose there is no resource constraint except CPU time in a parallel configuration. This hypothesis is realistic when parallel machines are built with processors that can manage very large memories, which is the case of RS/6000 SP systems. Also, NIC programs are generally CPU-bound, with a low I/O rate, which means we can ignore the I/O request interference in the first approximation. So, the performance will only depend on the way the algorithm uses the processor time. Let′s suppose a given serial program processor time can be divided into three parts:

s represents the part of processor time that cannot be parallelized
p represents the part of processor time that can be parallelized
k(n) is the communication function. It represents the overhead related to communications between the application program and other processes. As a first approximation, let′s suppose it is a one-variable function with respect to n, which is the number of nodes.

Let′s suppose the duration of the serial program is the unit of time. So, T_1 and T_n, that are respectively the duration of the serial program, and the duration of the parallelized program executed on a pool of n processors, are as follows:

$$T_1 = p + s + k(1) = 1$$

$$T_n = \frac{p}{n} + s + k(n) = \frac{p + ns + nk(n)}{n}$$

So, the speedup $S(n)$ is:

$$S(n) = \frac{T_1}{T_n} = \frac{1}{T_n} = \frac{n}{p + ns + nk(n)}$$

The derivative function in respect of n is:

$$\frac{dS(n)}{dn} = \frac{p - n^2 k'(n)}{(p + ns + nk(n))^2}$$

1.2.1 First Case: k=constant

If we suppose the $k(n)$ function is constant, which generally occurs for NIC programs with no need to transfer data between nodes (for instance, data is split on local nodes for parallel execution), then the speedup function is:

$$S(n) = \frac{n}{p + n(s + k)}$$

Because $p + s + k = 1 \Rightarrow s + k = 1 - p$, the speedup function becomes:

$$S(n) = \frac{n}{p + n(1 - p)}$$

and its derivative is:

$$\frac{dS(n)}{dn} = \frac{p}{(p + (1 - p)n)^2}$$

For a given p value, this derivative is always positive, and the S(n) function is always increasing with respect to n.

Figure 1 on page 6 presents the Speedup function in respect to the number of processors used to run the parallel program and in respect to p, percent of code that is parallelizable.

As you can observe, the speedup value you can expect is very sensitive to the p value: in fact, for a given p value, the speedup upper limit is:

$$\lim_{n \to \infty} [S(n)] = \frac{1}{1 - p}$$

For instance, if $p = 0.9$, which means the program is parallelizable up to 90%, then the maximum speedup you can expect is 9.343 with 128 processors nodes, and the speedup limit for an infinite number of nodes is 10.

The case $k = constant$ is certainly the most frequent for scientific and NIC applications. In fact, except for the initial step that reads and distributes the data, and the last step that gathers the results and stores them for further use, such parallel programs are CPU-bound and processes are generally independent.

Also, experienced techniques now available in message passing libraries, such as nonblocking communications, buffered communications, and persistent communications, improve the performance because the computation and the communications are optimized and simultaneously dispatched on each node. So, the *s* ratio becomes very low.

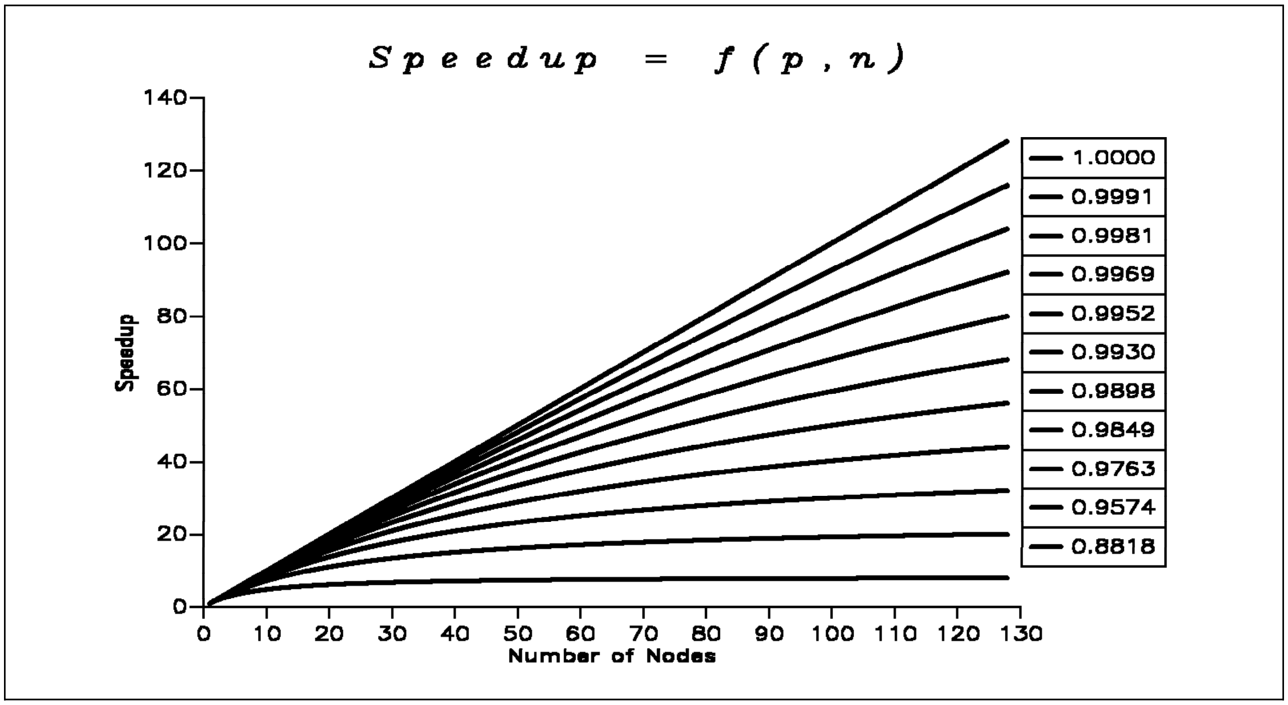


Figure 1. Speedup = f (p,n) for a Constant Communication Function

1.2.2 Second Case: k Linear

Now, let's suppose that the communication function is linear. So, $k(n) = \beta n$ and $k'(n) = \beta$. Then, the speedup function and its derivative become respectively:

$$S(n) = \frac{T_1}{T_n} = T_n = \frac{n}{p + ns + \beta n^2}$$

$$\frac{dS(n)}{dn} = \frac{p - \beta n^2}{(p + ns + \beta n^2)^2}$$

The speedup is increasing when $p - \beta n^2 > 0$. So, $\beta n^2 < p$. But $p \leq 1$; therefore, $\beta n^2 < 1$, and $\beta < \frac{1}{n^2}$.

Figure 2 is a set of curves that shows the impact of *p* over the speedup function, when the communication function $k(n) = \beta n$ is set up with the following values:

$\beta = \frac{1}{64^2}$
$p = 1 - \beta - s$

s varies from 0 to 0.1 by 0.01

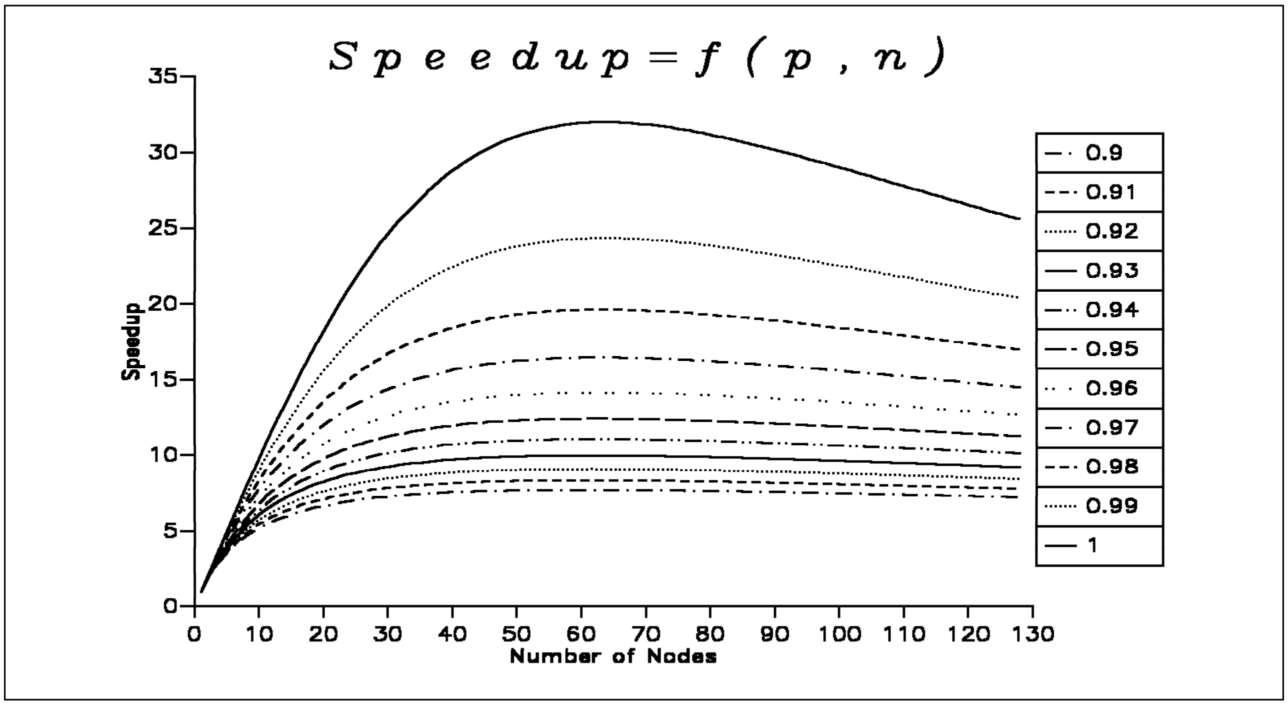


Figure 2. Speedup = f (p,n) for a Linear Communication Function

As one can observe, the communication function is the primary deciding factor for the expected speedup on parallel machines. As a first level approximation, we can consider the serial part of the application is the data transmission time between processes, which is related to the amount of data to be transmitted, and β is related to the number of message passing commands, which is related to the number of nodes.

1.3 Remarks

We can conclude this simplified presentation of Amdahl′s law with the following remarks:

Performance Improvement

- We can expect a good speedup either when the data transmission time is negligible compared to the processor time, or when each process accesses its own data.
- The bandwidth and the latency of the communication path between nodes are the primary deciding factors for the expected speedup.

In fact, we should also evaluate the communication function with respect to the quantity of data to be transmitted. For a given processor configuration, varying the size of data implies a variation of the communication time. Several studies were published on this subject: they show that the chosen algorithm determines the amount of data to be transmitted. These studies, when evaluating the theoretical speedup for a given algorithm, make the point that the communication speed is a keypoint for the speedup increase, which confirms what is deduced with good sense.

However, one can immediately understand the improvement given by nonblocking communications, as they are provided in message passing subroutines, such as PVMe, MPL, or MPI, or the even more efficient buffered communications, and persistent communications, as they are now provided by MPI:

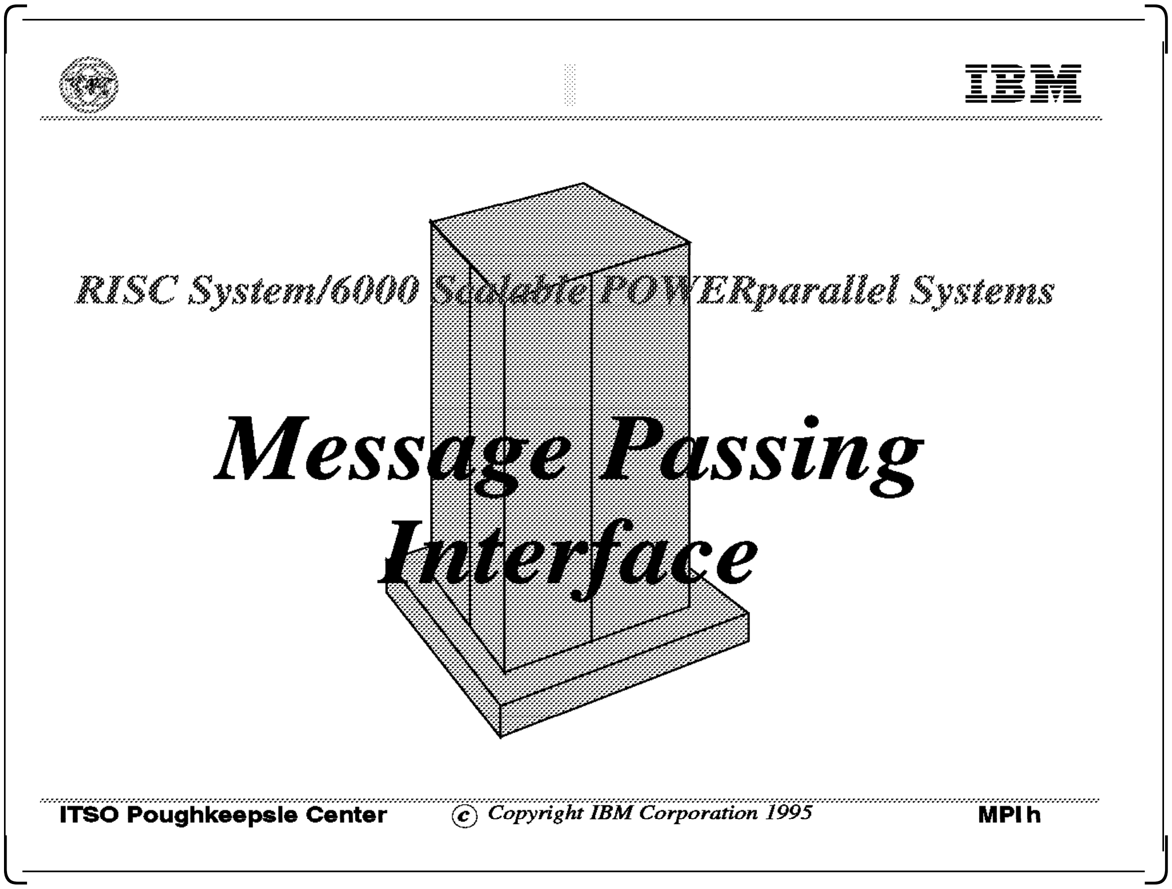
- When a blocking communication is running, the program is waiting and this wait time is included in s, the non-parallelizable part of the program. It has a strong effect on the speedup.
- When a nonblocking communication is running, the program is simultaneously running, and the speedup is not affected by the asynchronous communication.

Easy Development

But parallel programming needs knowledge, experience, and time. To help users develop their parallel code, IBM provides them with parallelized packages, such as Parallel OSL and Parallel ESSL. Furthermore, IBM announced in December 1995, with general availability scheduled in April 1996, a new XL High Performance Fortran compiler, which includes the Fortran 90 statements plus the subset HPF directives to (almost) implicitely generate a Fortran parallel program.

The following chapters include the basics of new IBM products developed to take advantage of RS/6000 SP systems for scientific and technical computing applications.

Chapter 2. Message Passing Interface


IBM
RISC System/6000 Scalable POWERparallel Systems
Message Passing Interface
ITSO Poughkeepsie Center
© Copyright IBM Corporation 1995
MPI h

IBM Parallel Environment for AIX, Version 2 Release 1 includes the following components:

Parallel Operating Environment (POE)
POE is designed to set up the parallel environment for parallel program execution. You can initialize POE either with the *poe* command or by exporting environment variables before the parallel program execution. POE includes a partition manager which manages the node allocation, the program loading, and the standard I/O distribution to nodes.

The parallel debuggers pdbx and xpdbx
The parallel debugger is based on the dbx debugger, and works as a server on the user workstation, while connected to dbx processes running on execution nodes. It gathers information from nodes and displays this data either in line mode (pdbx) or through a X/MOTIF GUI (xpdbx).

Visualization Tool (vt)
VT is a AIXwindows application that displays performance data and graphs in two modes:

Post mortem analysis
The POE can be set up with VT trace file initialization. Then, the parallel program creates a trace file. This trace file is read by the post mortem VT analysis. A set of icons displays windows and graphs that simulate the job

execution according to the trace file content. If the program was compiled with the -g flag, VT is able to display the source code in a window and to highlight the source statements when they are executed.

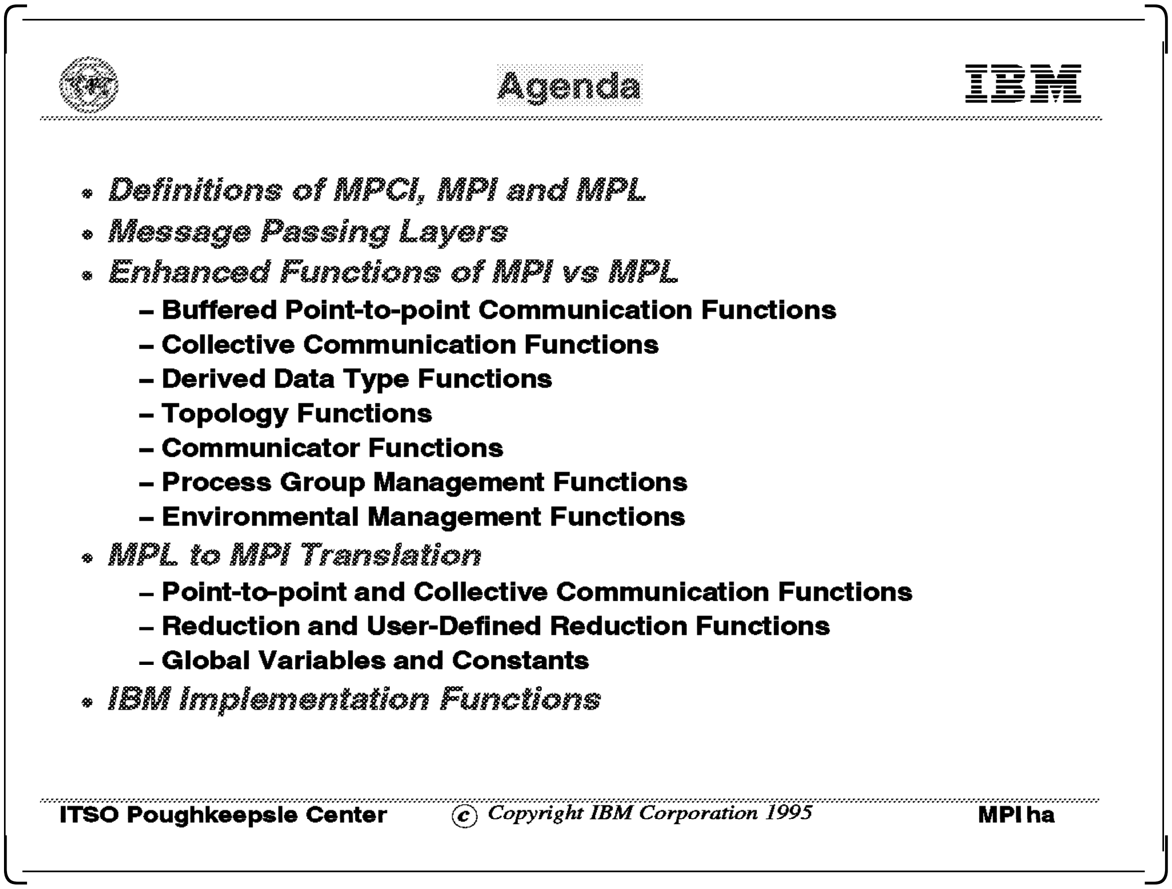
Performance monitoring
Also, VT can be started to display the performance data of running parallel jobs.

The message passing library (MPL)
IBM was part of the consortium that designed the Message Passing Interface destined to be the new standard of message passing library. When the POWERparallel systems were released, MPI was not ready yet, and IBM provided customers with MPL, which is an IBM set of message passing subroutines. Now, MPI is well defined, and becomes the basic message passing library available in IBM Parallel Environment for AIX, Version 2 Release 1.

IBM Parallel Environment for AIX, Version 2 Release 1 continues to provide support for MPL (Message Passing Library), the IBM message passing API. MPL and MPI subroutines can coexist in the same parallel program. The new library includes several IBM extensions (MPE) subroutines. These extensions, though not part of the MPI standard, provide powerful nonblocking collectives functions. It is up to the developer to choose between the conformity of his code with the MPI standard and the use of powerful IBM extensions.

The following presentation is devoted to the new MPI standard.

2.1 Overview



Agenda

- *Definitions of MPCI, MPI and MPL*
- *Message Passing Layers*
- *Enhanced Functions of MPI vs MPL*
  - Buffered Point-to-point Communication Functions
  - Collective Communication Functions
  - Derived Data Type Functions
  - Topology Functions
  - Communicator Functions
  - Process Group Management Functions
  - Environmental Management Functions
- *MPL to MPI Translation*
  - Point-to-point and Collective Communication Functions
  - Reduction and User-Defined Reduction Functions
  - Global Variables and Constants
- *IBM Implementation Functions*

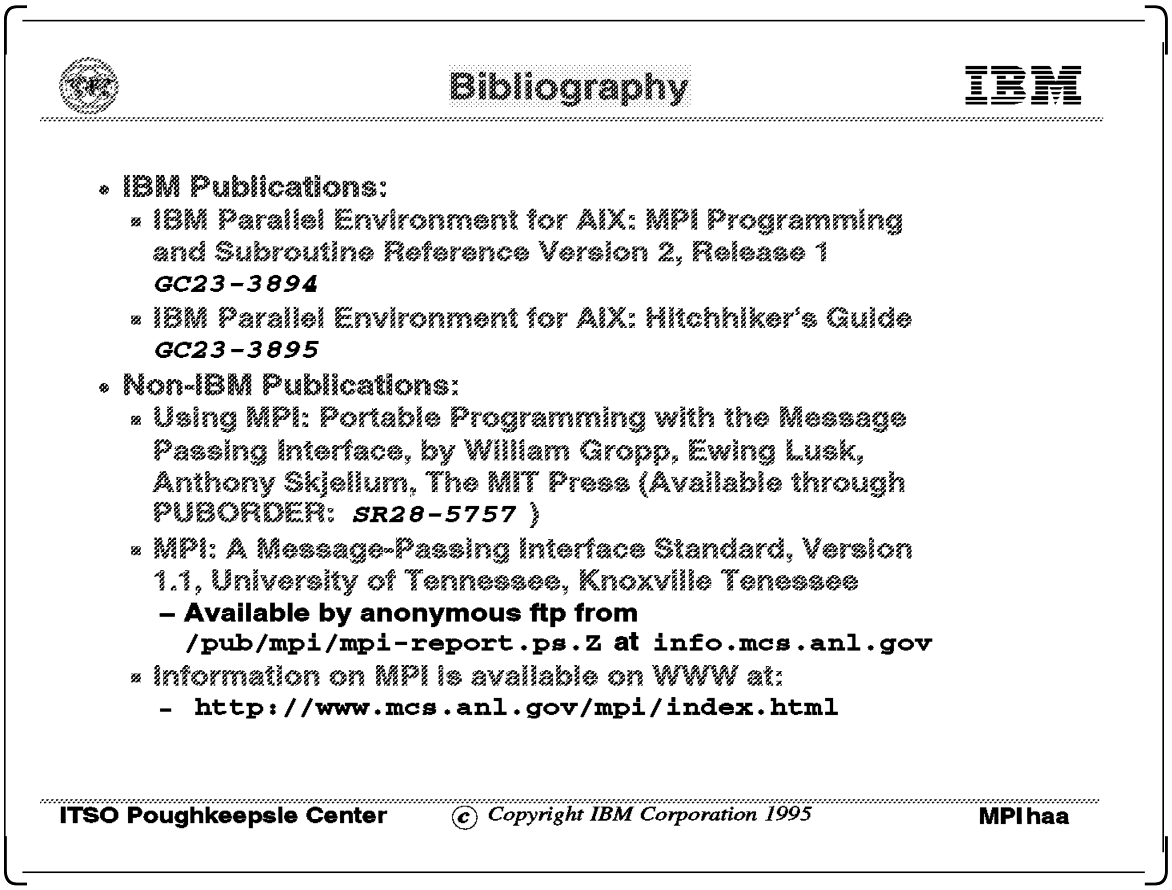
ITSO Poughkeepsie Center (c) *Copyright IBM Corporation 1995* MPI ha

This chapter describes the MPI function and environment used in the IBM implementation using IBM Parallel Environment for AIX, Version 2.1.

It provides basic knowledge about the message passing mechanisms used in the IBM Parallel Environment for scientific and technical computing, and how MPI interfaces with MPCI and MPL.

It also contains usage examples for the most used MPI functions.

Some extensions of MPI are possible. Such subroutines are MPE prefixed for the multiprocessing environment, but it could be message passing extension as well. Using this facility, IBM adds collective nonblocking communication subroutines to the MPI library. They are described in Section 2.7, "IBM MPI Enhancement vs MPI Standard" on page 47.



Bibliography

- IBM Publications:
  - IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference Version 2, Release 1 *GC23-3894*
  - IBM Parallel Environment for AIX: Hitchhiker's Guide *GC23-3895*
- Non-IBM Publications:
  - Using MPI: Portable Programming with the Message Passing Interface, by William Gropp, Ewing Lusk, Anthony Skjellum, The MIT Press (Available through PUBORDER: *SR28-5757*)
  - MPI: A Message-Passing Interface Standard, Version 1.1, University of Tennessee, Knoxville Tenessee
    - **Available by anonymous ftp from `/pub/mpi/mpi-report.ps.Z` at `info.mcs.anl.gov`**
  - Information on MPI is available on WWW at:
    - `http://www.mcs.anl.gov/mpi/index.html`

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **MPI haa**

This foil presents the documentation currently available about MPI.

MPI Programming and Subroutine Reference
This is the IBM MPI reference manual.

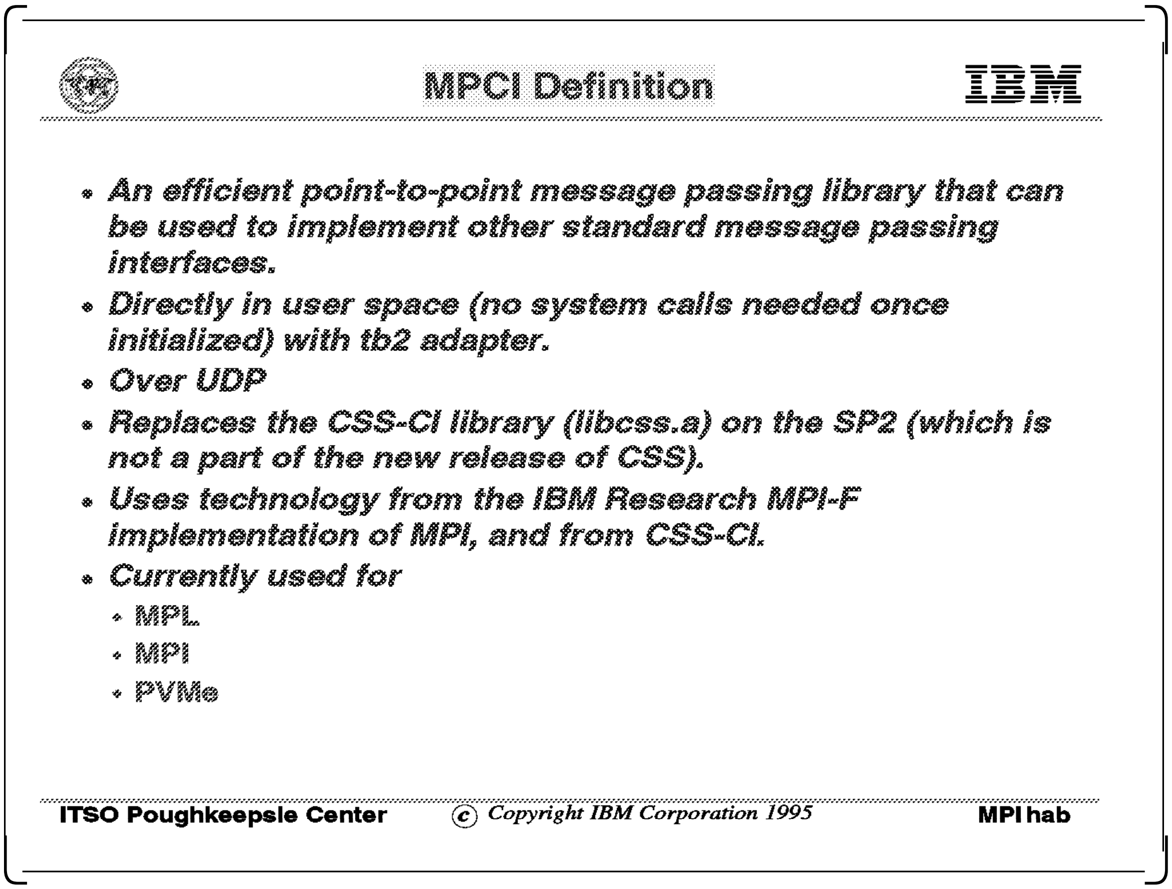
Hitchhiker′s Guide
This book is an MPI primer, with explanations and examples of parallel algorithms and the way you can develop them using MPI.

Using MPI: Portable Programming with Message Passing Interface
Published by MIT, written by William Gropp, Ewing Lusk, and Anthony Skjellum, this book is available through PUBORDER.

The foil gives you the Internet address and the directory that contains the *MPI: A Message Passing Interface Standard, Version 1.1*, as you can get it with anonymous ftp. This server is maintained by the University of Tennessee, which owns the MPI Language Specification together with the MIT Press.

Both organizations have dedicated the language definition to the public domain.

2.1.1 MCPI Definition



MPCI Definition

- An efficient point-to-point message passing library that can be used to implement other standard message passing interfaces.
- Directly in user space (no system calls needed once initialized) with tb2 adapter.
- Over UDP
- Replaces the CSS-CI library (libcss.a) on the SP2 (which is not a part of the new release of CSS).
- Uses technology from the IBM Research MPI-F implementation of MPI, and from CSS-CI.
- Currently used for
  - MPL
  - MPI
  - PVMe

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 MPI hab

When communicating on RS/6000 SP, parallel program occurrences can use either the IP protocol or the IBM user space protocol, which was specifically developed to take advantage of the high performance switch (HPS) architecture. MPCI is the software layer that interfaces the message passing subroutines called by the application program, and the IP or US protocol.

As a common message passing API, MPCI is used by:

- Aix Parallel Environment Version 2.1 MPI or MPL subroutines and functions
- PVMe Version 2.1 subroutines and functions

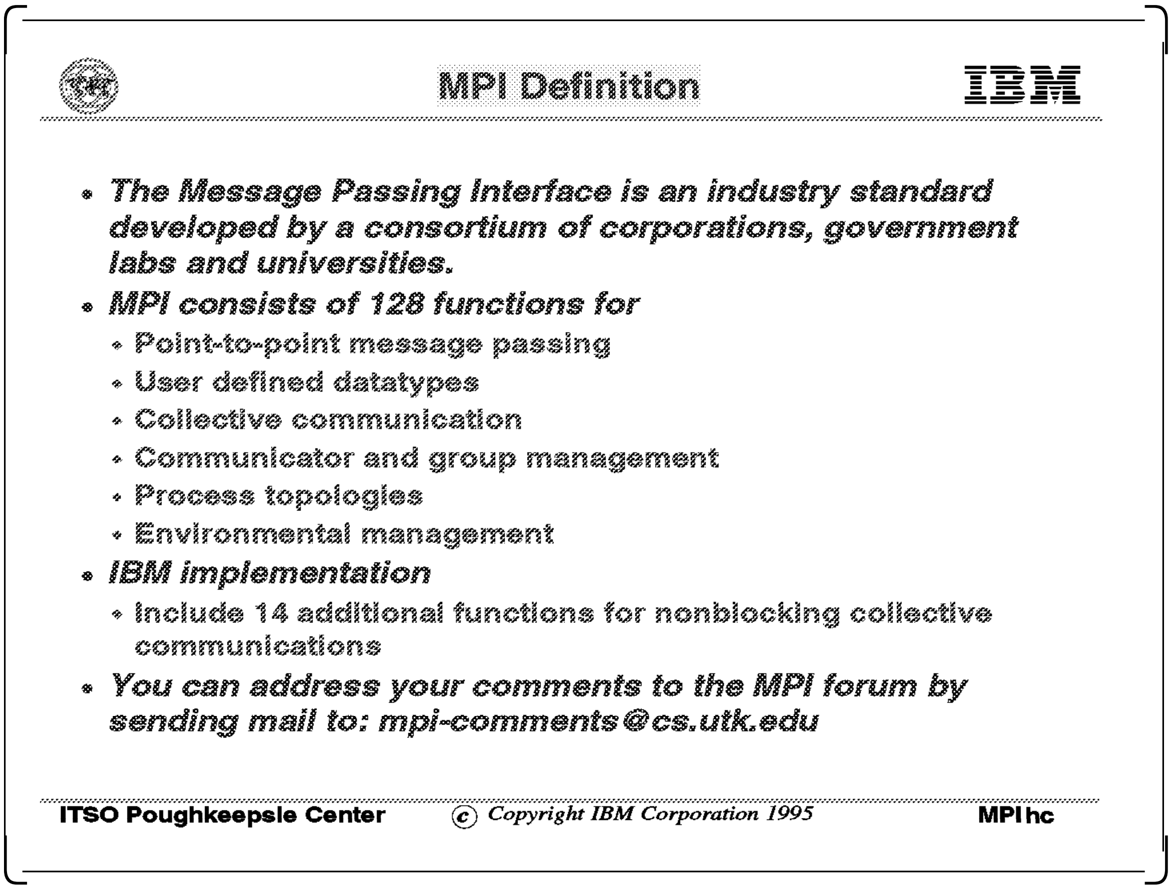
According to the run-time option, communications will use the IP protocol or directly work in user space with HPS Adapter-2. The user space protocol offers better performance because it bypasses most of the AIX kernel and TCP/IP software path length (no system calls needed once initialized).

In the CSS component of PSSP Version 2, the *libmpci.a* replaces the CSS-CI library (*libcss.a*).

MPCI no longer supports HPS Adapter-1 available on IBM 9076 SP1 systems.

MPCI also interfaces with IP for use over non-HPS networks.

2.1.2 MPI Definition



MPI Definition

- *The Message Passing Interface is an industry standard developed by a consortium of corporations, government labs and universities.*
- *MPI consists of 128 functions for*
  - Point-to-point message passing
  - User defined datatypes
  - Collective communication
  - Communicator and group management
  - Process topologies
  - Environmental management
- *IBM implementation*
  - Include 14 additional functions for nonblocking collective communications
- *You can address your comments to the MPI forum by sending mail to: mpi-comments@cs.utk.edu*

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* MPIhc

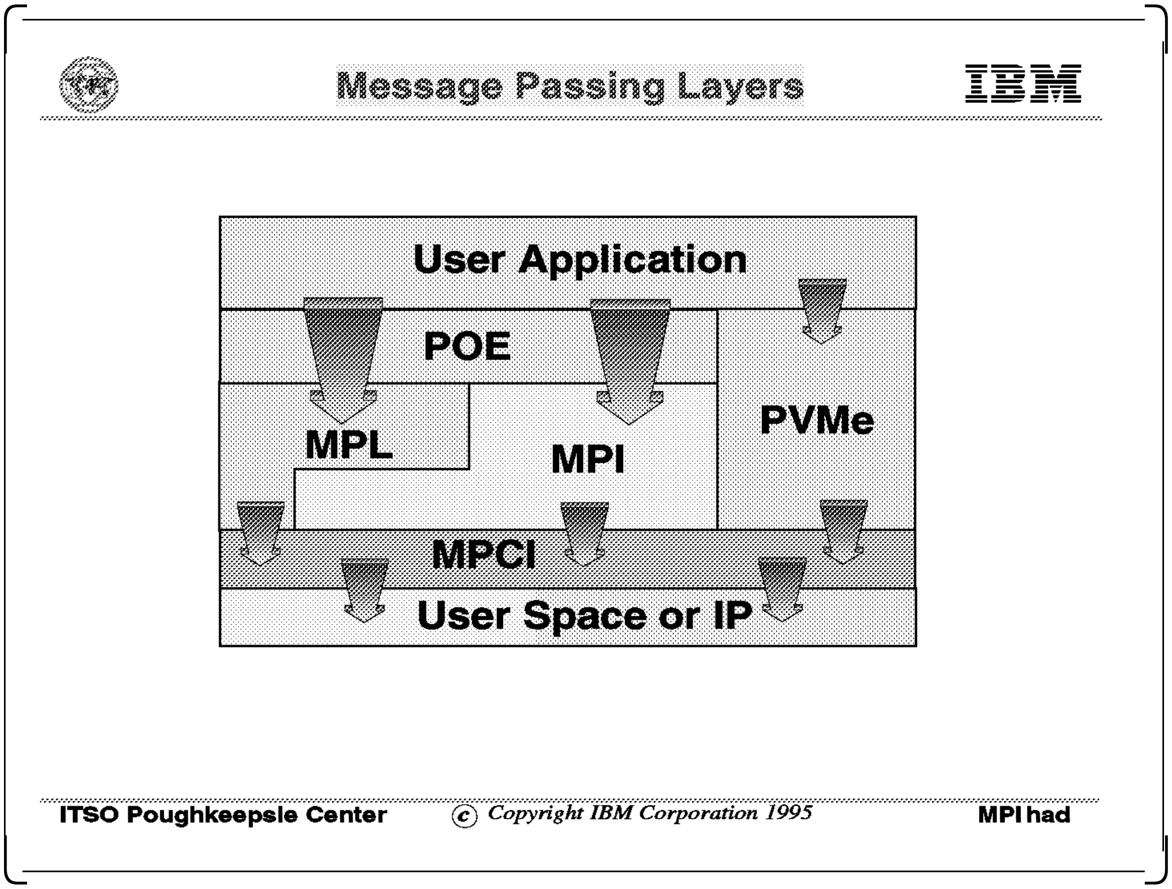
The MPI Standard, as it is copyrighted by the University of Tennessee, was strongly influenced by:

- Work at the IBM T.J. Watson Research Center
- Intel NX/2
- Express
- nCUBE′s Vertex
- PARMACS
- Chimp
- PVM
- PICL

The MPI standard library provides functions for:

- Blocking, nonblocking and synchronized message passing between pairs of processes
- Selectivity of messages by source process and message type
- Context control
- Ability to form and manipulate process groups

2.2 Message Passing Layers



The MPCI layer supports both IP and user space protocols. The protocol to be used can be specified at run time without having to link executables:

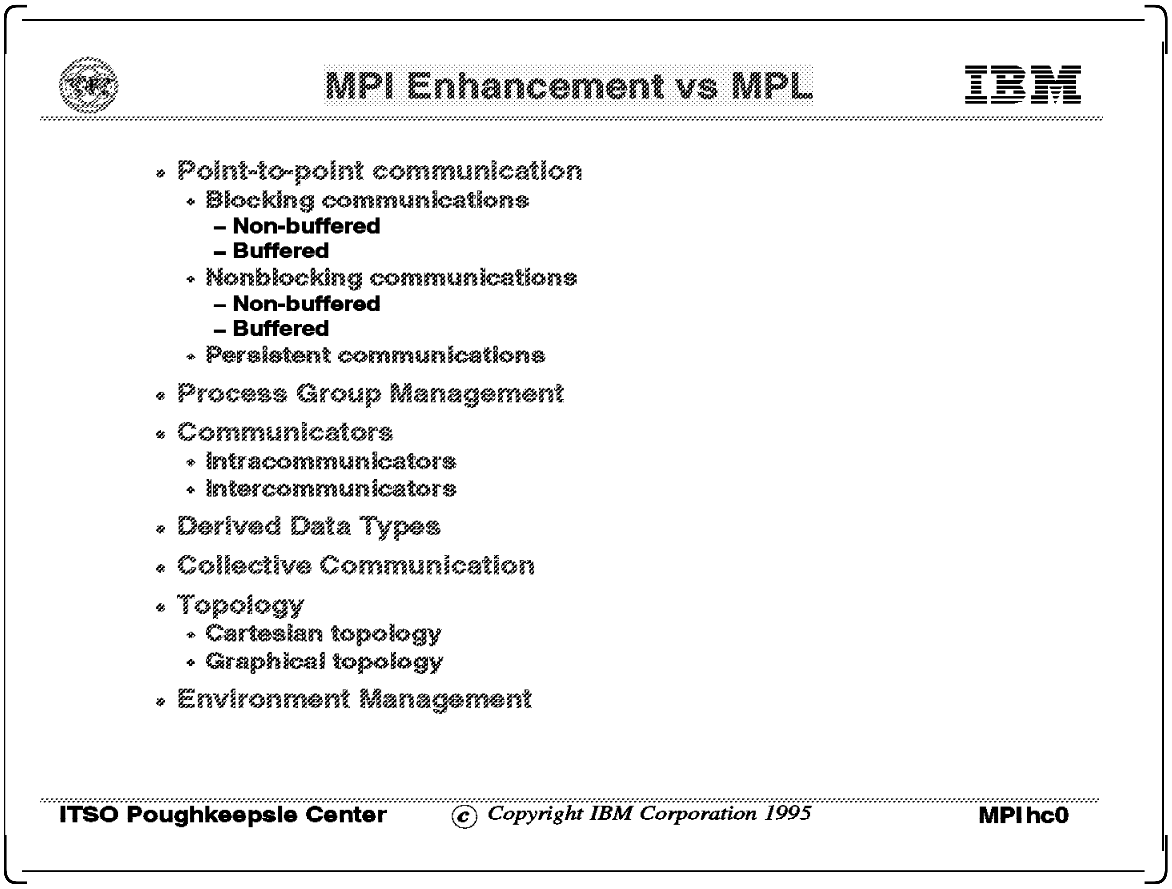
- To initialize the parallel operating environment (POE), you can either export environment variables or use the corresponding poe command flags. So, the communication protocol is set up with:

  ```
  export MP_EUILIB={ip|us}
  poe ... -euilib {ip|us}
  ```

- Using PVMe, the default communication protocol is user space. When you want to use the IP communication protocol through the switch, you specify the *-ip* option when starting the *pvmd3e* daemon.

In IBM Parallel Environment Version 2, several MPL subroutines and functions are rewritten and implicitly use the corresponding MPI routines.

2.3 MPI Enhancement vs MPL



MPI Enhancement vs MPL

- Point-to-point communication
  - Blocking communications
    - Non-buffered
    - Buffered
  - Nonblocking communications
    - Non-buffered
    - Buffered
  - Persistent communications
- Process Group Management
- Communicators
  - Intracommunicators
  - Intercommunicators
- Derived Data Types
- Collective Communication
- Topology
  - Cartesian topology
  - Graphical topology
- Environment Management

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* MPI hc0

MPI subroutines are logically grouped this way:

Point-to-Point Communication

Point-to-point communication is basic in every message passing library. Subroutines of this kind allow single communications between two processes, such as a message send or a message receive.

Section 2.3.1, “Blocking Communications” on page 18 describes the blocking communication subroutines, when the calling process is waiting for an operation complete acknowledgement before being reactivated.

Section 2.3.2, “Nonblocking Communications” on page 20 presents the nonblocking communication subroutines. MPI returns an immediate acknowledgement when the request is queued. This request will be executed in asynchroneous mode, which improves performances, but the calling process cannot reuse the message area before request completion verification.

Section 2.3.3, “Persistent Communications” on page 23 describes the persistent communication. MPI subroutines are provided to describe requests once and keep their description persistent. So, several identical requests can be sent to MPI without the overhead for request initialization.

Process Group Management

In sophisticated parallel programs, generally using the MPMD model, groups of processors are devoted to specific computing, while other processors communicate with their neighbors, for exchanging data. Naturally, a smart programmer can develop such a process relationship his way, but the result will probably be unstable and difficult to maintain or improve.

MPI provides developers with a set of subroutines for node group management. These subroutines are presented in Section 2.3.4, “Process Group Management” on page 25.

Communicators

This set of subroutines creates the information needed by MPI to manage communications inside a group of processes defined as a closed shop. Section 2.3.5, “Communicators” on page 27 presents the communicator management subroutines.

Derived Datatypes

Section 2.3.6, “Predefined Datatypes” on page 30 presents the predefined datatypes available in MPI.

When you want to send a set of data discontinuous in memory and with heterogeneous datatypes, you can use the derived datatypes to define your own data structure. This data structure becomes the basic element to be transmitted between processes. The derived datatypes are described in Section 2.3.7, “Derived Datatypes” on page 31.

Collective communication

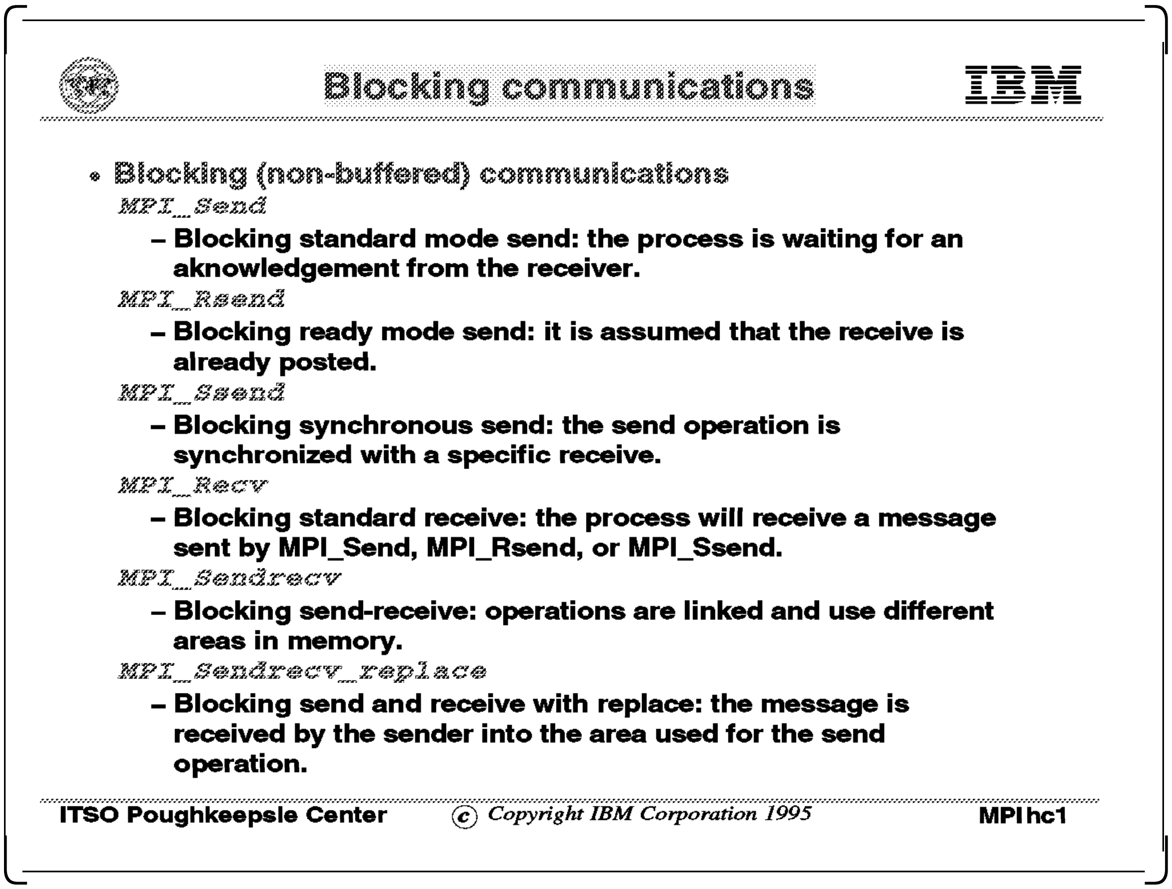
The set of MPI collective communication subroutines is described in Section 2.4, “Collective Communications” on page 37.

The IBM extension provides nonblocking collective communication subroutines listed in Section 2.7, “IBM MPI Enhancement vs MPI Standard” on page 47.

Topology

In MIMD sophisticated programs, several algorithms are based on the knowledge of node neighbors. This management becomes painful with respect to the number of nodes. The graph topology and the cartesian topology subroutines help programmers design the node neighborhood and easily manage communications in the neighborhood. More information and examples are given in Section 2.6, “Topologies” on page 40.

2.3.1 Blocking Communications



Blocking communications

- Blocking (non-buffered) communications

  MPI_Send
  - Blocking standard mode send: the process is waiting for an aknowledgement from the receiver.

  MPI_Rsend
  - Blocking ready mode send: it is assumed that the receive is already posted.

  MPI_Ssend
  - Blocking synchronous send: the send operation is synchronized with a specific receive.

  MPI_Recv
  - Blocking standard receive: the process will receive a message sent by MPI_Send, MPI_Rsend, or MPI_Ssend.

  MPI_Sendrecv
  - Blocking send-receive: operations are linked and use different areas in memory.

  MPI_Sendrecv_replace
  - Blocking send and receive with replace: the message is received by the sender into the area used for the send operation.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* MPI hc1

Blocking communications are serialized, that is, a process that issues a such request is in wait state up to the operation complete acknowledgement.

The following subroutines request a blocking operation:

MPI_Send Blocking standard mode send.

MPI_Rsend The blocking ready mode send, MPI_Rsend, is destined to a specific receive request from a specific node. So, the sender is in wait state until the receive request is posted. You can use this subroutine if you expect the receiver node has already executed the receive request.

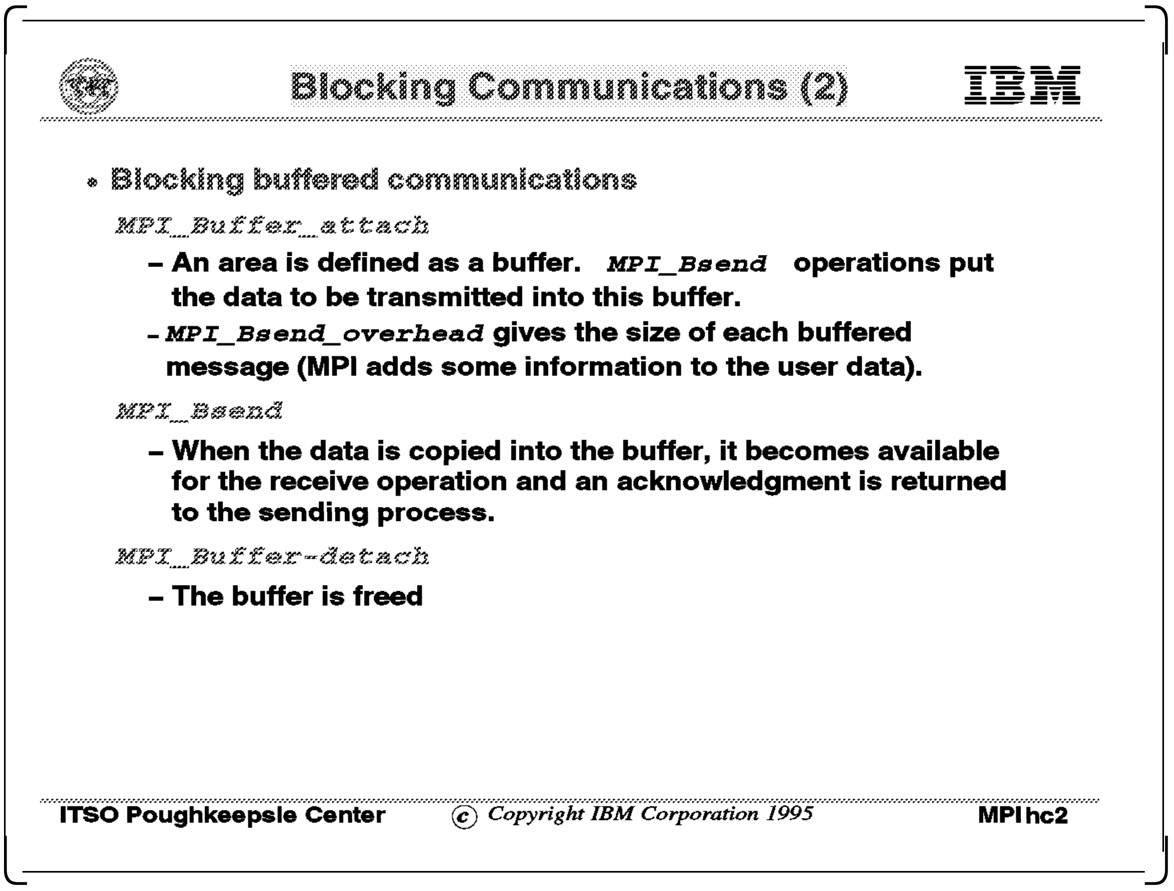
MPI_Ssend The blocking synchronous mode send, MPI_Ssend, can be started any time. It will be complete when the matching receive is started. If the receive request is a blocking operation, the communication is synchronous.

MPI_Sendrecv MPI_Sendrecv is a blocking send and receive operation using different buffers, while MPI_Sendrecv_replace uses the same buffer for send and receive operations.

MPI_Recv

MPI_Recv is the unique standard blocking mode receive.

Note: MPI_Sendrecv gives better performances than standard blocking mode subroutines because of the simultaneity of both operations.



Blocking Communications (2)

- Blocking buffered communications

  MPI_Buffer_attach
  - An area is defined as a buffer. *MPI_Bsend* operations put the data to be transmitted into this buffer.
  - *MPI_Bsend_overhead* gives the size of each buffered message (MPI adds some information to the user data).

  MPI_Bsend
  - When the data is copied into the buffer, it becomes available for the receive operation and an acknowledgment is returned to the sending process.

  MPI_Buffer-detach
  - The buffer is freed

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* MPI hc2

MPI provides subroutines for blocking buffered communications:

MPI_Buffer_attach In the sending process involved in a buffered communication, a work area is specified by the MPI_Buffer_attach subroutine. The application puts the messages into the buffer using the MPI_Bsend subroutine. The receiving process gets the message from the buffer with MPI_Recv.

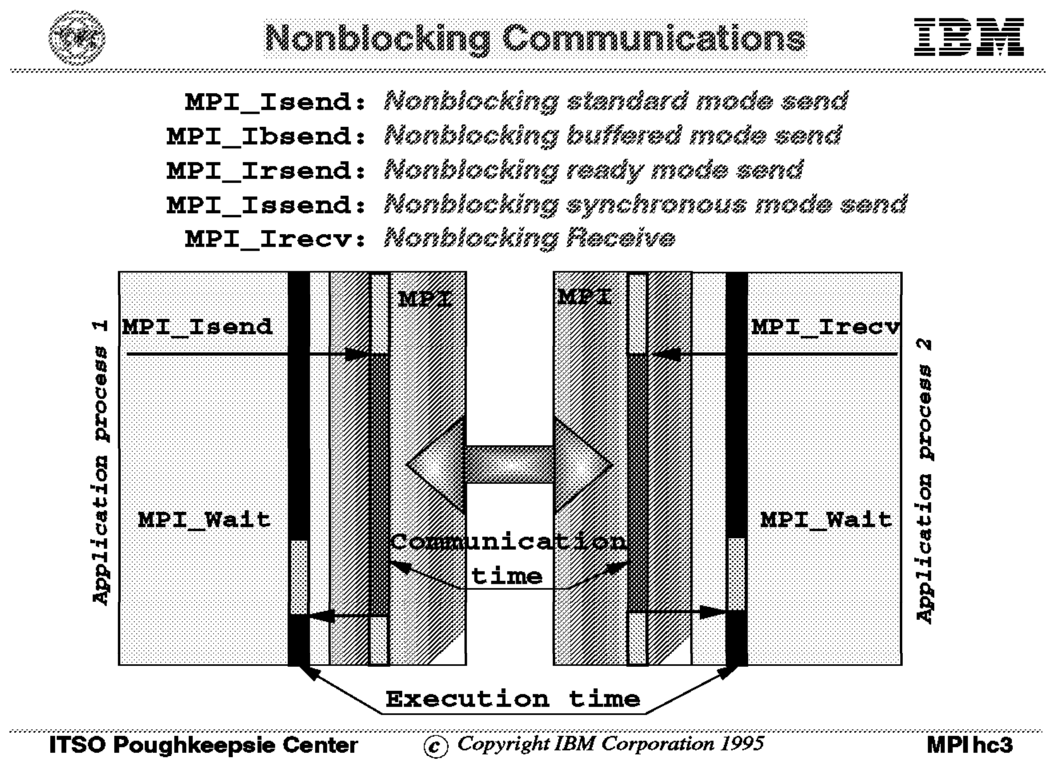
MPI_Bsend The MPI_Bsend subroutine puts a message into the buffer. When the message is copied, an acknowledgment is returned to the sending process. The message is available for the MPI_Recv receive operation.

MPI_Buffer_detach When buffered messages have been received, the buffer can be detached using the MPI_Buffer_detach subroutine.

Using blocking buffered communications means that you want to store your messages in a buffer predefined with the MPI_Buffer_attach subroutine. You can define a buffer for several messages. You define your own buffer management according to the following rules:

- The buffer size must be large enough to contain all messages that can potentially reside into the buffer at the same time.
- MPI adds its own information to each message. The actual size of a message will be given by the MPI_Bsend_overhead.
- An error occurs when a send is executed on an already full buffer.
- The space that contains a buffered message is freed when the receive request is complete. Before any buffer space reuse, you have to test the receive completion.
- You can define only one buffer in a process at a time.

2.3.2 Nonblocking Communications



Nonblocking Communications

MPI_Isend: *Nonblocking standard mode send*
MPI_Ibsend: *Nonblocking buffered mode send*
MPI_Irsend: *Nonblocking ready mode send*
MPI_Issend: *Nonblocking synchronous mode send*
MPI_Irecv: *Nonblocking Receive*

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 MPI hc3

The nonblocking communication is more efficient in terms of performance because communications between processes are executed asynchroneously. So, each occurrence of the parallel program executes its own operations with a certain level of multitasking. For instance, a process can be computing some data, while sending preceding results and receiving the next data.

However, such programming is not so easy:

- You must not access the buffer where the data is stored before the operation completes.
- Each operation returns a request number. This request number is transmitted to MPI in the parameter list of the following subroutines when you wait for the request completion, or when you ask MPI for the request completion status:

  MPI_Wait waits for the completion of a specific request.

  MPI_Waitall refers to an array that contains the list of request numbers you want to wait for.

  MPI_Waitany This subroutine is waiting for any specified requests to complete.

  MPI_Waitsome This subroutine is waiting for some specified requests to complete.

MPI_Test returns the status of a specific request identified by its request number.

MPI_Testall returns the status of a list requests identified by their request numbers in an array.

MPI_Testany tests for completion of any previously initiated request.

MPI_Testsome tests for completion of some previously initiated request.

Your computing task will be waiting until the end of requests that reserve the buffers you want to use. The communication between processes is assumed by MPI in asynchronous mode.

You can use the following subroutines for nonblocking communications:

MPI_Isend MPI_Isend is the standard nonblocking send. A request value is set up by MPI for each MPI_Isend. This request value is specified in the MPI_Wait subroutine parameter list to indicate the operation you are waiting for when the MPI_Wait request is sent to MPI. The wait state lasts until the receive operation is complete.

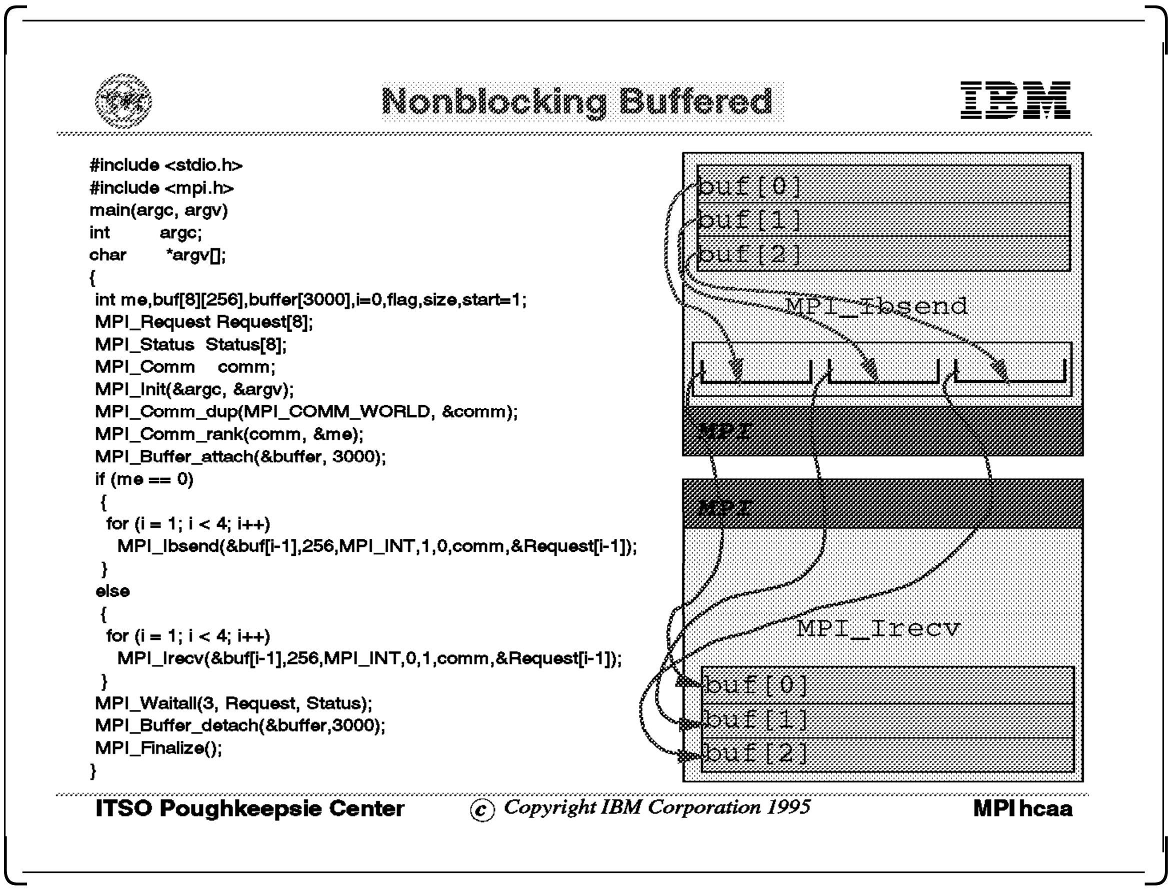
MPI_Ibsend MPI_Ibsend is the nonblocking buffered send that puts the message into the buffer defined by `MPI_Buffer_attach`.

MPI_Irsend MPI_Irsend is a nonblocking ready send. The destination task must have posted a matching receive before you post the send request.

MPI_Issend MPI_Issend is a nonblocking synchronous send. It means that your request completes only when a matching receive will be posted.

Each nonblocking function is assigned to a request number, so you can manage them using an MPI_Waitxxx function or an MPI_Testxxx one.

In this foil, process 1 transmits a `MPI_Isend` request to MPI and continues its local computing until it needs to use some resource reserved by MPI. Process 2 transmits a `MPI_Irecv` to MPI, which provides the message. Process 2 continues its local processing until it needs resources held by MPI. Both processes test the receive completion using one of MPI_Wait or MPI_Test subroutines. They are in wait state until the receive completion.



Nonblocking Buffered

```
#include <stdio.h>
#include <mpi.h>
main(argc, argv)
int         argc;
char        *argv[];
{
 int me,buf[8][256],buffer[3000],i=0,flag,size,start=1;
 MPI_Request Request[8];
 MPI_Status  Status[8];
 MPI_Comm    comm;
 MPI_Init(&argc, &argv);
 MPI_Comm_dup(MPI_COMM_WORLD, &comm);
 MPI_Comm_rank(comm, &me);
 MPI_Buffer_attach(&buffer, 3000);
 if (me == 0)
  {
   for (i = 1; i < 4; i++)
     MPI_Ibsend(&buf[i-1],256,MPI_INT,1,0,comm,&Request[i-1]);
  }
 else
  {
   for (i = 1; i < 4; i++)
     MPI_Irecv(&buf[i-1],256,MPI_INT,0,1,comm,&Request[i-1]);
  }
 MPI_Waitall(3, Request, Status);
 MPI_Buffer_detach(&buffer,3000);
 MPI_Finalize();
}
```

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 MPIhcaa

A nonblocking buffered send request stores the message into a buffer previously specified to MPI with MPI_Buffer_attach. Then, the local process can continue its operations while the message is available for the receive operation. You can use nonblocking buffered sends with the following restrictions:

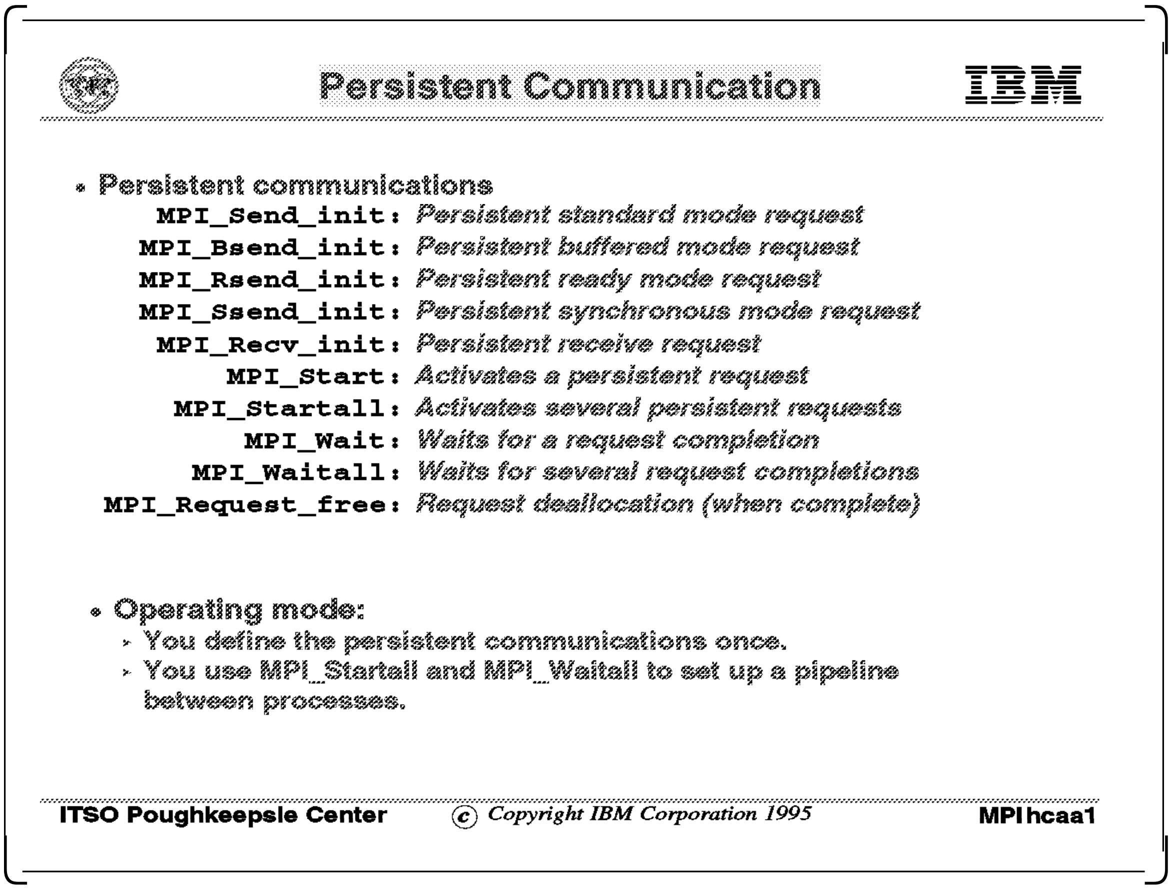
- You cannot reuse the buffer used for a nonblocking communication while you are not sure MPI has freed your buffer. You must manage the state of your request using some MPI_Waitxxx or MPI_Testxxx routine (see Section 2.3.2, "Nonblocking Communications" on page 20).
- You must define enough buffers to contain the maximum number of nonblocking requests you need to manage.
- Using the same buffer space for different nonblocking communications without being sure that it is free may produce unpredictable results.

The example shown in the foil presents a SPMD program using nonblocking buffered communications. Process 0 uses MPI_Ibsend to put messages into the buffer, and process 1 uses MPI_Irecv to get the messages from the buffer.

When the MPI_Ibsend or the MPI_Irecv operations have been started, each process can continue its own operations except those using the buffer space reserved for the current messages (not shown in this example).

In each process, MPI_Waitall indicates the processes will be waiting for request completion and will be resumed when all current requests are complete.

2.3.3 Persistent Communications



Persistent Communication

- Persistent communications

MPI_Send_init:	*Persistent standard mode request*
MPI_Bsend_init:	*Persistent buffered mode request*
MPI_Rsend_init:	*Persistent ready mode request*
MPI_Ssend_init:	*Persistent synchronous mode request*
MPI_Recv_init:	*Persistent receive request*
MPI_Start:	*Activates a persistent request*
MPI_Startall:	*Activates several persistent requests*
MPI_Wait:	*Waits for a request completion*
MPI_Waitall:	*Waits for several request completions*
MPI_Request_free:	*Request deallocation (when complete)*

- Operating mode:
  - You define the persistent communications once.
  - You use MPI_Startall and MPI_Waitall to set up a pipeline between processes.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* MPIhcaa1

The concept of persistent communication can be summarized this way: in some parallel programs, a process repetitively executes identical send and receive operations (same senders and receivers, same messages, and so on). To avoid the overhead due to each operation initialization, MPI provides a set of subroutines that describe operations without execution, and these descriptions are persistent.

So, it is not necessary to redo the communication setting for each identical operation; and you start one operation with MPI_Start, or you start all currently described operations with MPI_Startall. Then, you test the completion of operations respectively with MPI_Wait or MPI_Waitall. This way, you define a continous flow of data transfer through the so called pipe-line:

- In your program, the first paragraph describes persistent communication with MPI_xxxx_init subroutines.
- A second paragraph is a loop including MPI_Startall and MPI_Waitall.
- The last paragraph includes the MPI_Request_free subroutine to unspecify persistent communications.

The possible persistent communications you can define are as follows:

MPI_Send_init Specifies a persistent communication request for a standard mode send.

MPI_Bsend_init Specifies a persistent communication request for a buffered mode send.

MPI_Rsend_init Specifies a persistent communication request for a ready mode send.

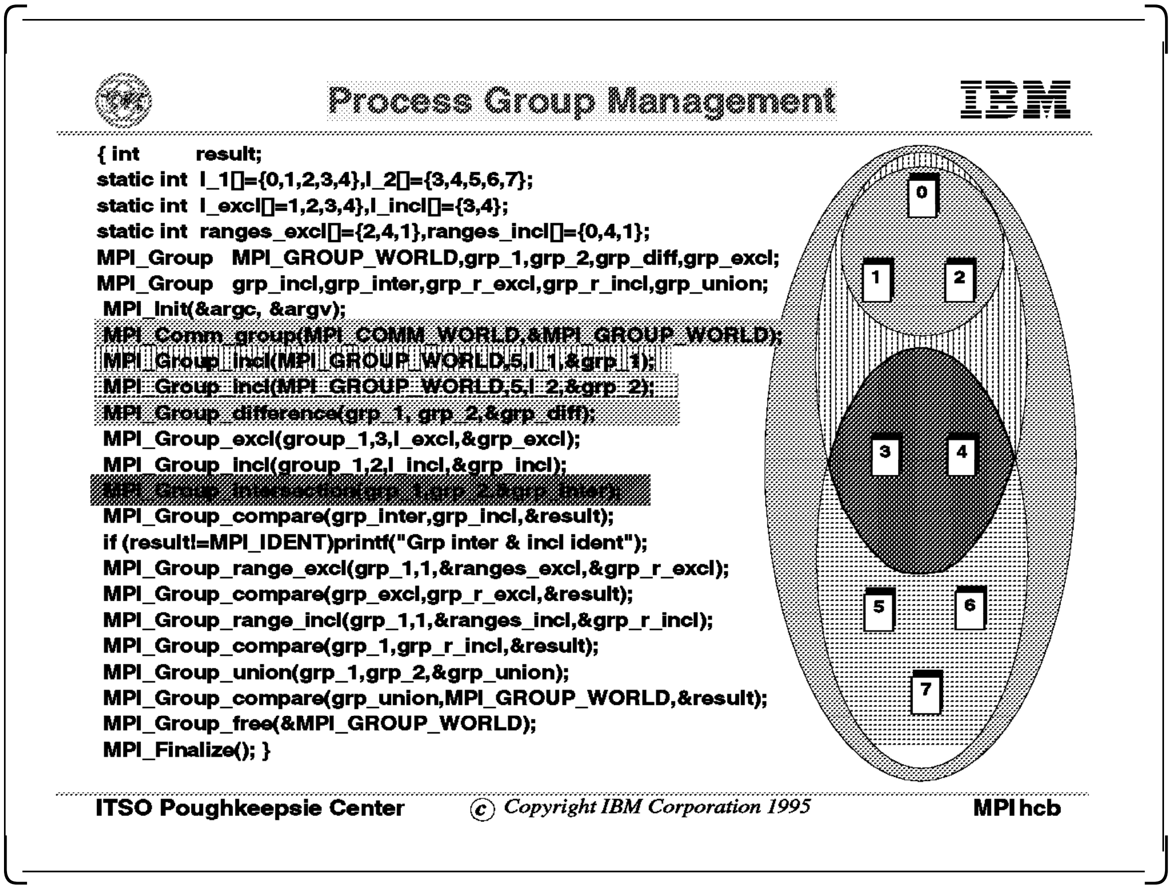
MPI_Ssend_init Specifies a persistent communication request for a synchronous mode send.

MPI_Start (MPI_Startall) Starts one or several requests.

MPI_Wait (MPI_Waitall) Waits for one or several request completions.

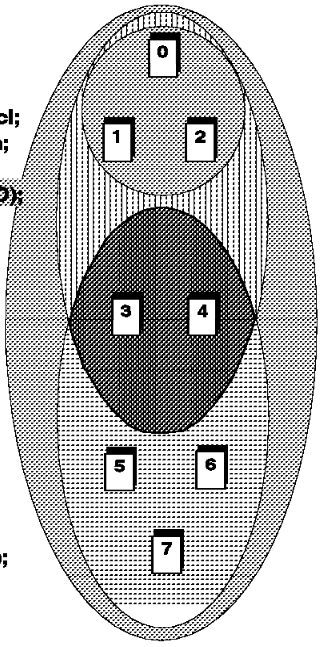
MPI_Request_free Frees a persistent communication request.

2.3.4 Process Group Management



Process Group Management

```
{ int       result;
static int  l_1[]={0,1,2,3,4},l_2[]={3,4,5,6,7};
static int  l_excl[]=1,2,3,4},l_incl[]={3,4};
static int  ranges_excl[]={2,4,1},ranges_incl[]={0,4,1};
MPI_Group   MPI_GROUP_WORLD,grp_1,grp_2,grp_diff,grp_excl;
MPI_Group   grp_incl,grp_inter,grp_r_excl,grp_r_incl,grp_union;
 MPI_Init(&argc, &argv);
 MPI_Comm_group(MPI_COMM_WORLD,&MPI_GROUP_WORLD);
 MPI_Group_incl(MPI_GROUP_WORLD,5,l_1,&grp_1);
 MPI_Group_incl(MPI_GROUP_WORLD,5,l_2,&grp_2);
 MPI_Group_difference(grp_1, grp_2,&grp_diff);
 MPI_Group_excl(group_1,3,l_excl,&grp_excl);
 MPI_Group_incl(group_1,2,l_incl,&grp_incl);
 MPI_Group_intersection(grp_1,grp_2,&grp_inter);
 MPI_Group_compare(grp_inter,grp_incl,&result);
 if (result!=MPI_IDENT)printf("Grp inter & incl ident");
 MPI_Group_range_excl(grp_1,1,&ranges_excl,&grp_r_excl);
 MPI_Group_compare(grp_excl,grp_r_excl,&result);
 MPI_Group_range_incl(grp_1,1,&ranges_incl,&grp_r_incl);
 MPI_Group_compare(grp_1,grp_r_incl,&result);
 MPI_Group_union(grp_1,grp_2,&grp_union);
 MPI_Group_compare(grp_union,MPI_GROUP_WORLD,&result);
 MPI_Group_free(&MPI_GROUP_WORLD);
 MPI_Finalize(); }
```



ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPI hcb

When the number of processes involved in a parallel program, particularly when the model MPMD model is chosen, the design and development of process group management becomes a little bit difficult. To simplify this process group management, MPI provides a set of subroutines that allow the developer to create and manipulate the process groups needed by the application design.

Once a group is defined, each process is ordered in a list from zero to $n - 1$ (n is the number of processes that belong to the group), and the process rank will be used in place of the actual process number. The group definition will be used to specify communicators as described in the next foil.

A process group is an ordered set of process identifiers. Each process in a group is associated with a rank. Ranks are contiguous and start from zero.

The example will create the following groups:

MPI_Comm_group returns the group handle associated with the "WORLD" communicator. It means that it will return a group containing all the tasks started by the MPI_Init, here tasks zero to seven.

MPI_Group_incl The two MPI_Group_incl will create:

- *grp_1* including process in the *l_1 {0,1,2,3,4}*
- *grp_2* including process in the *l_2 {3,4,5,6,7}*

MPI_Group_difference The MPI_Group_difference will create the *grp_diff*, which is the difference *{0,1,2}* between *grp_1* and *grp_2* (elements of the first group that are not in the second group).

MPI_Group_excl The MPI_Group_excl will create the *grp_excl*, which will contain tasks from *grp_1* minus tasks *{1,2,3}*.

MPI_Group_intersection The MPI_Group_intersection will create the group *grp_inter* that will contain the tasks *{3,4}* included in both *grp_1* and *grp_2*.

MPI_Group_range_excl The MPI_Group_range_excl will create the group *grp_r_excl* that will contain tasks from *grp_1* excluded tasks from ranges *2* to *4*.

MPI_Group_range_incl The MPI_Group_range_incl will create the group *group_r_incl* that will contain tasks from *grp_1* including only tasks from ranges *0* to *4* that in that case is equal to the whole *group_1*.

MPI_Group_compare Compares the two groups specified in the parameter list. The integer result is set up to one of the following symbolic values:

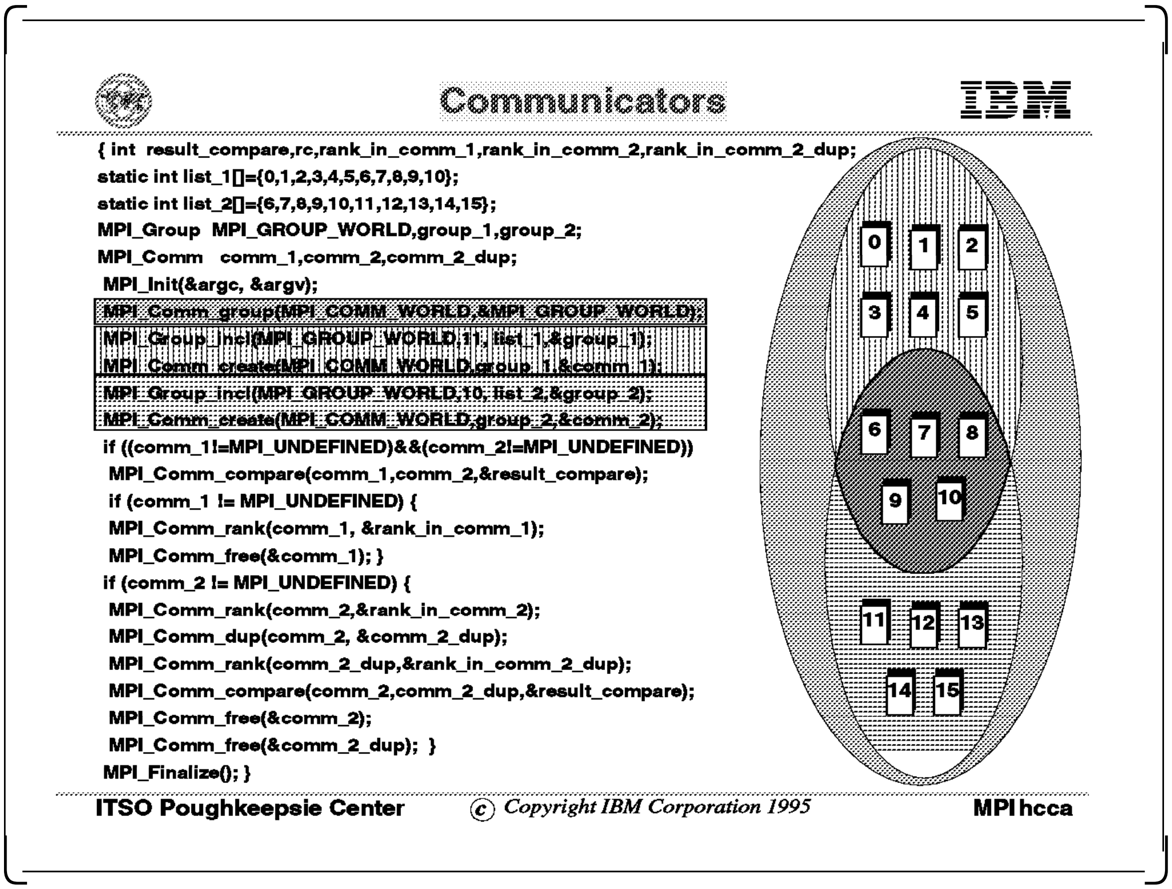
- *MPI_IDENT* when both groups include the same processes in the same order
- *MPI_SIMILAR* when both groups include the same processes in a different order
- *MPI_UNEQUAL* when group size or members are different

MPI_Group_union This subroutine creates a new group, *grp_union* in this example, which includes all elements from both groups *grp_1* and *grp_2*.

MPI_Group_free prepares the group deallocation:

- The group is marked for deallocation.
- Pending requests specifying the group in the parameter list will be complete before deallocating the group.
- New requests specifying the group in the parameter list will be rejected.

2.3.5 Communicators



Communicators

```
{ int  result_compare,rc,rank_in_comm_1,rank_in_comm_2,rank_in_comm_2_dup;
static int list_1[]={0,1,2,3,4,5,6,7,8,9,10};
static int list_2[]={6,7,8,9,10,11,12,13,14,15};
MPI_Group  MPI_GROUP_WORLD,group_1,group_2;
MPI_Comm   comm_1,comm_2,comm_2_dup;
 MPI_Init(&argc, &argv);
MPI_Comm_group(MPI_COMM_WORLD,&MPI_GROUP_WORLD);
MPI_Group_incl(MPI_GROUP_WORLD,11,list_1,&group_1);
MPI_Comm_create(MPI_COMM_WORLD,group_1,&comm_1);
MPI_Group_incl(MPI_GROUP_WORLD,10,list_2,&group_2);
MPI_Comm_create(MPI_COMM_WORLD,group_2,&comm_2);
 if ((comm_1!=MPI_UNDEFINED)&&(comm_2!=MPI_UNDEFINED))
  MPI_Comm_compare(comm_1,comm_2,&result_compare);
  if (comm_1 != MPI_UNDEFINED) {
  MPI_Comm_rank(comm_1, &rank_in_comm_1);
  MPI_Comm_free(&comm_1); }
 if (comm_2 != MPI_UNDEFINED) {
  MPI_Comm_rank(comm_2,&rank_in_comm_2);
  MPI_Comm_dup(comm_2, &comm_2_dup);
  MPI_Comm_rank(comm_2_dup,&rank_in_comm_2_dup);
  MPI_Comm_compare(comm_2,comm_2_dup,&result_compare);
  MPI_Comm_free(&comm_2);
  MPI_Comm_free(&comm_2_dup);  }
 MPI_Finalize(); }
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPIhcca

The definition of a communicator could be a kind of “tube” that is associated with a group to be used by the tasks in the group to communicate with other tasks in the group. In the example, we can find the following MPI functions to manipulate communicators:

MPI_Comm_group is used to retrieve the group handle associated with a communicator.

MPI_Comm_create MPI_Comm_create will create *comm_1* and *comm_2* respectively associated with *group_1* and *group_2*.

MPI_Comm_compare returns *MPI_UNEQUAL* in *result_compare* because, in this example, the groups associated with *comm_1* and *comm_2* are different. The other possible values returned by MPI_Comm_compare are:

- *MPI_IDENT*
- *MPI_CONGRUENT*
- *MPI_SIMILAR*

MPI_Comm_rank returns the *rank* of the process in the communicator. A process can have multiple ranks, as it can use more than one communicator that may not include the same group of processes.

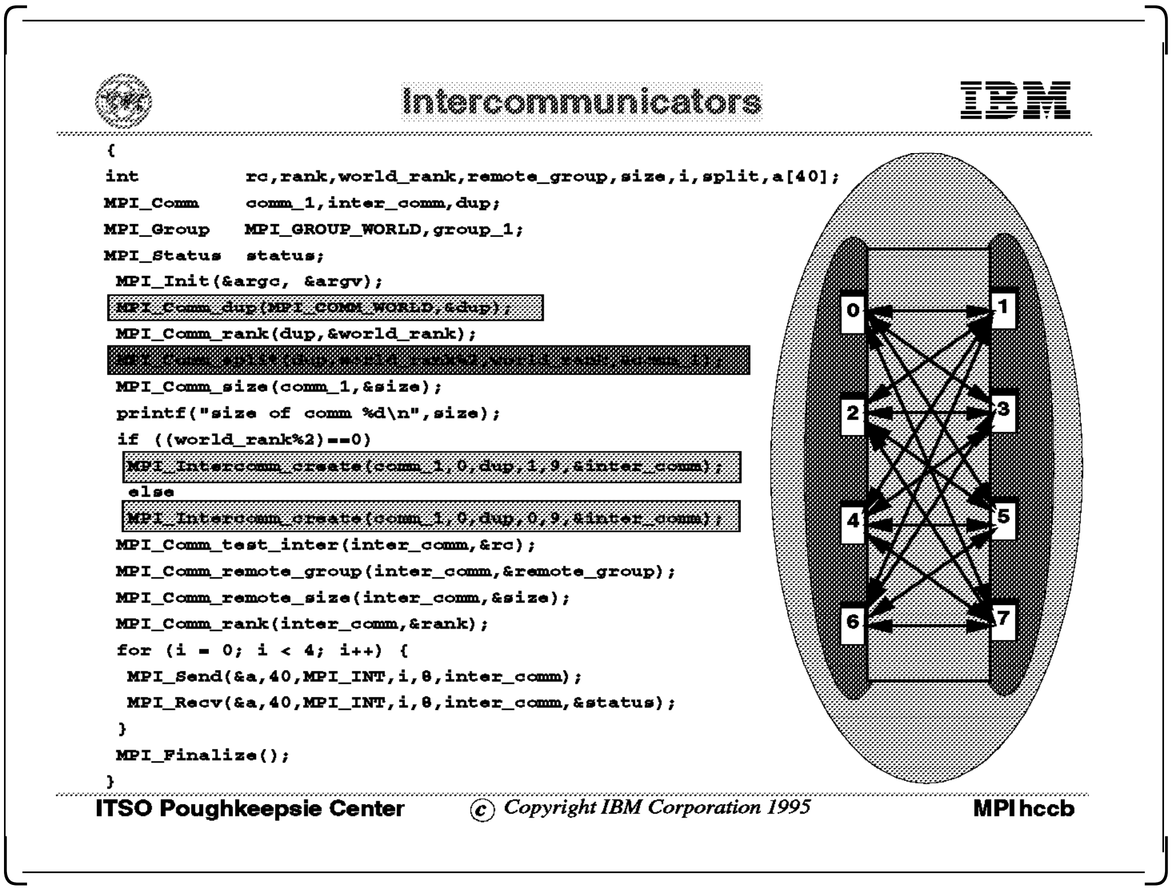
MPI_Comm_dup will create the *comm_dup_2* communicator, which will have the same characteristics as the *comm_2*.

MPI_Comm_free will free the communicator handle so that it can be used to define another one.

Table 1 gives the actual process number and the process ranks as specified in the example shown in this foil.

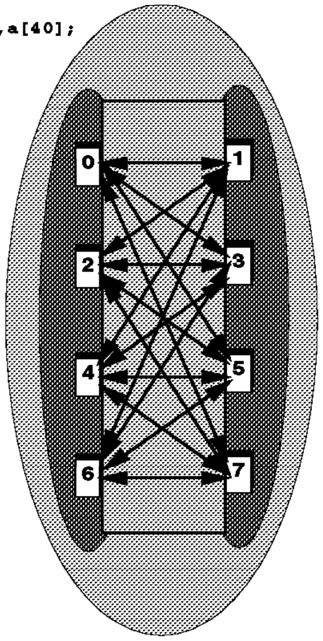
Table 1. Tasks Relative Ranks

Task	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
rank_in_comm_1	0	1	2	3	4	5	6	7	8	9	10					
rank_in_comm_2							0	1	2	3	4	5	6	7	8	9



Intercommunicators

```
{
int         rc,rank,world_rank,remote_group,size,i,split,a[40];
MPI_Comm    comm_1,inter_comm,dup;
MPI_Group   MPI_GROUP_WORLD,group_1;
MPI_Status  status;
 MPI_Init(&argc, &argv);
 MPI_Comm_dup(MPI_COMM_WORLD,&dup);
 MPI_Comm_rank(dup,&world_rank);
 MPI_Comm_split(dup,world_rank%2,world_rank,&comm_1);
 MPI_Comm_size(comm_1,&size);
 printf("size of comm %d\n",size);
 if ((world_rank%2)==0)
  MPI_Intercomm_create(comm_1,0,dup,1,9,&inter_comm);
  else
  MPI_Intercomm_create(comm_1,0,dup,0,9,&inter_comm);
 MPI_Comm_test_inter(inter_comm,&rc);
 MPI_Comm_remote_group(inter_comm,&remote_group);
 MPI_Comm_remote_size(inter_comm,&size);
 MPI_Comm_rank(inter_comm,&rank);
 for (i = 0; i < 4; i++) {
  MPI_Send(&a,40,MPI_INT,i,8,inter_comm);
  MPI_Recv(&a,40,MPI_INT,i,8,inter_comm,&status);
 }
 MPI_Finalize();
}
```



ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 MPI hccb

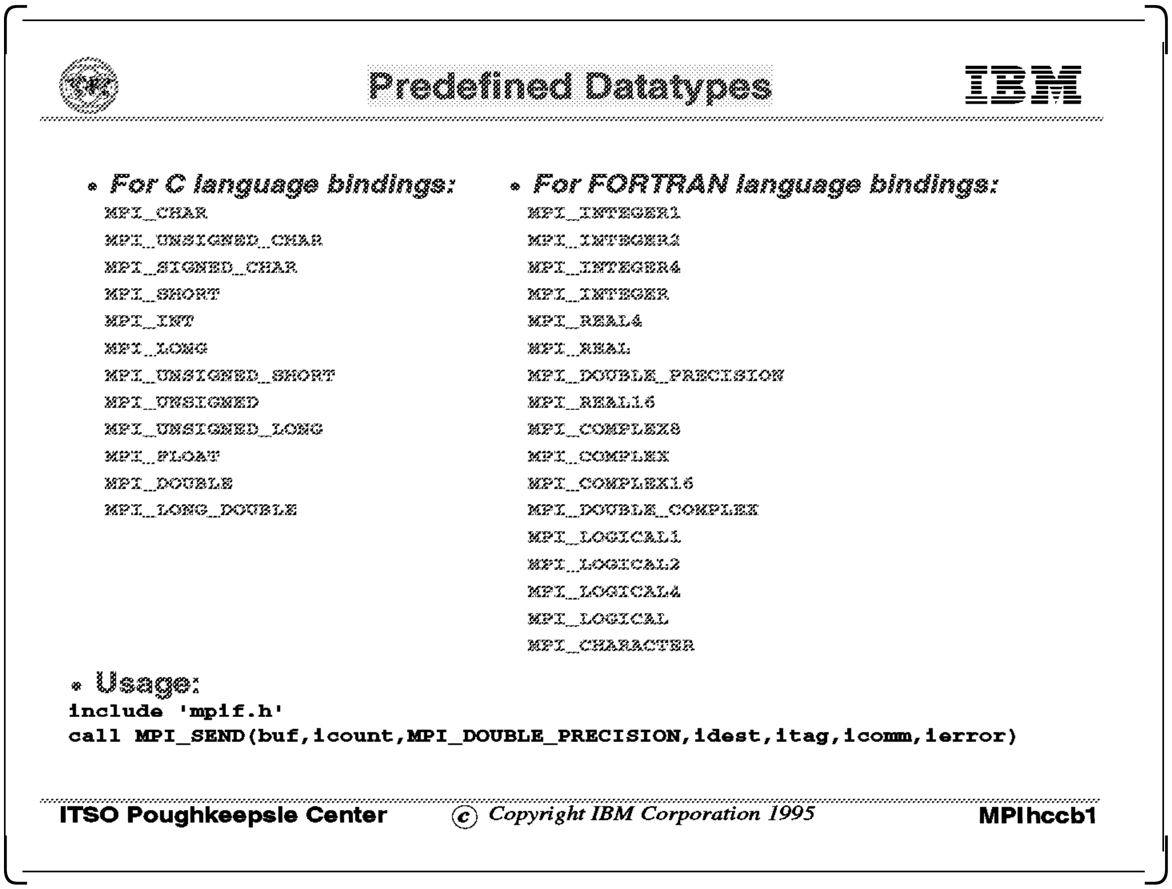
Customer applications may need to create separate groups and associated communicators that will have different kinds of work to do. These are known as intracommunicators.

Also, they may need to make two intracommunicators communicate with each others. These are known as intercommunicators.

The example consists of:

- MPI_Comm_split that will split the *dup_world* communicator into two separate intracommunicators. The split will be done by separate even and odd ranks of processes in *dup_world*. The two intracommunicators will have the same logical name *comm_1* but not the same "physical" tasks.
- MPI_Intercomm_create will create the *inter_comm* intercommunicator that will be used by tasks in *comm_1* to talk with tasks in *comm_2*. So when a task in *comm_1* talks to the task of rank *2* in *inter_comm*, it will talk effectively to tasks of rank *5* in *comm_dup*

2.3.6 Predefined Datatypes



Predefined Datatypes

- *For C language bindings:*
  - MPI_CHAR
  - MPI_UNSIGNED_CHAR
  - MPI_SIGNED_CHAR
  - MPI_SHORT
  - MPI_INT
  - MPI_LONG
  - MPI_UNSIGNED_SHORT
  - MPI_UNSIGNED
  - MPI_UNSIGNED_LONG
  - MPI_FLOAT
  - MPI_DOUBLE
  - MPI_LONG_DOUBLE
- *For FORTRAN language bindings:*
  - MPI_INTEGER1
  - MPI_INTEGER2
  - MPI_INTEGER4
  - MPI_INTEGER
  - MPI_REAL4
  - MPI_REAL
  - MPI_DOUBLE_PRECISION
  - MPI_REAL16
  - MPI_COMPLEX8
  - MPI_COMPLEX
  - MPI_COMPLEX16
  - MPI_DOUBLE_COMPLEX
  - MPI_LOGICAL1
  - MPI_LOGICAL2
  - MPI_LOGICAL4
  - MPI_LOGICAL
  - MPI_CHARACTER
- Usage:

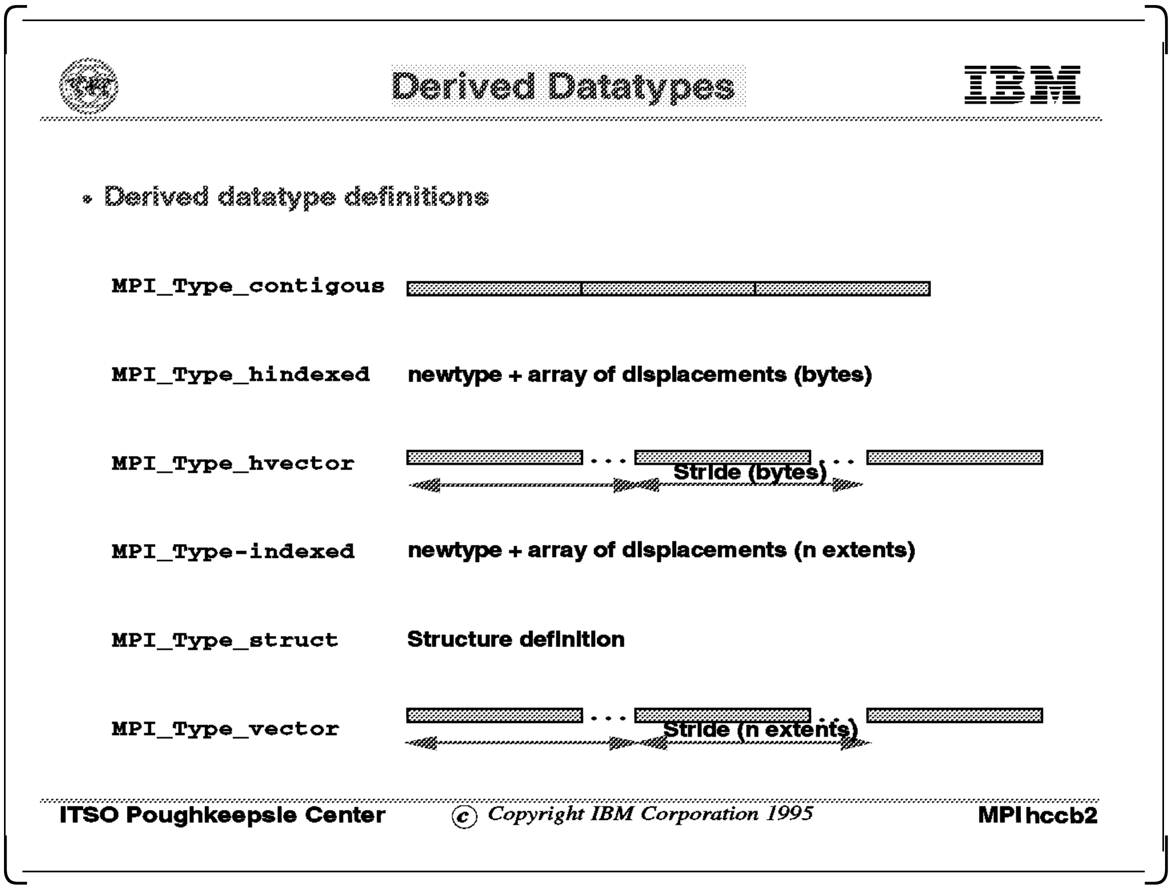
```
include 'mpif.h'
call MPI_SEND(buf,icount,MPI_DOUBLE_PRECISION,idest,itag,icomm,ierror)
```

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* MPIhccb1

Send and receive subroutines generally include a datatype specification in their parameter list, which is used as unit for the buffer length determination. Predefined datatypes, which cover traditional data types available in C and Fortran, are included in *mpif.h* (Fortran version) or in *mpi.h* in C.

The predefined datatype names are self explanatory for Fortran and C programmers.

2.3.7 Derived Datatypes



Derived Datatypes

- Derived datatype definitions

MPI_Type_contigous	
MPI_Type_hindexed	newtype + array of displacements (bytes)
MPI_Type_hvector	Stride (bytes)
MPI_Type-indexed	newtype + array of displacements (n extents)
MPI_Type_struct	Structure definition
MPI_Type_vector	Stride (n extents)

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* MPI hccb2

The derived datatypes allow a user to define his own datatypes with the following subroutines:

MPI_Type_contiguous
You can create a new datatype that represents the concatenation of a specified count of oldtypes.

MPI_Type_hindexed
Elements are addressed by an array of displacements, which are measured in bytes. The length of each element, put in the block length array, is evaluated in terms of number of blocks.

This derived datatype is useful when the datatype elements have variable length and are erratically distributed. You have to create the displacement array and the length array first, and then specify these arrays in the MPI_Type_hindexed parameter list.

MPI_Type_hvector
Elements are separated by a specified number of bytes.

The new datatype represents a specified count of oldtypes (fixed length evaluated in number of blocks, fixed stride evaluated in bytes).

MPI_Type_indexed
Elements are separated by multiples of the input datatype extent.

Same as MPI_Type_hindexed, but the displacement unit is the oldtype extent.

MPI_Type_struct
This datatype is a structure of several datatypes specifying data of different types scattered in the memory.

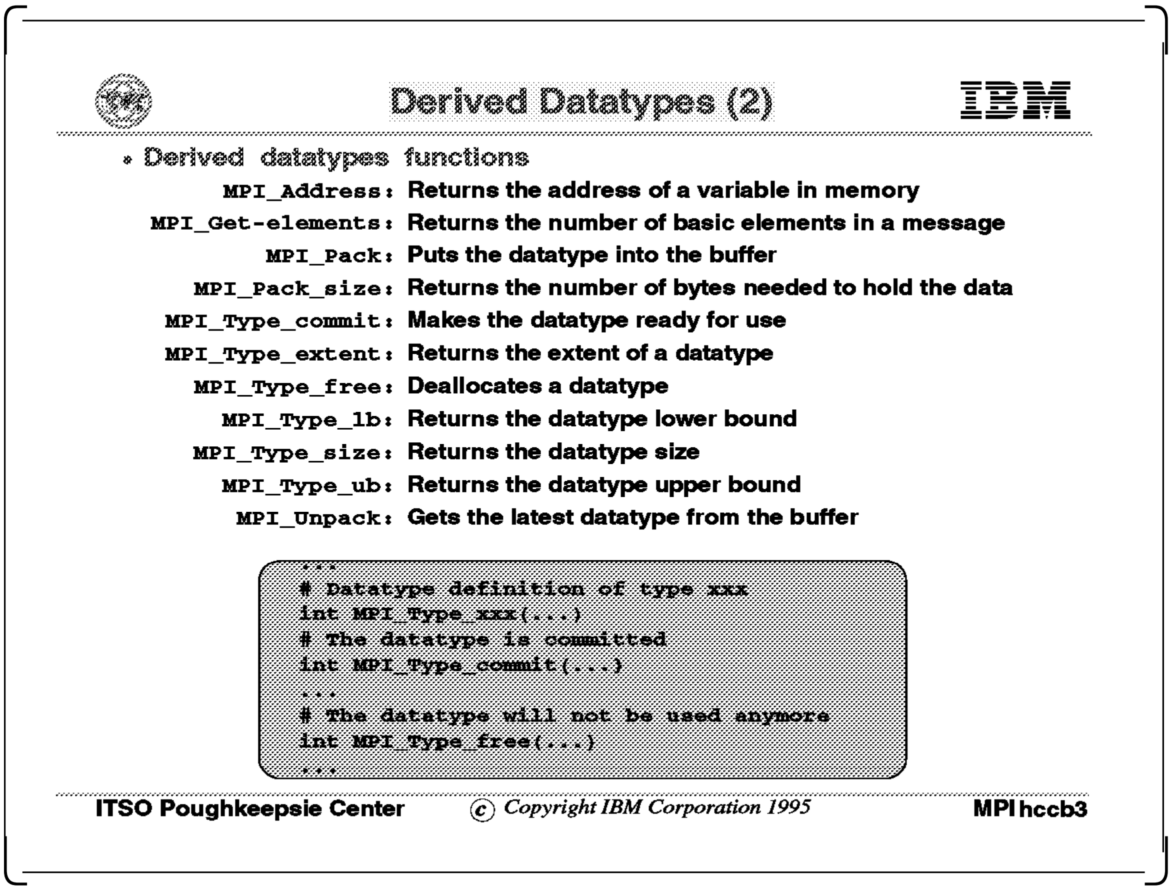
MPI_Type_vector
Same as MPI_Type_hvector, but the block length is a multiple of oldtype extent.

The stride is a multiple of the input datatype extent.

A datatype is made available by the MPI_Type_commit subroutine, and disabled by the MPI_Type_free subroutine.

When a datatype is defined, you can get information about it with the following subroutines:

- MPI_Type_extent gets the actual size of the datatype including the space lost due to alignment requirements.
- MPI_Type_lb gets the lower bound.
- MPI_Type_size gets the size of data in the datatype, excluding the padded areas due to alignment requirments.
- MPI_Type_ub gets the upper bound.



Derived Datatypes (2)

- Derived datatypes functions

MPI_Address:	Returns the address of a variable in memory
MPI_Get-elements:	Returns the number of basic elements in a message
MPI_Pack:	Puts the datatype into the buffer
MPI_Pack_size:	Returns the number of bytes needed to hold the data
MPI_Type_commit:	Makes the datatype ready for use
MPI_Type_extent:	Returns the extent of a datatype
MPI_Type_free:	Deallocates a datatype
MPI_Type_lb:	Returns the datatype lower bound
MPI_Type_size:	Returns the datatype size
MPI_Type_ub:	Returns the datatype upper bound
MPI_Unpack:	Gets the latest datatype from the buffer

```
...
# Datatype definition of type xxx
int MPI_Type_xxx(...)
# The datatype is committed
int MPI_Type_commit(...)
...
# The datatype will not be used anymore
int MPI_Type_free(...)
...
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPI hccb3

User Controlled Buffering

MPI_Pack and MPI_Unpack should not be confused with compress/uncompress programs generally available everywhere. Because the message passing performance is better when you send and receive large messages, you can program the communication this way:

- On the sender process, you define a buffer and you use MPI_Pack to fill out this work area with the messages you want to transmit.

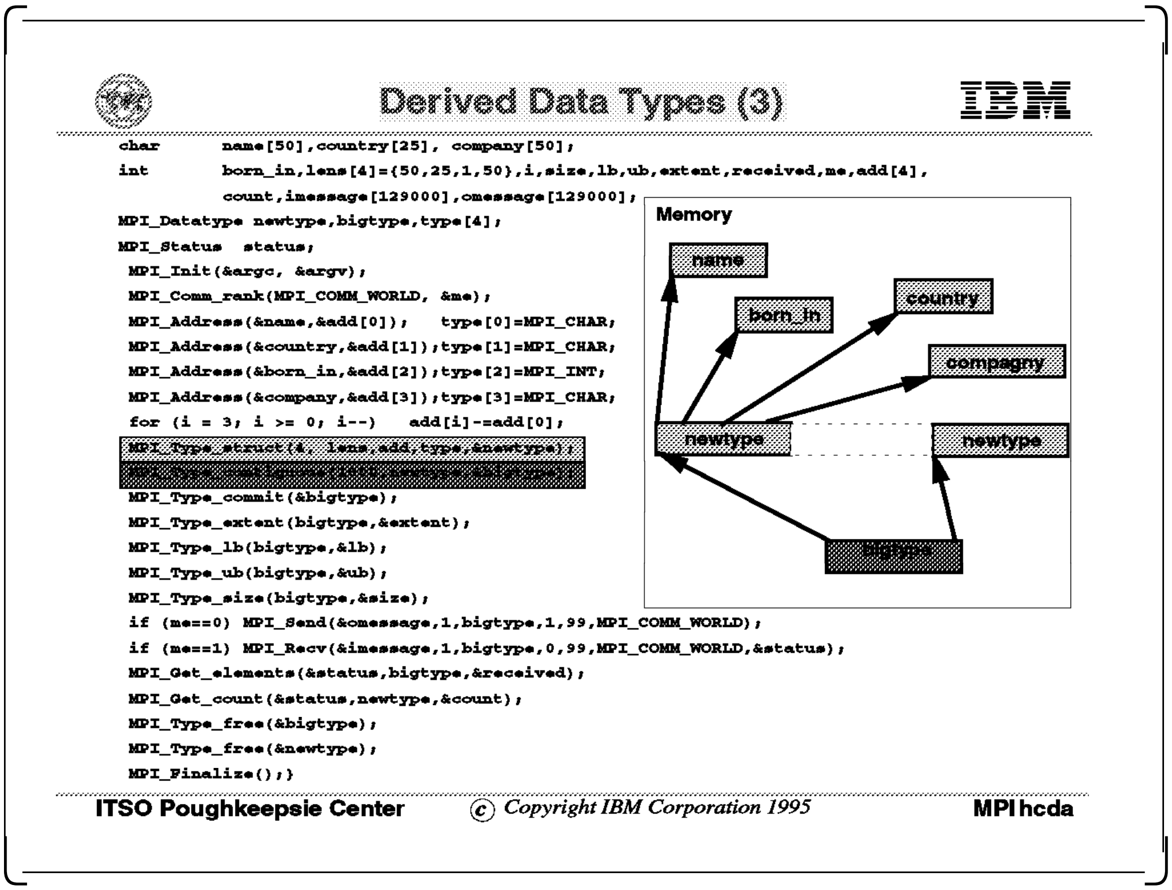
  Before the first MPI_Pack, the offset of the free space in the buffer is set up to zero, and MPI updates this offset after each MPI_Pack.

  Before each MPI_Pack, you have to use MPI_Pack_size that gives you the space required to pack your data into the buffer.

  When the buffer is full, you may send it with any MPI send subroutine, with the MPI_Packed datatype.

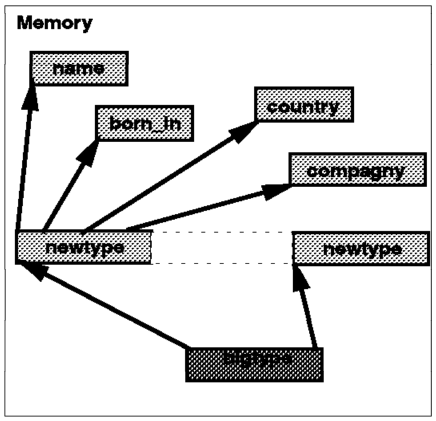
- On the receiver process, the MPI receive subroutine receives the buffer, and you may use MPI_Get_count to obtain the length of the transmitted buffer.

  You set up the position variable to zero, and each MPI_Unpack operation updates this variable to the next message starting point.



Derived Data Types (3)

```
char        name[50],country[25], company[50];
int         born_in,lens[4]={50,25,1,50},i,size,lb,ub,extent,received,me,add[4],
            count,imessage[129000],omessage[129000];
MPI_Datatype newtype,bigtype,type[4];
MPI_Status  status;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &me);
 MPI_Address(&name,&add[0]);    type[0]=MPI_CHAR;
 MPI_Address(&country,&add[1]);type[1]=MPI_CHAR;
 MPI_Address(&born_in,&add[2]);type[2]=MPI_INT;
 MPI_Address(&company,&add[3]);type[3]=MPI_CHAR;
 for (i = 3; i >= 0; i--)    add[i]-=add[0];
 MPI_Type_struct(4, lens,add,type,&newtype);
 MPI_Type_contiguous(1000,newtype,&bigtype);
 MPI_Type_commit(&bigtype);
 MPI_Type_extent(bigtype,&extent);
 MPI_Type_lb(bigtype,&lb);
 MPI_Type_ub(bigtype,&ub);
 MPI_Type_size(bigtype,&size);
 if (me==0) MPI_Send(&omessage,1,bigtype,1,99,MPI_COMM_WORLD);
 if (me==1) MPI_Recv(&imessage,1,bigtype,0,99,MPI_COMM_WORLD,&status);
 MPI_Get_elements(&status,bigtype,&received);
 MPI_Get_count(&status,newtype,&count);
 MPI_Type_free(&bigtype);
 MPI_Type_free(&newtype);
 MPI_Finalize();}
```



ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 MPI hcda

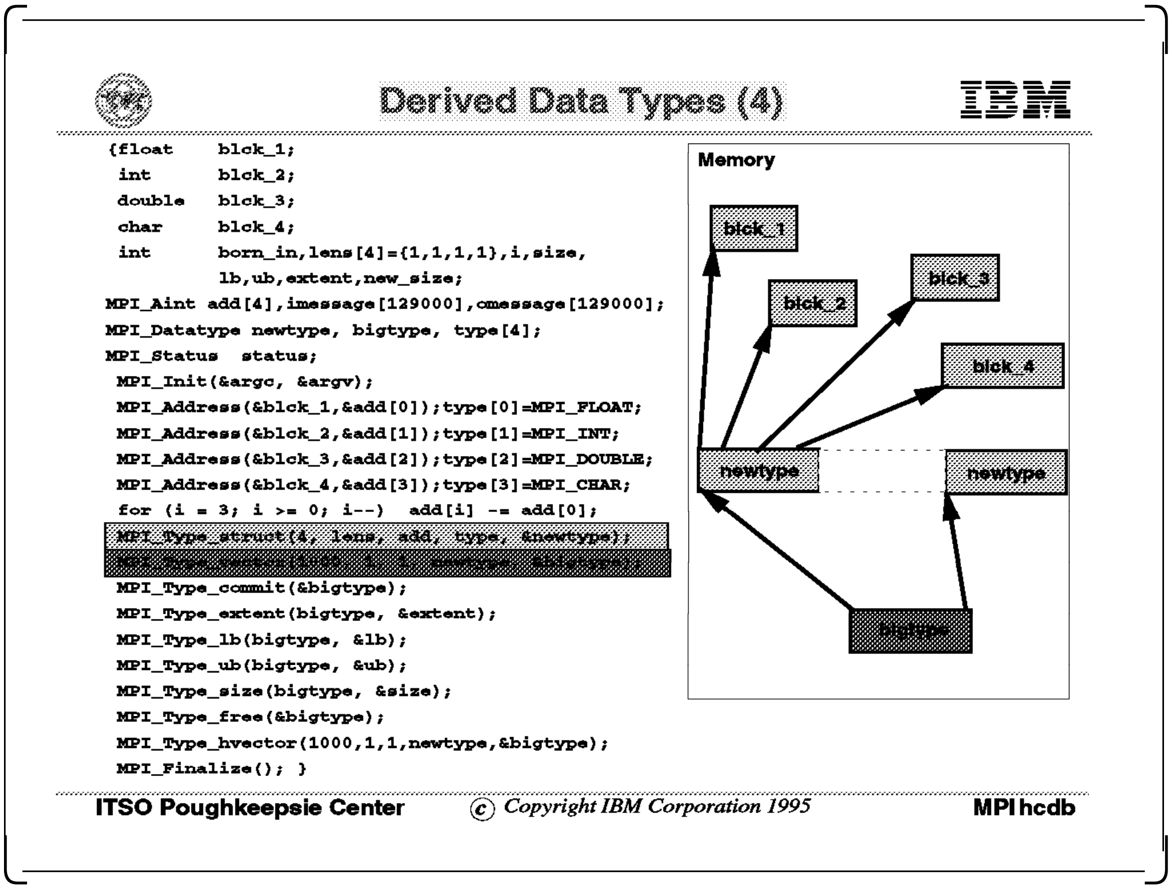
As already described, MPI provides predefined datatypes that fit with data types available in Fortran and C languages. Using these basic datatypes, you can create derived datatypes with MPI_Type-xxx subroutines.

This foil presents an example of MPI_Type_struct use that describes a complex structure of data spread in the memory, but referenced by a unique name in every MPI operation after that derived datatype has been acitvated by MPI_Type commit.

The following MPI routine calls are made to create a first datatype, named *newtype*. Then this newtype is concatenated in *bigtype* to obtain a large message.

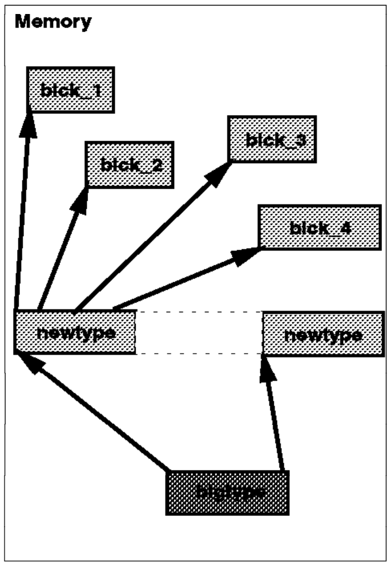
- MPI_Type_struct creates a datatype containing four basic datatypes: So the derived datatype *newtype* will be an array of datatypes and displacements in memory to find these datatypes. So that when you need to send this kind of datatype to another task, you just need to address it to get the contents of different parts of the memory.
- MPI_Type contiguous will take the previously created *newtype* as a base to create a *bigtype* of *1000* entries of type newtype.
- MPI_Type_commit will commit the defined datatype so that it can be used to send and receive messages.
- MPI_type extent is able to find the total extent of the datatype in memory. In the example extent will be equal to *250-80=170+padding*.
- MPI_Type_lb will give the lower bound *lb* of the datatype that must always be equal to *0*.
- MPI_Type_up gives the upper bound *ub* of the datatype that will be here *250-80=170*.

- The MPI_Type_size will give the real size *size* of the datatype, here *50+50+25+4=129*.



Derived Data Types (4)

```
 {float    blck_1;
  int      blck_2;
  double   blck_3;
  char     blck_4;
  int      born_in,lens[4]={1,1,1,1},i,size,
           lb,ub,extent,new_size;
MPI_Aint add[4],imessage[129000],omessage[129000];
MPI_Datatype newtype, bigtype, type[4];
MPI_Status  status;
 MPI_Init(&argc, &argv);
 MPI_Address(&blck_1,&add[0]);type[0]=MPI_FLOAT;
 MPI_Address(&blck_2,&add[1]);type[1]=MPI_INT;
 MPI_Address(&blck_3,&add[2]);type[2]=MPI_DOUBLE;
 MPI_Address(&blck_4,&add[3]);type[3]=MPI_CHAR;
 for (i = 3; i >= 0; i--)  add[i] -= add[0];
 MPI_Type_struct(4, lens, add, type, &newtype);
 MPI_Type_vector(1000, 1, 1, newtype, &bigtype);
 MPI_Type_commit(&bigtype);
 MPI_Type_extent(bigtype, &extent);
 MPI_Type_lb(bigtype, &lb);
 MPI_Type_ub(bigtype, &ub);
 MPI_Type_size(bigtype, &size);
 MPI_Type_free(&bigtype);
 MPI_Type_hvector(1000,1,1,newtype,&bigtype);
 MPI_Finalize(); }
```

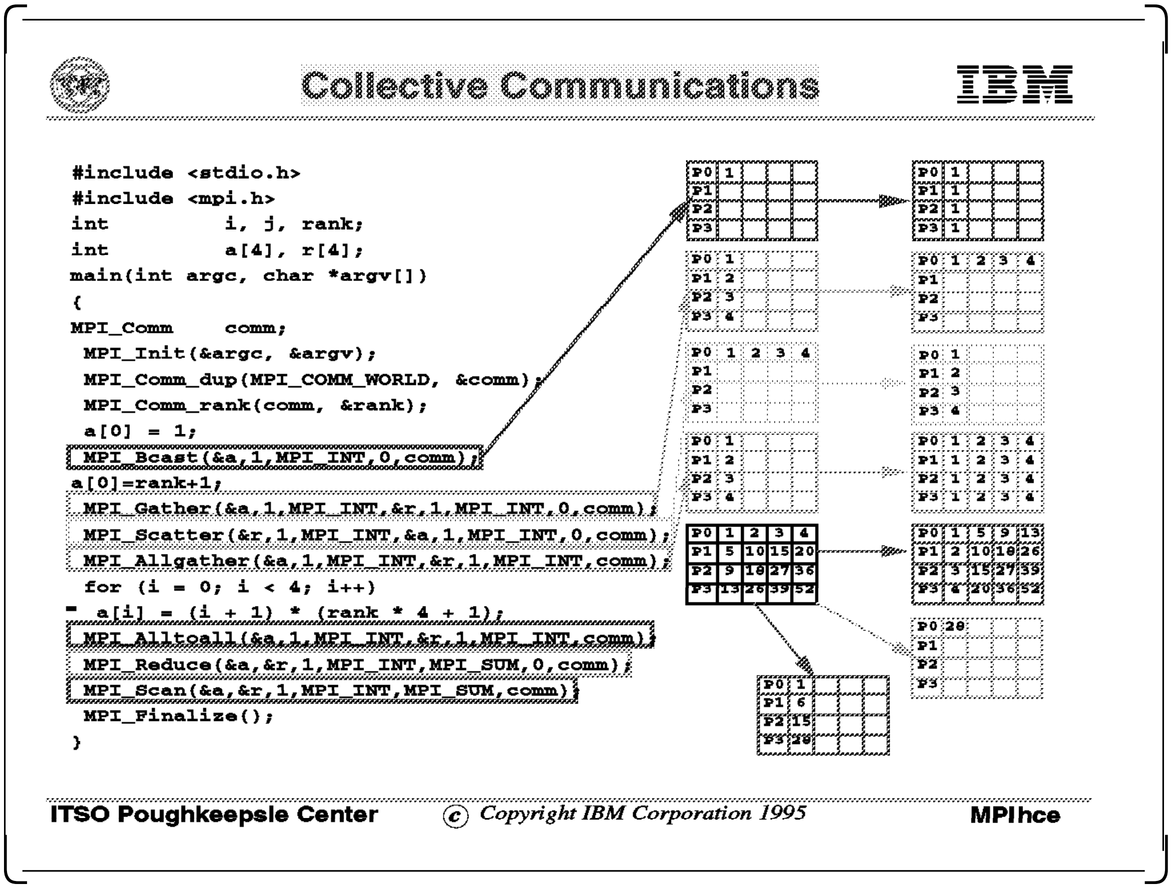


ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **MPI hcdb**

The behavior of the MPI_Type_vector is somewhat the same as the MPI_Type_contiguous except that the data in memory has to be equally spaced blocks. The space between blocks is defined by a quantity of basic datatype extents, here *1* so we will get all the entries of the array. For example we could have chosen a different spacing to take for example only even blocks in the basic datatype.

The MPI_Type_hvector can also be used if you need to define the spacing between blocks in byte units.

2.4 Collective Communications



```
#include <stdio.h>
#include <mpi.h>
int         i, j, rank;
int         a[4], r[4];
main(int argc, char *argv[])
{
MPI_Comm    comm;
 MPI_Init(&argc, &argv);
 MPI_Comm_dup(MPI_COMM_WORLD, &comm);
 MPI_Comm_rank(comm, &rank);
 a[0] = 1;
 MPI_Bcast(&a,1,MPI_INT,0,comm);
a[0]=rank+1;
 MPI_Gather(&a,1,MPI_INT,&r,1,MPI_INT,0,comm);
 MPI_Scatter(&r,1,MPI_INT,&a,1,MPI_INT,0,comm);
 MPI_Allgather(&a,1,MPI_INT,&r,1,MPI_INT,comm);
 for (i = 0; i < 4; i++)
-  a[i] = (i + 1) * (rank * 4 + 1);
 MPI_Alltoall(&a,1,MPI_INT,&r,1,MPI_INT,comm);
 MPI_Reduce(&a,&r,1,MPI_INT,MPI_SUM,0,comm);
 MPI_Scan(&a,&r,1,MPI_INT,MPI_SUM,comm);
 MPI_Finalize();
}
```

Collective communications are useful when the programmer needs to spread data from arrays over the network to one or more tasks.

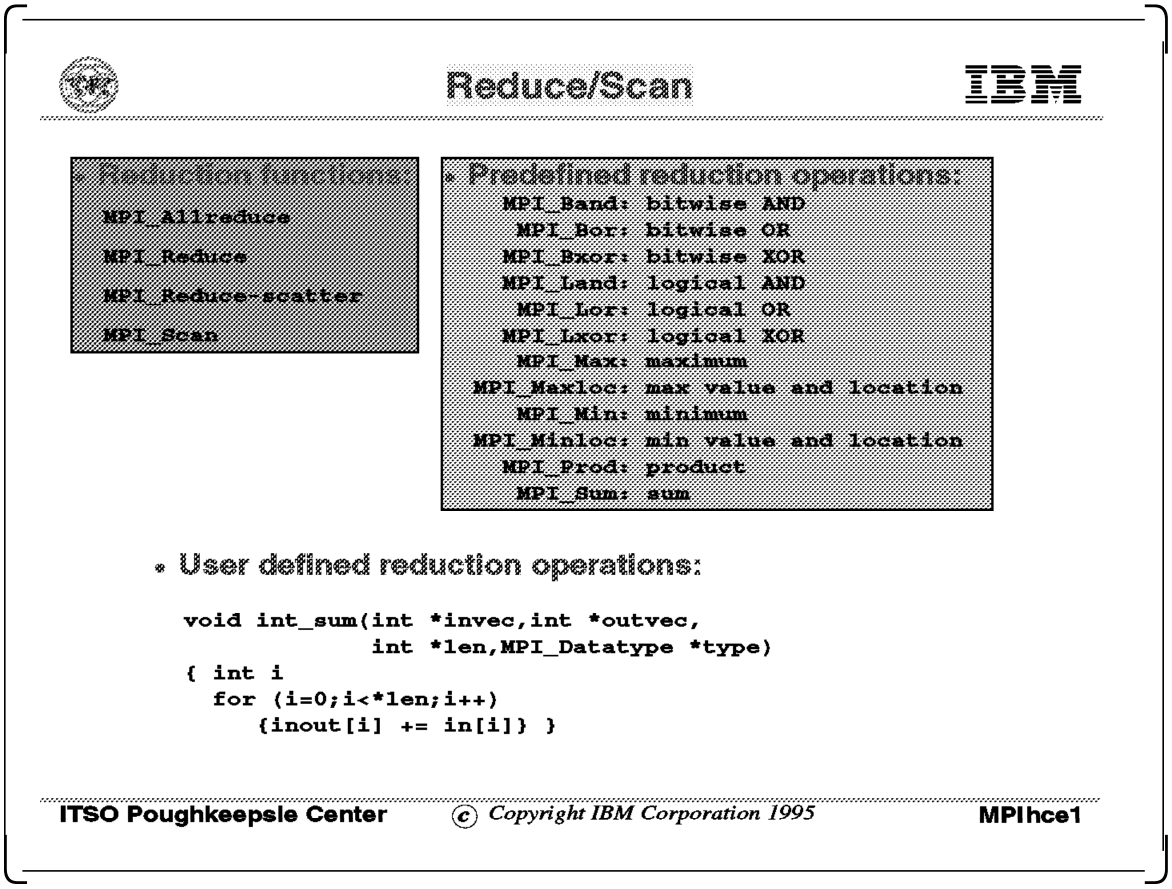
The example shows the most common collective communication functions:

- MPI_Bcast will broadcast the first element of the array *a* from the root process *0* to all others processes in the *comm* communicator.
- MPI_Gather will take each first element of the array *a* from process *J* to put it in the *Jth* element of the array *r* of the root process *0*.
- MPI_Scatter is the reverse function of the MPI_Gather. It means that it will take the *Jth* element of the root process to put it in the first element of process *J*.
- MPI_Allgather also has almost the same behavior as the MPI_Gather function except that the data is sent to each process, not only to the root process.
- MPI_Alltoall sends the *Ith* element of process *P* to the *Pth* element of process *I*.
- MPI_Reduce not only sends the data from one task to another task but also makes some reduction operation on the data before it is received on the receiving process. See Section 2.5, “Reduction Operations” on page 39 for a list of reduction operations and for information on how to create your own reduction operation. In the example, we make the *MPI_SUM* of the first

elements of the array *a* from each process and put the result in the first element of the array *r* in the root process *0*

- MPI_Scan will have the same behavior as the MPI_Reduce function except that intermediate results will be distributed to the process in the rank of the result:
  - On process 0 we will get in *r*[*0*] the value of *a*[*0*] of process 0.
  - On process 1 we will get in *r*[*0*] the value of *a*[*0*] of process 0 plus *a*[*0*] of process 1.
  - On process 2 we will get in *r*[*0*] the value of *a*[*0*] of process 0 plus *a*[*0*] of process 1 plus *a*[*0*] of process 2.
  - On process 3 we will get in *r*[*0*] the value of *a*[*0*] of process 0 plus *a*[*0*] of process 1 plus *a*[*0*] of process 2 plus *a*[*0*] of process 3.

2.5 Reduction Operations



Reduce/Scan

- Reduction functions:
  - MPI_Allreduce
  - MPI_Reduce
  - MPI_Reduce-scatter
  - MPI_Scan

- Predefined reduction operations:

MPI_Band:	bitwise AND
MPI_Bor:	bitwise OR
MPI_Bxor:	bitwise XOR
MPI_Land:	logical AND
MPI_Lor:	logical OR
MPI_Lxor:	logical XOR
MPI_Max:	maximum
MPI_Maxloc:	max value and location
MPI_Min:	minimum
MPI_Minloc:	min value and location
MPI_Prod:	product
MPI_Sum:	sum

- User defined reduction operations:

```
void int_sum(int *invec,int *outvec,
             int *len,MPI_Datatype *type)
{ int i
  for (i=0;i<*len;i++)
     {inout[i] += in[i]} }
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPIhce1

Common logical and arithmetical operations are available as operators for collective communications in MPI. Users can define their own reduction operation: they write a C function or a Fortran subroutine with four parameters. For instance:

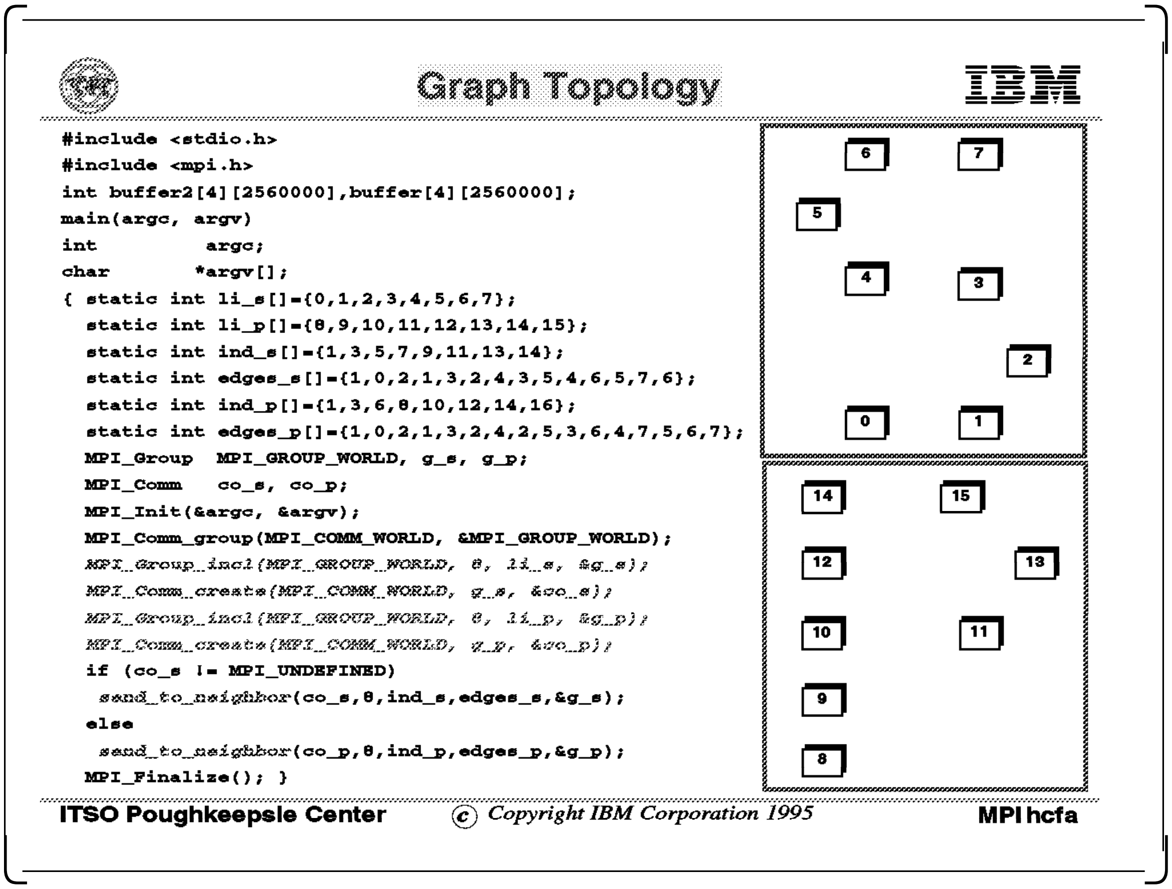
IN	Scalar or vector (dimension in third parameter)
OUT	Scalar or vector (same dimension)
LEN	IN and OUT dimension
TYPE	Datatype of IN and OUT

2.6 Topologies

In this chapter, we describe the processor topology. According to the application design, some data may have to be sent or received by a node from its neighbors. The developer must maintain the table of neighbors for each node involved in the parallel program. Now, MPI provides the developer with subroutines that create and maintain the node topology and the list of neighbors. With these subroutines, you can define either a graphical topology or a cartesian topology.

Section 2.6.1, "Graph Topology" presents the graphical topology.
Section 2.6.2, "Cartesian Topology" presents the cartesian topology.

2.6.1 Graph Topology



Graph Topology

```
#include <stdio.h>
#include <mpi.h>
int buffer2[4][2560000],buffer[4][2560000];
main(argc, argv)
int          argc;
char        *argv[];
{ static int li_s[]={0,1,2,3,4,5,6,7};
  static int li_p[]={8,9,10,11,12,13,14,15};
  static int ind_s[]={1,3,5,7,9,11,13,14};
  static int edges_s[]={1,0,2,1,3,2,4,3,5,4,6,5,7,6};
  static int ind_p[]={1,3,6,8,10,12,14,16};
  static int edges_p[]={1,0,2,1,3,2,4,2,5,3,6,4,7,5,6,7};
  MPI_Group  MPI_GROUP_WORLD, g_s, g_p;
  MPI_Comm   co_s, co_p;
  MPI_Init(&argc, &argv);
  MPI_Comm_group(MPI_COMM_WORLD, &MPI_GROUP_WORLD);
  MPI_Group_incl(MPI_GROUP_WORLD, 8, li_s, &g_s);
  MPI_Comm_create(MPI_COMM_WORLD, g_s, &co_s);
  MPI_Group_incl(MPI_GROUP_WORLD, 8, li_p, &g_p);
  MPI_Comm_create(MPI_COMM_WORLD, g_p, &co_p);
  if (co_s != MPI_UNDEFINED)
   send_to_neighbor(co_s,8,ind_s,edges_s,&g_s);
  else
   send_to_neighbor(co_p,8,ind_p,edges_p,&g_p);
  MPI_Finalize(); }
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPI hcfa

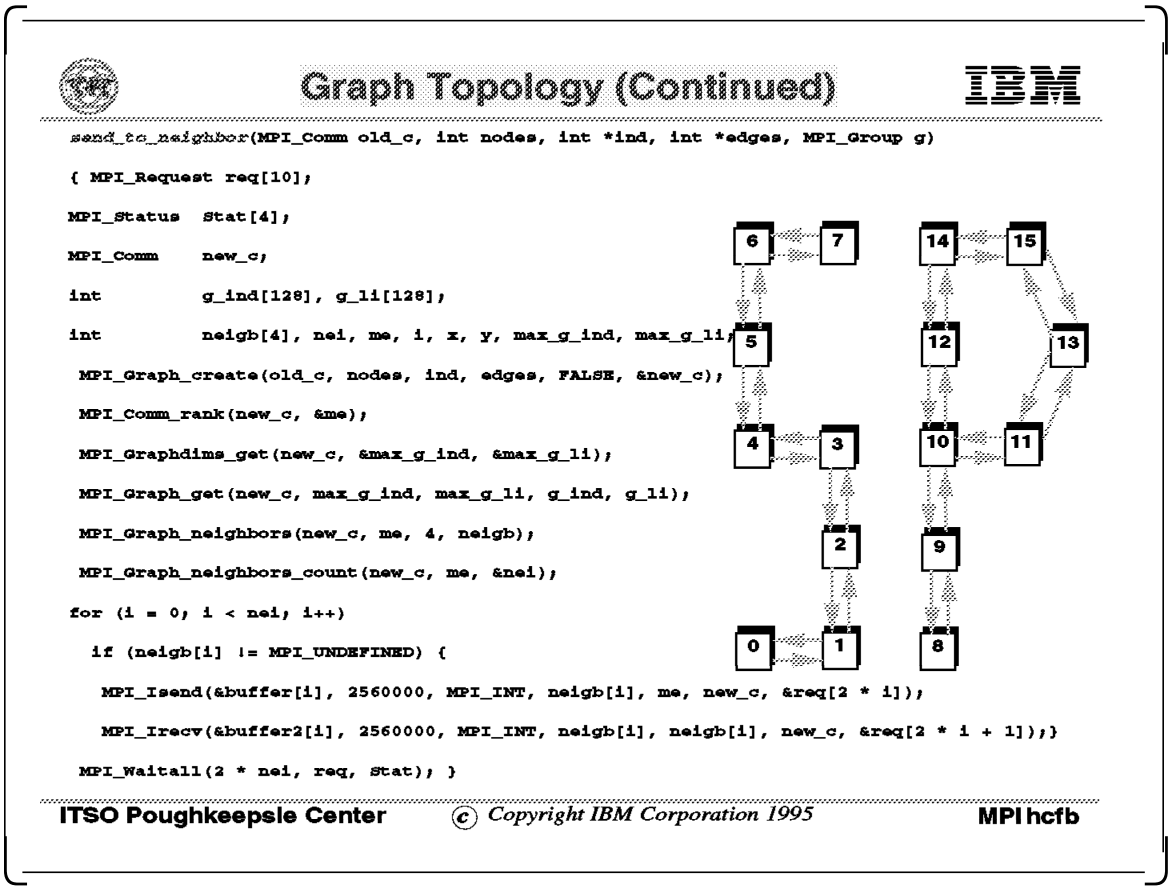
Topologies are used for ease in programming when constraints have to be exchanged between neighbors in the field of computation.

Graphical computation is used when the field of computation cannot be translated in a cartesian topology.

In the example, we first create two groups of nodes that will simulate two separate fields of computation. The *li_s* and *li_p* are the list of tasks included in *co_s* and *co_p* communicators.

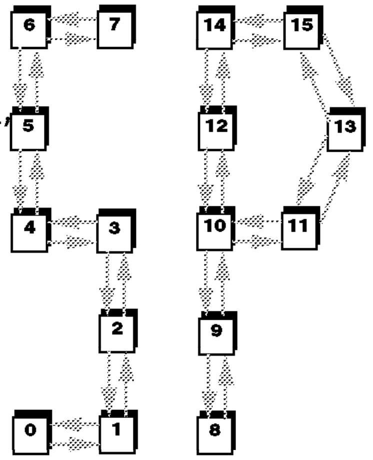
ind_s,ind_p,edges_s and *edges_p* will be used on the next page to construct the cartesian topology. See the following example on how to find the neighbor according to the indexes and the edges arrays.

```
neighbor[i]=edges[index[i-1]],...,edges[index[i]-1]
neighbor[5]=edges_s[ind_s[4],...,edges[index[5]-1]
neighbor[5]=edges_s[9],....edges[11-1]
neighbor[5]={4,6}
```



Graph Topology (Continued)

```
send_to_neighbor(MPI_Comm old_c, int nodes, int *ind, int *edges, MPI_Group g)
{ MPI_Request req[10];
MPI_Status  Stat[4];
MPI_Comm    new_c;
int         g_ind[128], g_li[128];
int         neigb[4], nei, me, i, x, y, max_g_ind, max_g_li;
 MPI_Graph_create(old_c, nodes, ind, edges, FALSE, &new_c);
 MPI_Comm_rank(new_c, &me);
 MPI_Graphdims_get(new_c, &max_g_ind, &max_g_li);
 MPI_Graph_get(new_c, max_g_ind, max_g_li, g_ind, g_li);
 MPI_Graph_neighbors(new_c, me, 4, neigb);
 MPI_Graph_neighbors_count(new_c, me, &nei);
for (i = 0; i < nei; i++)
  if (neigb[i] != MPI_UNDEFINED) {
   MPI_Isend(&buffer[i], 2560000, MPI_INT, neigb[i], me, new_c, &req[2 * i]);
   MPI_Irecv(&buffer2[i], 2560000, MPI_INT, neigb[i], neigb[i], new_c, &req[2 * i + 1]);}
 MPI_Waitall(2 * nei, req, Stat); }
```

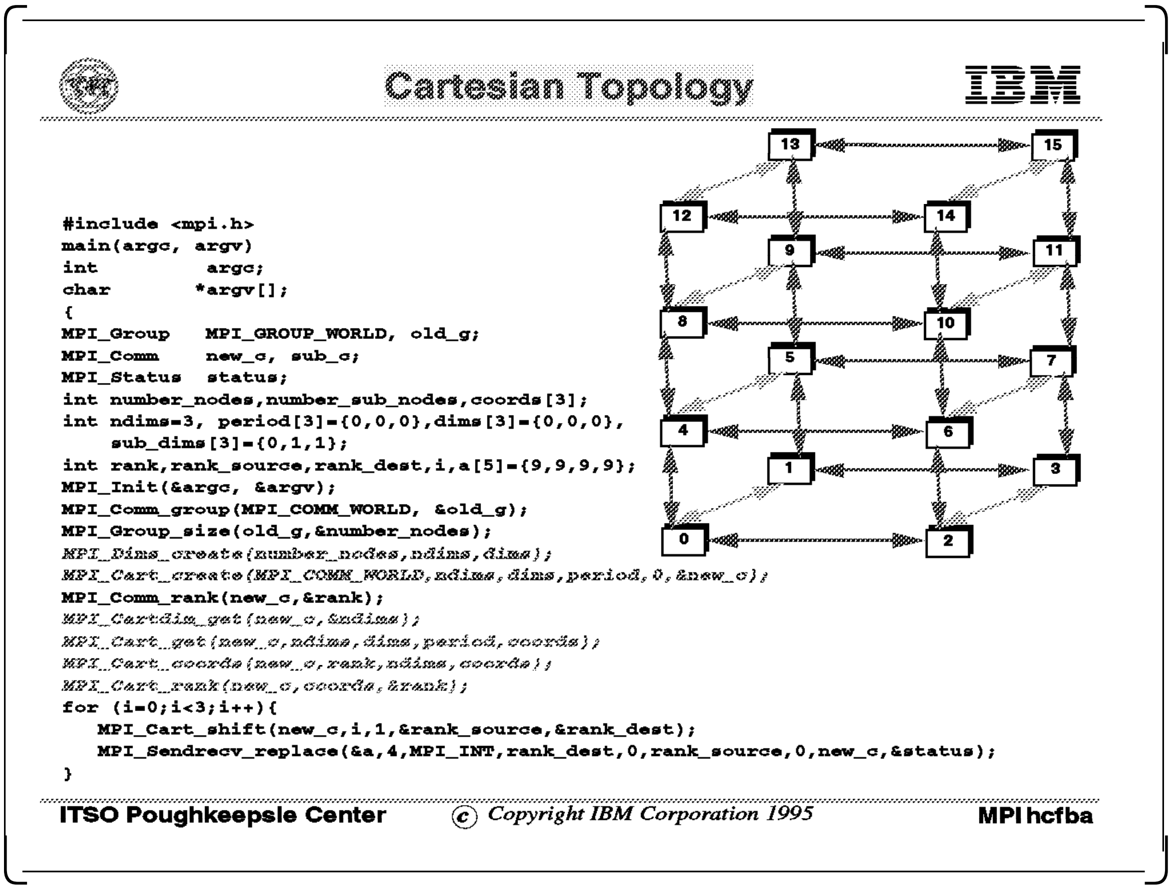


ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPI hcfb

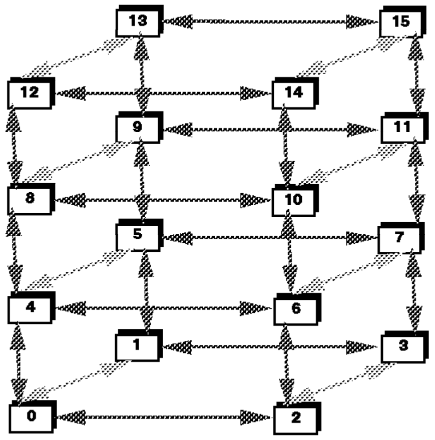
MPI_Graph_create creates the graphical topology from the communicator passed as an argument assuming the edges constraint.

MPI_Graph_neighbors helps the programmer to find easily the neighbors of each process in the topology so that he can send messages to them.

2.6.2 Cartesian Topology



Cartesian Topology



```
#include <mpi.h>
main(argc, argv)
int          argc;
char        *argv[];
{
MPI_Group   MPI_GROUP_WORLD, old_g;
MPI_Comm    new_c, sub_c;
MPI_Status  status;
int number_nodes,number_sub_nodes,coords[3];
int ndims=3, period[3]={0,0,0},dims[3]={0,0,0},
    sub_dims[3]={0,1,1};
int rank,rank_source,rank_dest,i,a[5]={9,9,9,9};
MPI_Init(&argc, &argv);
MPI_Comm_group(MPI_COMM_WORLD, &old_g);
MPI_Group_size(old_g,&number_nodes);
MPI_Dims_create(number_nodes,ndims,dims);
MPI_Cart_create(MPI_COMM_WORLD,ndims,dims,period,0,&new_c);
MPI_Comm_rank(new_c,&rank);
MPI_Cartdim_get(new_c,&ndims);
MPI_Cart_get(new_c,ndims,dims,period,coords);
MPI_Cart_coords(new_c,rank,ndims,coords);
MPI_Cart_rank(new_c,coords,&rank);
for (i=0;i<3;i++){
   MPI_Cart_shift(new_c,i,1,&rank_source,&rank_dest);
   MPI_Sendrecv_replace(&a,4,MPI_INT,rank_dest,0,rank_source,0,new_c,&status);
}
```

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* **MPI hcfba**

The cartesian topology is easier to manage than a graphical topology as you don′t need to define all the neighbors of each task. It takes into account the following parameters in the example:

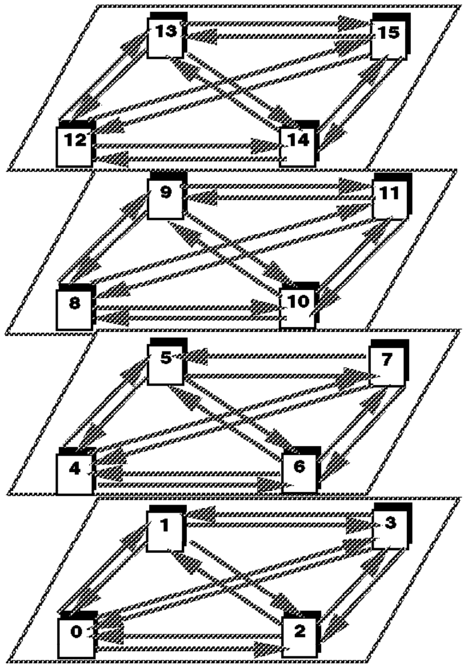
- *ndims* is the number of dimensions that the topology will have to manage, here *3*.
- *dims* is the number of tasks that will be in each dimension. Here we define them to *0* so that MPI_Dims_create will decide himself how many processes he must put in each dimension.
- *period* if set to true means that a process at the edge of the topology will have a neighbor in that dimension. His neighbor will be the process at the other edge of the topology in that dimension.

In the example, we use the following functions to manage the topology:

- MPI_Dims_create will compute the number of tasks to put in each dimension according to the number of dims you want, the number of processes you gave him for some dimensions, and the total number of processes you have. For example if you request 3 dimensions with *16* tasks and *dims={2,0,0}*, it will give you two tasks for the first dimension, four for the second and two for the last one.
- MPI_Cart_create will create the cartesian topology according to the previously defined *dims*, *ndims* and *period*.
- The MPI_Cart_shift helps the programmer to find the neighbors of a process in the specified dimension.

Cartesian Topology (Continued)

```
MPI_Cart_sub(new_c,sub_dims,&sub_c);
MPI_Cart_get(sub_c,ndims,sub_dims,
             period,coords);
MPI_Comm_rank(sub_c,&rank);
a[rank]=rank;
MPI_Comm_size(sub_c,&number_sub_nodes);
for (i=0;i<number_sub_nodes;i++)
    MPI_Bcast(&a[i],1,MPI_INT,i,sub_c);
MPI_Finalize();
}
```



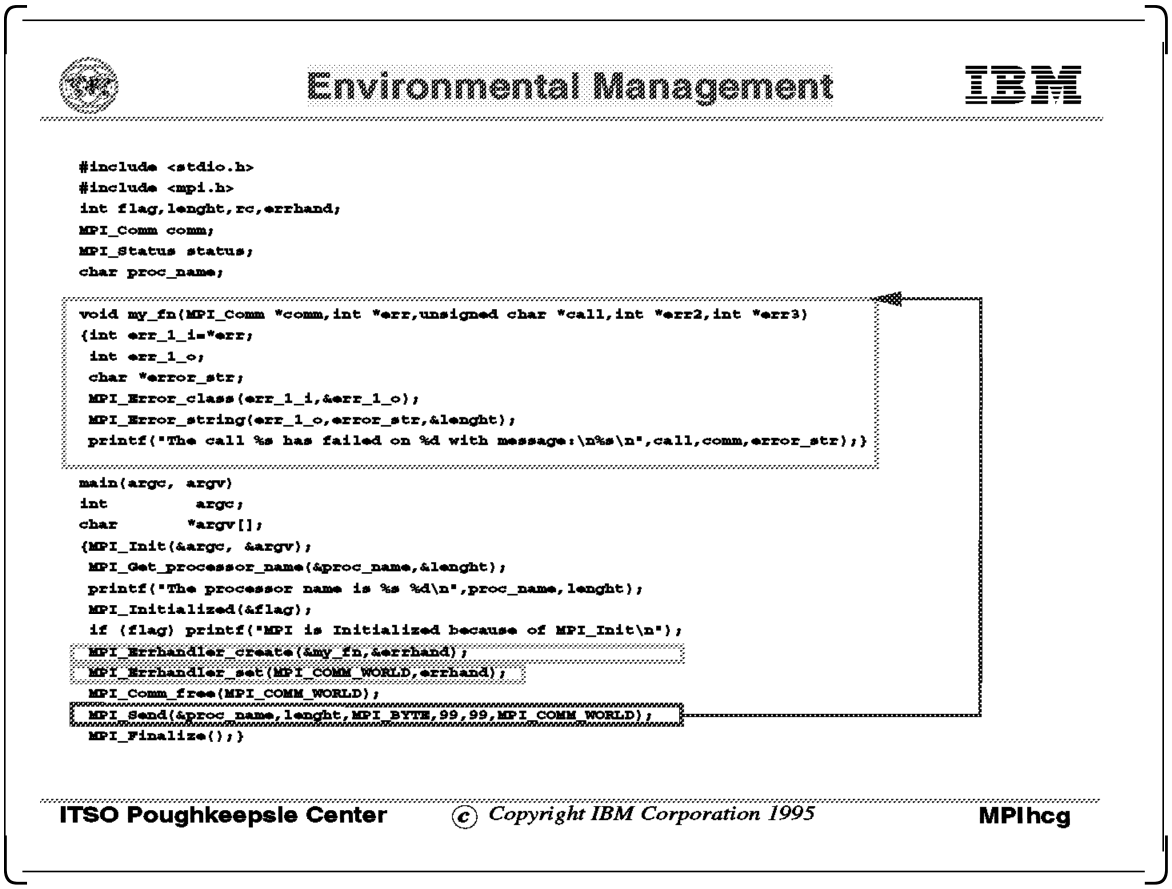
ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 MPI hcfbb

The cartesian topology can also easily split into sub-topologies of one or more dimensions of the mother.

In the example, the MPI_Cart_sub will split the topology according to the *sub_dims* specifications. Here *sub_dims* contains *{0,1,1}* which means that we keep the two last dimensions to create each of the sub-topologies.

So the MPI_Bcast will use these two dimensions topologies to send the broadcast.

2.6.3 Environmental Management



Environmental Management

```
#include <stdio.h>
#include <mpi.h>
int flag,lenght,rc,errhand;
MPI_Comm comm;
MPI_Status status;
char proc_name;

void my_fn(MPI_Comm *comm,int *err,unsigned char *call,int *err2,int *err3)
{int err_1_i=*err;
 int err_1_o;
 char *error_str;
 MPI_Error_class(err_1_i,&err_1_o);
 MPI_Error_string(err_1_o,error_str,&lenght);
 printf("The call %s has failed on %d with message:\n%s\n",call,comm,error_str);}

main(argc, argv)
int          argc;
char        *argv[];
{MPI_Init(&argc, &argv);
 MPI_Get_processor_name(&proc_name,&lenght);
 printf("The processor name is %s %d\n",proc_name,lenght);
 MPI_Initialized(&flag);
 if (flag) printf("MPI is Initialized because of MPI_Init\n");
 MPI_Errhandler_create(&my_fn,&errhand);
 MPI_Errhandler_set(MPI_COMM_WORLD,errhand);
 MPI_Comm_free(MPI_COMM_WORLD);
 MPI_Send(&proc_name,lenght,MPI_BYTE,99,99,MPI_COMM_WORLD);
 MPI_Finalize();}
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   MPIhcg

By default, the behavior of MPI is to exit if any error occurs. It can be avoid by setting the appropriate environment management.

In the example, we use the following functions to manage possible errors:

- MPI_Errhandler_create associates the *my_fn* error handling function with the *errhand* handler.
- MPI_Errhandler_set makes the error handling to *my_fn* effective as soon as this call exits.
- In *my_fn* we use MPI_Error_class to find the class of the error passed to *my_fn* as the *err* argument.

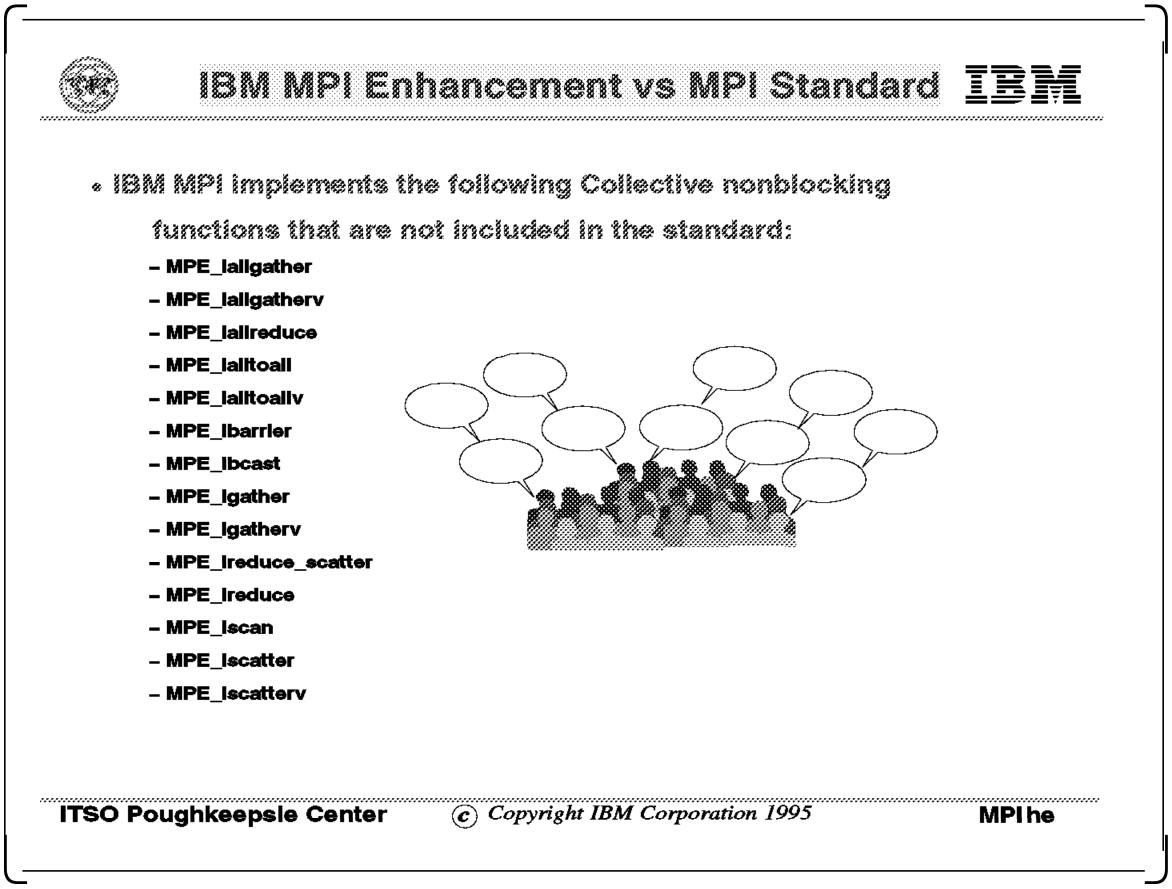
  We use the error_class given by MPI_Error_class in the MPI_Error_string parameter list to get the error message associated with that error. Then, we print the message
- In the main procedure we create an error condition: an MPI_send routine is executed after the communicator is freed. So, the *my_fn* function will be executed.

Valid error classes are:

MPI_SUCCESS	No error
MPI_ERR_BUFFER	Invalid buffer pointer
MPI_ERR_COUNT	Invalid count argument

MPI_ERR_TYPE	Invalid datatype
MPI_ERR_TAG	Invalid tag argument
MPI_ERR_COMM	Invalid communicator
MPI_ERR_RANK	Invalid rank
MPI_ERR_REQUEST	Invalid request handle
MPI_ERR_ROOT	Invalid root
MPI_ERR_GROUP	Invalid group
MPI_ERR_OP	Invalid operation
MPI_ERR_TOPOLOGY	Invalid topology
MPI_ERR_DIMS	Invalid dimension argument
MPI_ERR_ARG	Invalid argument
MPI_ERR_UNKNOWN	Unknown error
MPI_ERR_TRUNCATE	Message truncated on receive
MPI_ERR_OTHER	Known error not provided
MPI_ERR_INTERM	Internal MPI error

2.7 IBM MPI Enhancement vs MPI Standard



IBM MPI Enhancement vs MPI Standard

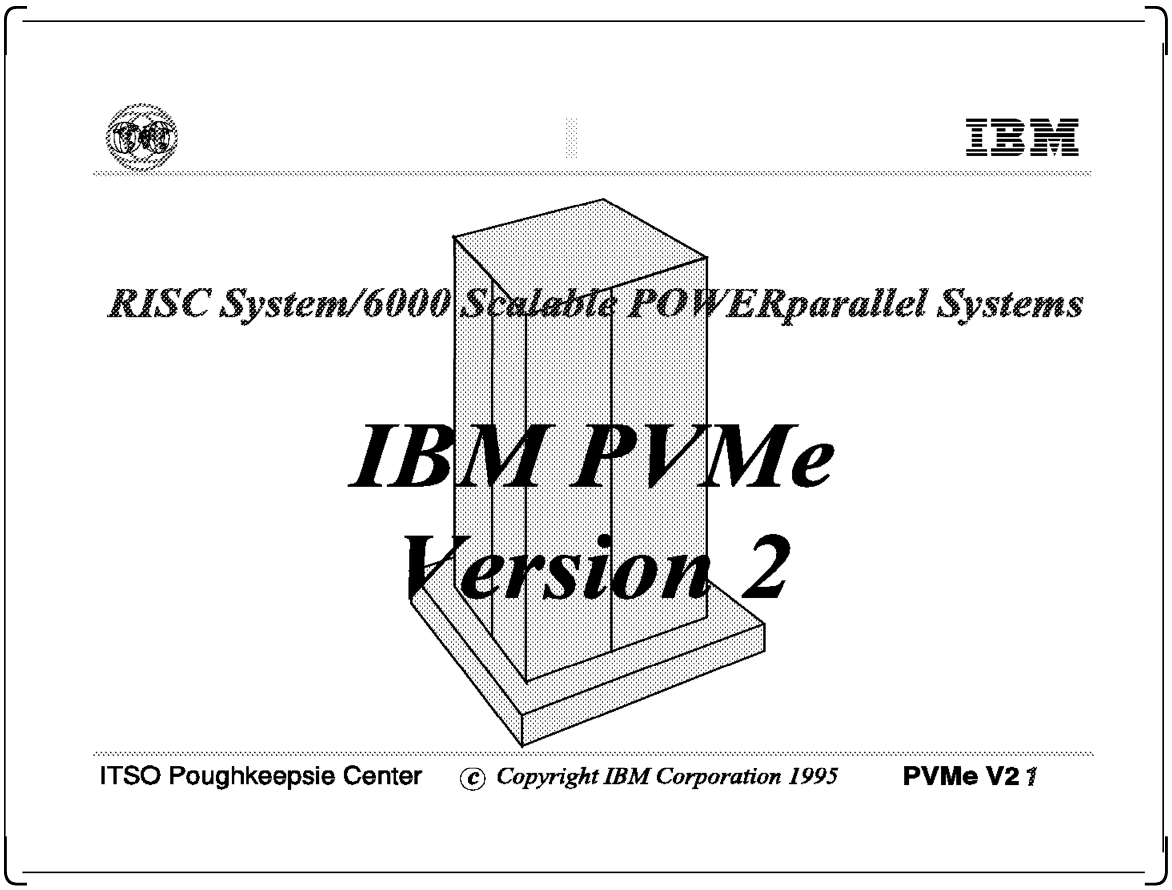
- IBM MPI implements the following Collective nonblocking functions that are not included in the standard:
  - MPE_Iallgather
  - MPE_Iallgatherv
  - MPE_Iallreduce
  - MPE_Ialltoall
  - MPE_Ialltoallv
  - MPE_Ibarrier
  - MPE_Ibcast
  - MPE_Igather
  - MPE_Igatherv
  - MPE_Ireduce_scatter
  - MPE_Ireduce
  - MPE_Iscan
  - MPE_Iscatter
  - MPE_Iscatterv

ITSO Poughkeepsie Center (c) *Copyright IBM Corporation 1995* MPIhe

IBM provides 14 more functions than the Standard MPI. It is the responsibility of the customer to decide to use them or not to make their applications portable in other MPI environments.

These functions are nonblocking collective functions. Their behavior is the same as the blocking ones described in 2.4, "Collective Communications" on page 37, except that they are nonblocking and must be managed using MPI_Waitxxx or MPI_Testxxx functions.

Chapter 3. PVMe V2



RISC System/6000 Scalable POWERparallel Systems

IBM PVMe Version 2

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 1

Parallel virtual machine (PVM) is currently the most wide-spread parallel environment. Designed and developed by Oak Ridge National Laboratory, in cooperation with Emory University, University of Tennessee, Carnegie Mellon University and Pittsburgh SuperComputer Center, under Department of Energy, National Science Foundation, and State of Tennessee Research grants. PVM is available as free distribution copyrighted software. It provides a message passing library that uses the IP protocol to communicate between the parallel program processes. PVM also provides the functions needed to execute a parallel program, and a post mortem monitor, *xpvm*, which uses the trace file generated during the program execution.

The xpvm monitor simulates the program execution and helps developers debug and tune their parallel program. PVM is available on many hardware platforms under UNIX systems, and allows parallel program executions on heterogeneous environments.

PVMe was developed by European Center for Scientific and Engineering Computing (IBM ECSEC, Rome Italy) to be used on RS/6000 SP systems with two objectives:

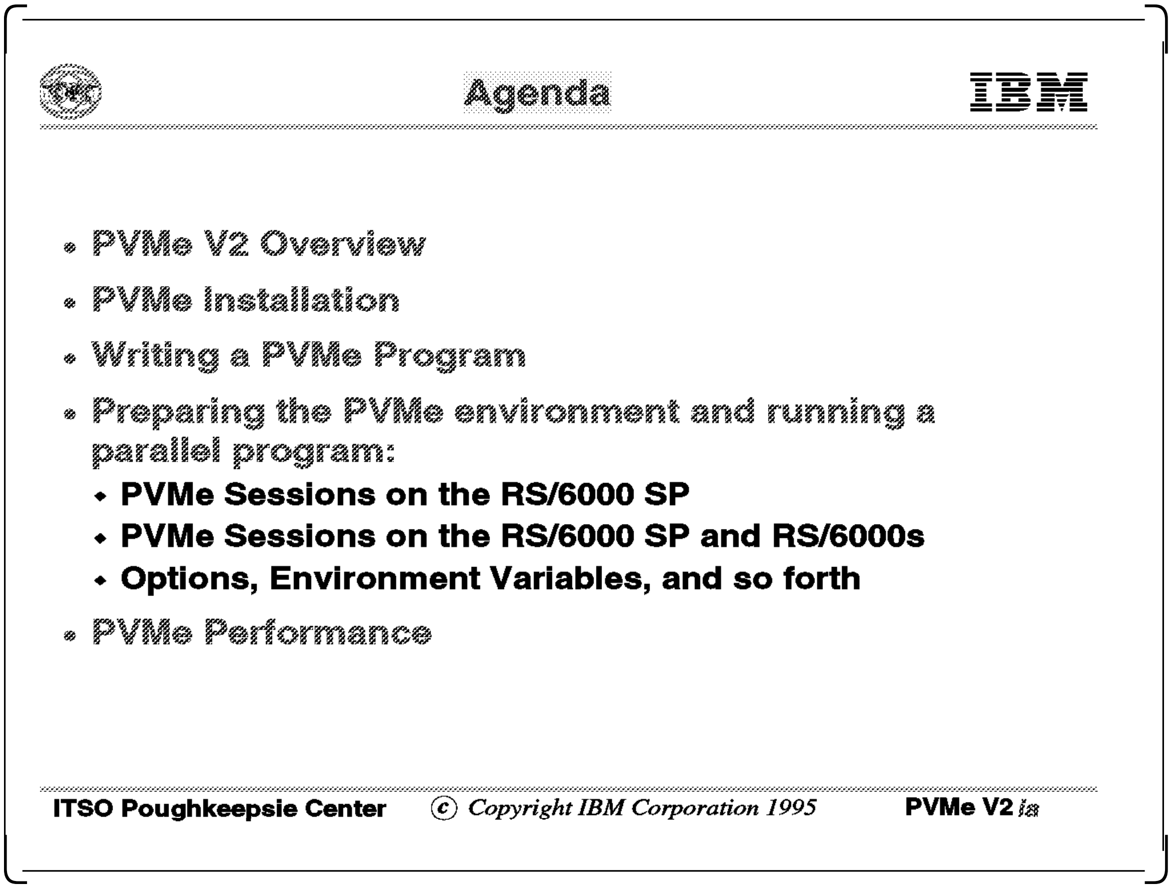
- External compatibility with the public domain PVM

- Written to optimize the performance of parallel programs when executed on RS/6000 SP equipped with the HPS in user space mode

PVMe Version 1 Release 1 was externally compatible with PVM 2.3, and PVMe Version 1 Release 2 and 3 were compatible with PVM 3.2. PVMe Version 2 Release 1 is compatible with PVM 3.3.7 and includes new facilities. This chapter is a presentation of PVMe Version 2 Release 1, organized as follows:

- Section 3.2, "PVMe Overview" on page 52 presents an overview of PVMe Version 2 features.
- Section 3.3, "PVMe 2.1 Installation" on page 66 provides information about the PVMe installation.
- Section 3.4, "Writing a PVMe Program" on page 67 presents examples about parallel program development.
- Section 3.4, "Writing a PVMe Program" on page 67 describes the way parallel programs are executed in the PVMe environment.

3.1 Covered Topics



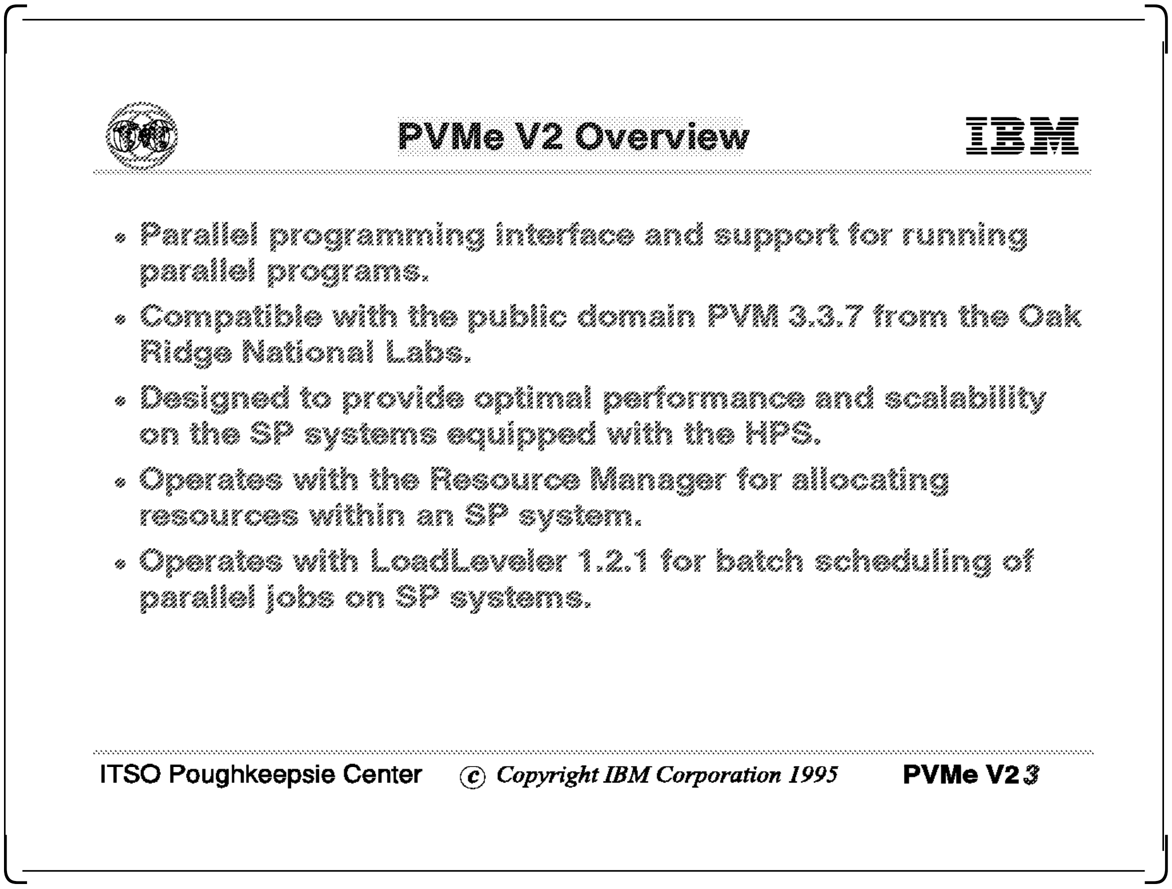
Agenda

- PVMe V2 Overview
- PVMe Installation
- Writing a PVMe Program
- Preparing the PVMe environment and running a parallel program:
  - PVMe Sessions on the RS/6000 SP
  - PVMe Sessions on the RS/6000 SP and RS/6000s
  - Options, Environment Variables, and so forth
- PVMe Performance

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PVMe V2

This presentation provides an overview of the PVMe V2 product for the RS/6000 SP systems and a description of the new features available with this version, as well as some guidelines to help users run parallel programs in this environment. Also the differences and the common features with the public domain PVM will be discussed, since many applications written to run on the PVM platform can be ported to the PVMe platform with little effort and can experience improved performance.

3.2 PVMe Overview



PVMe V2 Overview

- Parallel programming interface and support for running parallel programs.
- Compatible with the public domain PVM 3.3.7 from the Oak Ridge National Labs.
- Designed to provide optimal performance and scalability on the SP systems equipped with the HPS.
- Operates with the Resource Manager for allocating resources within an SP system.
- Operates with LoadLeveler 1.2.1 for batch scheduling of parallel jobs on SP systems.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V2** *3*

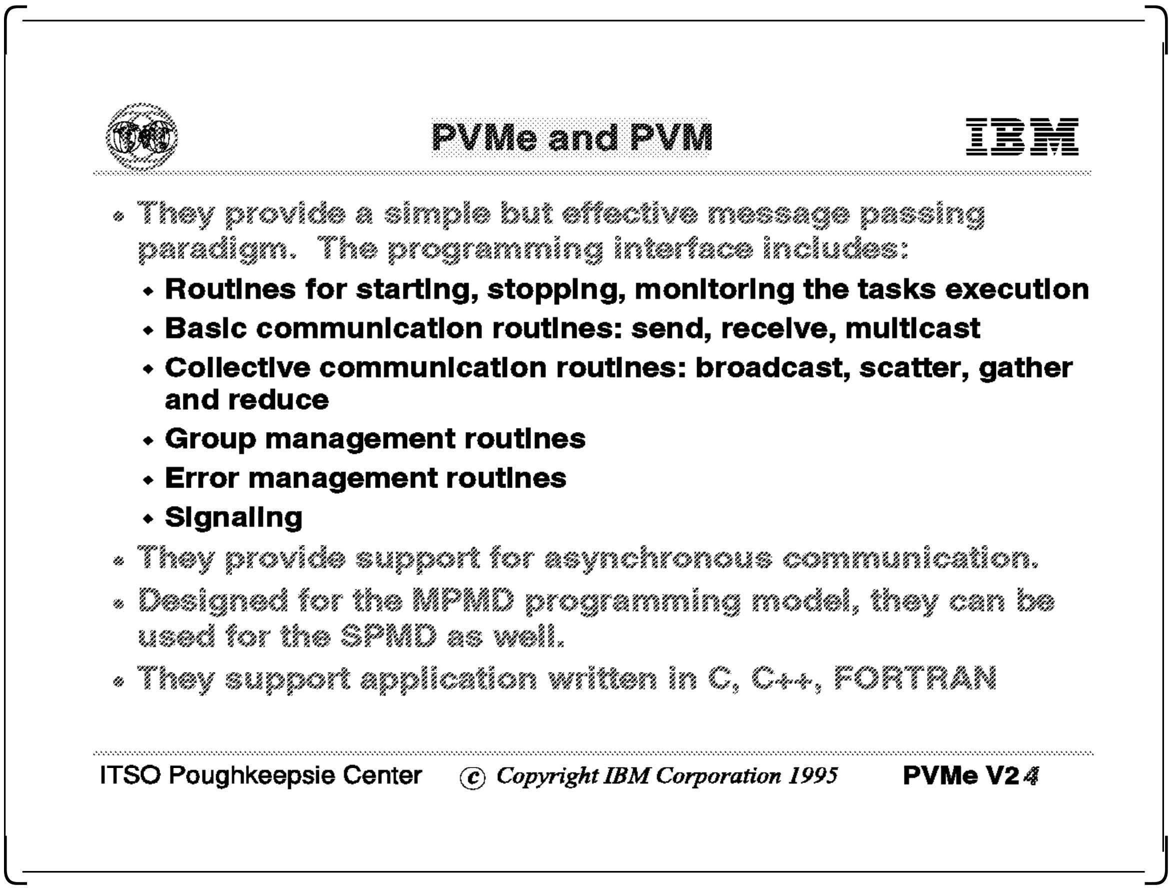
- PVMe provides a parallel programming interface (libraries of routines used to write a parallel programs) and a support software that allows you to run a parallel program on a set of hosts.
- PVMe is compatible with the public domain PVM 3.3.7. It offers the same programming interface (same name of the routines, same syntax, and so on) so that an existing PVM application only needs to be linked to the PVMe libraries to be executed in the PVMe environment.
- PVM is a public domain software designed to allow processes running on different machines (possibly running different operating systems) to cooperate to solve a given problem: to break a program into logical parts (as independent as possible) and run those parts simultaneously on multiple processors. This reduces the time needed to run the program.
  PVM is widely used, mainly in the academic environment, to the extent that it can be considered a *de facto* standard for parallel and distributed programming.

  While PVM supports multiple architectures and operating systems, PVMe has been specifically designed to run on an homogeneous environment and to take advantage as much as possible of the AIX operating system features and of the HPS communication hardware and software available for the SP systems.

Note: An implementation of PVMe is also available (although it is not a licensed program product) for RS/6000 clusters connected through networks, such as FDDI, SOCC, and Allnode Switch.

- It interfaces with the Resource Manager (RM), which is responsible for resource (CPU and communication adapter) allocation on the SP.
- It also interfaces with LoadLeveler to allow batch submissions of parallel jobs. In that case, LoadLeveler is responsible for requesting the resources to the RM, and if they are available, for starting the parallel job. If the needed resources are not available, LoadLeveler keeps the job in the queue for later attempts.

  Note: Given the current policy LoadLeveler uses to request/allocate resources from the RM, it can be unsafe to configure *general members* subpools within the RM pools; it is advisable, instead, to have subpools only devoted to interactive activity and subpools only devoted to batch activity.



PVMe and PVM

- They provide a simple but effective message passing paradigm. The programming interface includes:
  - Routines for starting, stopping, monitoring the tasks execution
  - Basic communication routines: send, receive, multicast
  - Collective communication routines: broadcast, scatter, gather and reduce
  - Group management routines
  - Error management routines
  - Signaling
- They provide support for asynchronous communication.
- Designed for the MPMD programming model, they can be used for the SPMD as well.
- They support application written in C, C++, FORTRAN

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V24**

3.2.1 PVMe and PVM

The parallel programming interface is quite simple, but it allows almost any of the actions commonly requested by parallel application programmers to be executed.

- The programming paradigm is quite different from that used, for example, in the Parallel Environment: you can manually start multiple processes (they can be either instances of the same program or of different programs) and make them to communicate through the PVMe routines, or you can start one process, and then this process requests that new processes be started remotely. You have routines that allow starting and stopping other tasks, and also to verify, at any time during the program execution, the number and the status of any other tasks.
- The basic communication routines allow you to exchange data among processes. Auxiliary routines for packing/unpacking data to/from a single buffer have been designed to minimize the number of calls to the data exchange routines.
- Collective communication routines allow more sophisticated actions, such as receiving data from multiple processes and combining all the data in one process using a specific function.
- You also have the possibility of grouping a subset of processes, typically processes that play the same role within the parallel program, so that the collective communication routines will only involve processes within the same group.

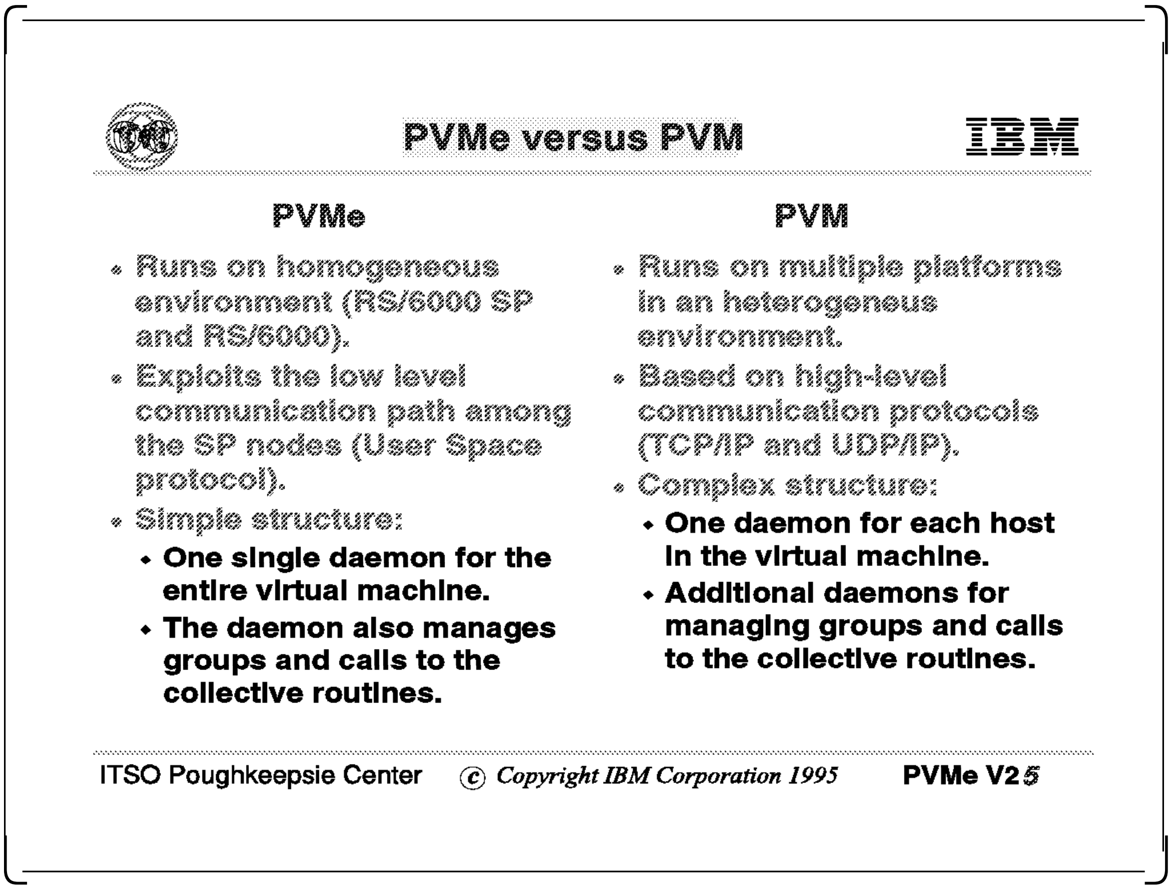
- Additional routines for sending signals and managing error conditions.

Both PVM and PVMe allow asynchronous communication: one task can start to send a message even though the recipient task is not ready to receive it; under normal conditions, moreover, the sender is able to complete the call and to continue with the processing in case the recipient doesn′t receive the message (or before the recipient receives it). In this way, the tasks are loosely synchronized, and they don′t block unless they need data that someone else has not yet sent to them.

PVM (and PVMe) was basically designed for the MPMD model, where different programs are executed by different task; this model is really convenient when the application is suited to a functional decomposition (possibly in addition to data decomposition). The master-slave approach (one master program starts n instances of slave programs, distributes initial data, subdomain boundaries, and so forth, collects intermediate results, coordinates the slaves, waits for final results, and saves them) is an example.

The design does not prevent the two packages from being used for the SPMD programming (the manually started task is responsible for requesting that n-1 more processes be started, where n is the size of the parallel run).

3.2.2 PVMe versus PVM



PVMe versus PVM

PVMe

- Runs on homogeneous environment (RS/6000 SP and RS/6000).
- Exploits the low level communication path among the SP nodes (User Space protocol).
- Simple structure:
  - **One single daemon for the entire virtual machine.**
  - **The daemon also manages groups and calls to the collective routines.**

PVM

- Runs on multiple platforms in an heterogeneus environment.
- Based on high-level communication protocols (TCP/IP and UDP/IP).
- Complex structure:
  - **One daemon for each host in the virtual machine.**
  - **Additional daemons for managing groups and calls to the collective routines.**

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V25**

- PVM is available on multiple platforms, and different tasks in a parallel program can also run on machines with different architectures. This is possible because the communication among tasks and among the service processes (daemons) is based on the *de facto* standard protocols TCP and UDP. The choice of implementing PVMe on an homogeneous environment made possible a simpler and more efficient implementation, and on the other hand, allowed using a specialized communication protocol tuned for the specific platform.
- In PVM, the task-to-task communication can occur over two paths:
  - From the sender process to the local daemon, then to the remote daemon, and finally to the recipient task through UDP and TCP sockets
  - Directly from the sender to receiver through a TCP socket

  In both cases, the high level protocol can become a bottleneck for communication, even though the underlying hardware connection is fast.
  In PVMe, data communication among the tasks exploits, by default, the User Space protocol over the HPS. The communication routines directly interface with the MPCI software bypassing the IP, and the user tasks directly send the message. Communication between the tasks and the service process (daemon) still uses the IP protocol (over the SP Ethernet), but since that traffic is expected to be occasional and the amount of transmitted data is very small, it does not affect the performance of the parallel application.

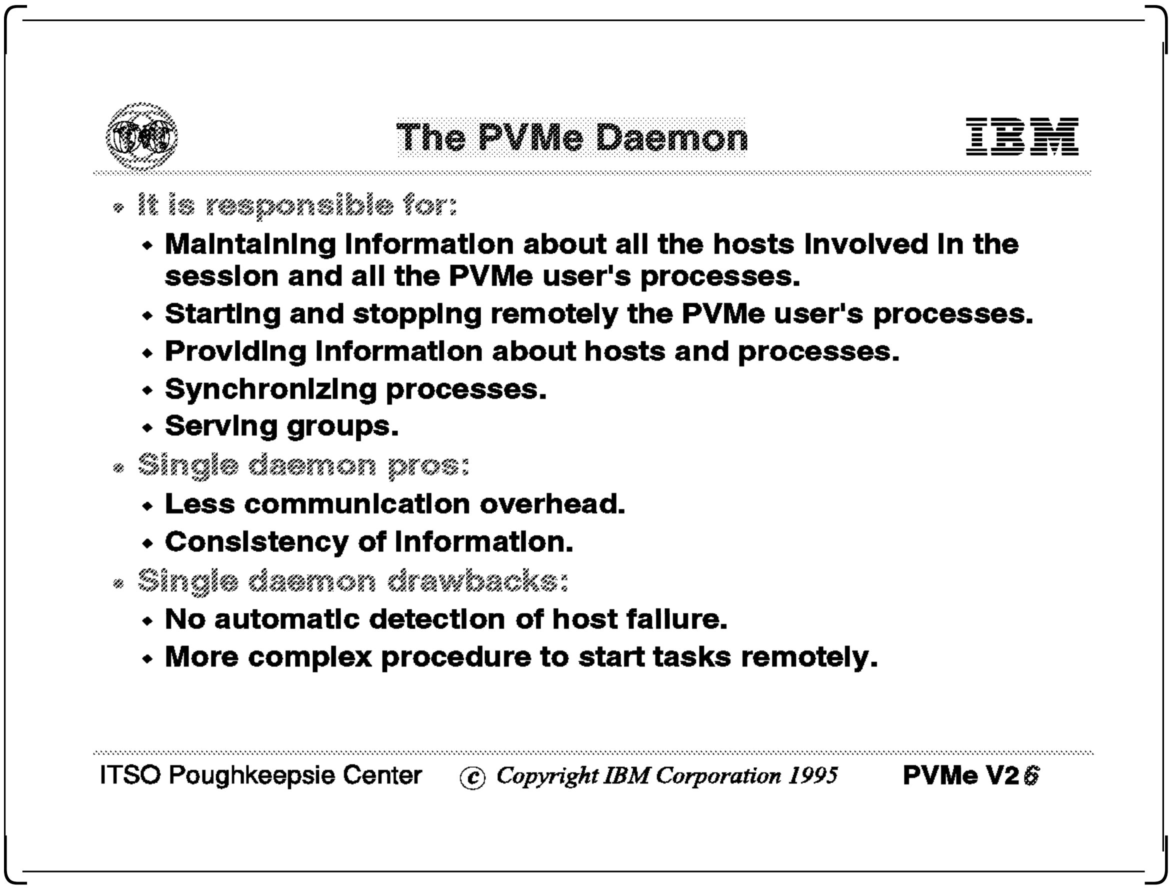
- In PVM, one service process runs on each host belonging to the *virtual machine*.

  Note: The term *virtual machine* is used to indicate the set of hosts where a PVM session is running. User tasks can be started in this environment, and multiple parallel applications can be executed at the same time.

  An additional daemon, the *group server*, is started if the application uses the group or the collective communication routines.

  In PVMe, only one instance of the service process (PVMe daemon) runs within the virtual machine. It also plays the role of group server if required.

3.2.3 The PVMe Daemon



The PVMe Daemon

- It is responsible for:
  - Maintaining information about all the hosts involved in the session and all the PVMe user's processes.
  - Starting and stopping remotely the PVMe user's processes.
  - Providing information about hosts and processes.
  - Synchronizing processes.
  - Serving groups.
- Single daemon pros:
  - Less communication overhead.
  - Consistency of information.
- Single daemon drawbacks:
  - No automatic detection of host failure.
  - More complex procedure to start tasks remotely.

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* **PVMe V26**

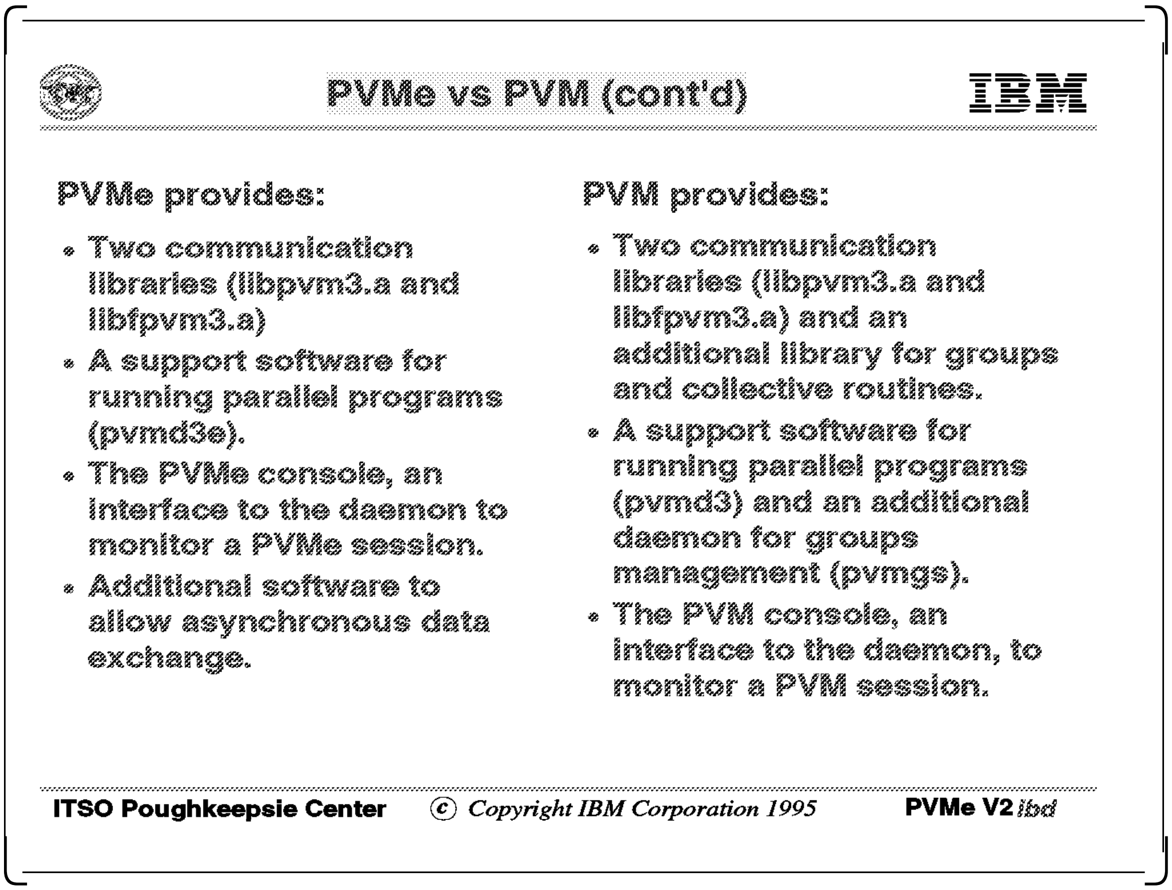
The PVMe daemon:

- Maintains a database of all the processes currently active in the virtual machine (their name, the host they are running on, task identifiers, addresses over the network, status of tasks, and so on) and of all the hosts participating in the virtual machine (host name, address, and so on)
- Starts or stops user′s tasks on the local or on the remote hosts. When a process calls the routine to spawn a new process, a request is sent to the daemon, which executes the command remotely and returns the result to the task. Whenever a new process is started, or is registered to the daemon, the daemon assigns to it a unique identifier that will be used later both for the task-to-task and for the task-daemon communication.
- Provides information about host and processes on demand.
- Manages groups, which implies:
  - Keeping a database for the group information (how many groups, which process belongs to which group, the processes′ identifiers within groups, and so on)
  - Assigning unique group identifiers to processes when they join a group
  - Performing global synchronization among processes belonging to the same group (barrier)

Since the all the information is kept by a single daemon, there is less communication overhead, compared with PVM, and also, there is no need for communication to keep information consistent on the different servers.

On the other side, the mechanism to start new processes is more complicated: a remote execution is required whenever a new user task is started. In PVM a remote execution is required only at the session startup (you start manually the first daemon, which, in turn starts the daemons on all the remote hosts); once all the daemons are active, the request to start a new task is sent to the daemon on the selected machine, which starts the process locally.

Typically, when you start a parallel application, the daemon will be asked to start all the other instances at the same time.



PVMe vs PVM (cont'd)

PVMe provides:

- Two communication libraries (libpvm3.a and libfpvm3.a)
- A support software for running parallel programs (pvmd3e).
- The PVMe console, an interface to the daemon to monitor a PVMe session.
- Additional software to allow asynchronous data exchange.

PVM provides:

- Two communication libraries (libpvm3.a and libfpvm3.a) and an additional library for groups and collective routines.
- A support software for running parallel programs (pvmd3) and an additional daemon for groups management (pvmgs).
- The PVM console, an interface to the daemon, to monitor a PVM session.

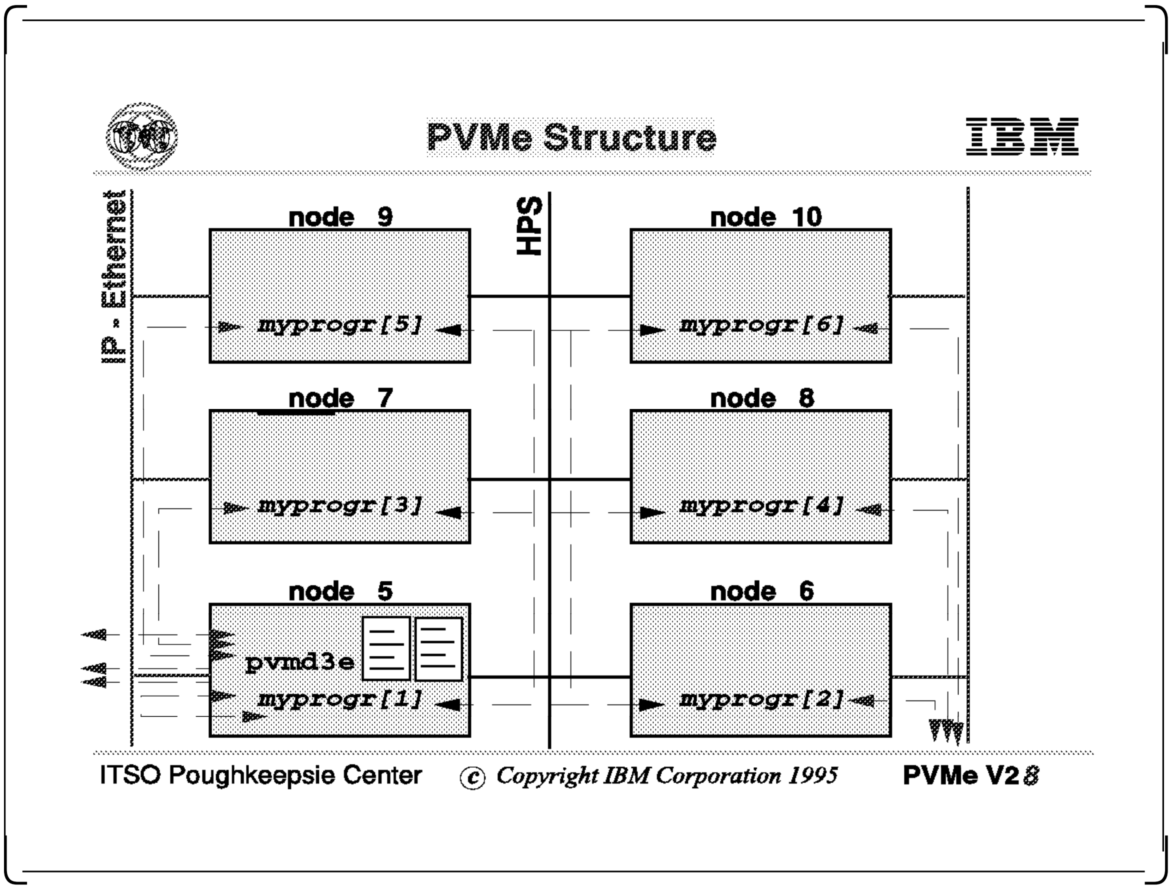
ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 *ibd*

PVMe does not provide the group library, since it is included in the general library, nor the group server software, because the PVMe daemon plays the role of group server.

Both provide the *console*, which is an additional program the user can run to monitor the virtual machine and the parallel program execution. The console connects to the daemon (if it is already running) and gives a prompt to the user, so that he can run several commands: query about the processes, about the hosts, start of a new application, kill processes, send signals, and so on. The console can then be stopped and started later during the session if desired.

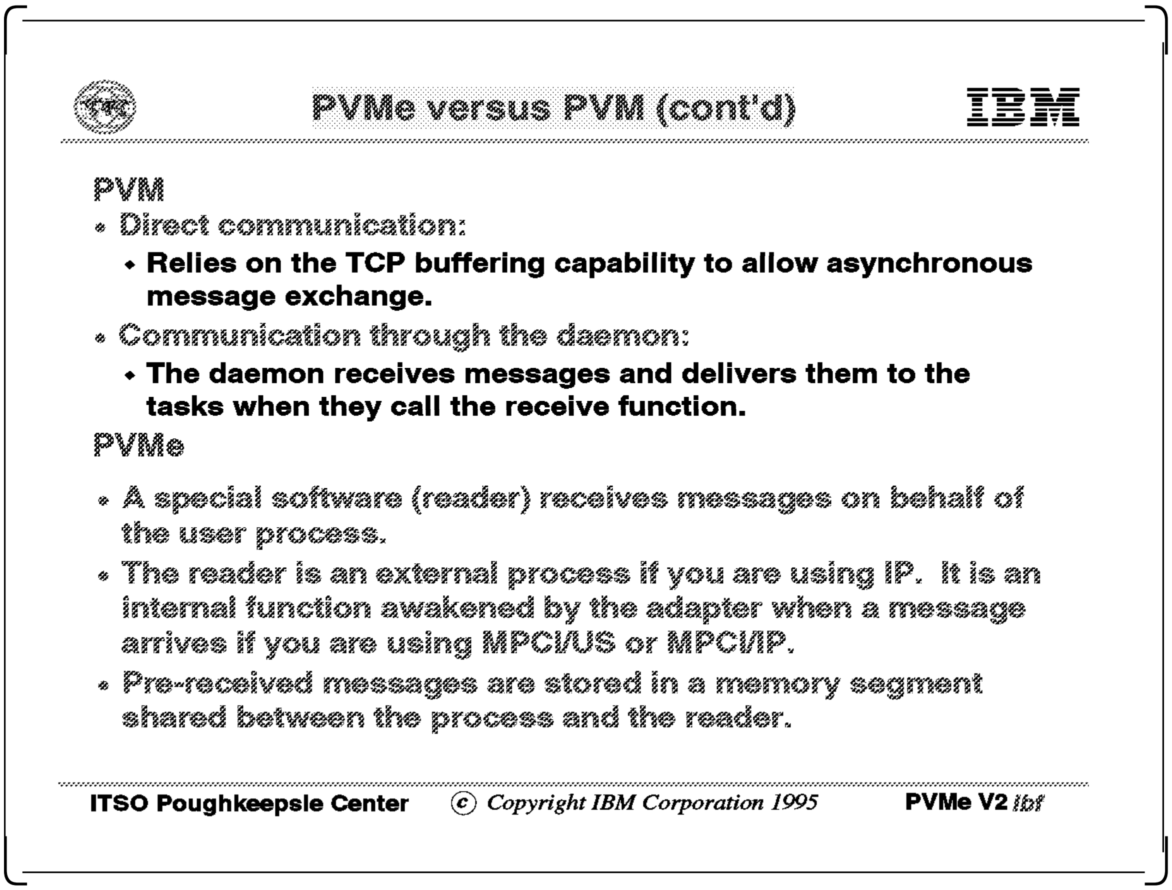
PVMe also provides an additional software that is used to manage asynchronous communication when IP is used for data communication. Note that the underlying communication layer, MPCI, has both US and IP modes. References to PMVe using IP involve bypassing MPCI, not switching MPCI mode. (see 3.2.4, “PVMe Structure” on page 61).

3.2.4 PVMe Structure



The picture shows the structure of a PVMe session:

- The daemon was started on one node (node 5 in the example), and controls six nodes (from 5 to 10). A parallel application is running on the six nodes, so that one instance of the parallel program is running on each node; the application was manually started on node 5, where the first instance is running.
- All the task-to-task communication (black dashed line) goes over the HPS, using the US protocol (by default); each task talks to the daemon through an IP connection (red dashed line) over the Ethernet.
- The daemon maintains information about the status of all processes and provides it to the user′s tasks on demand.



PVMe versus PVM (cont'd)

PVM

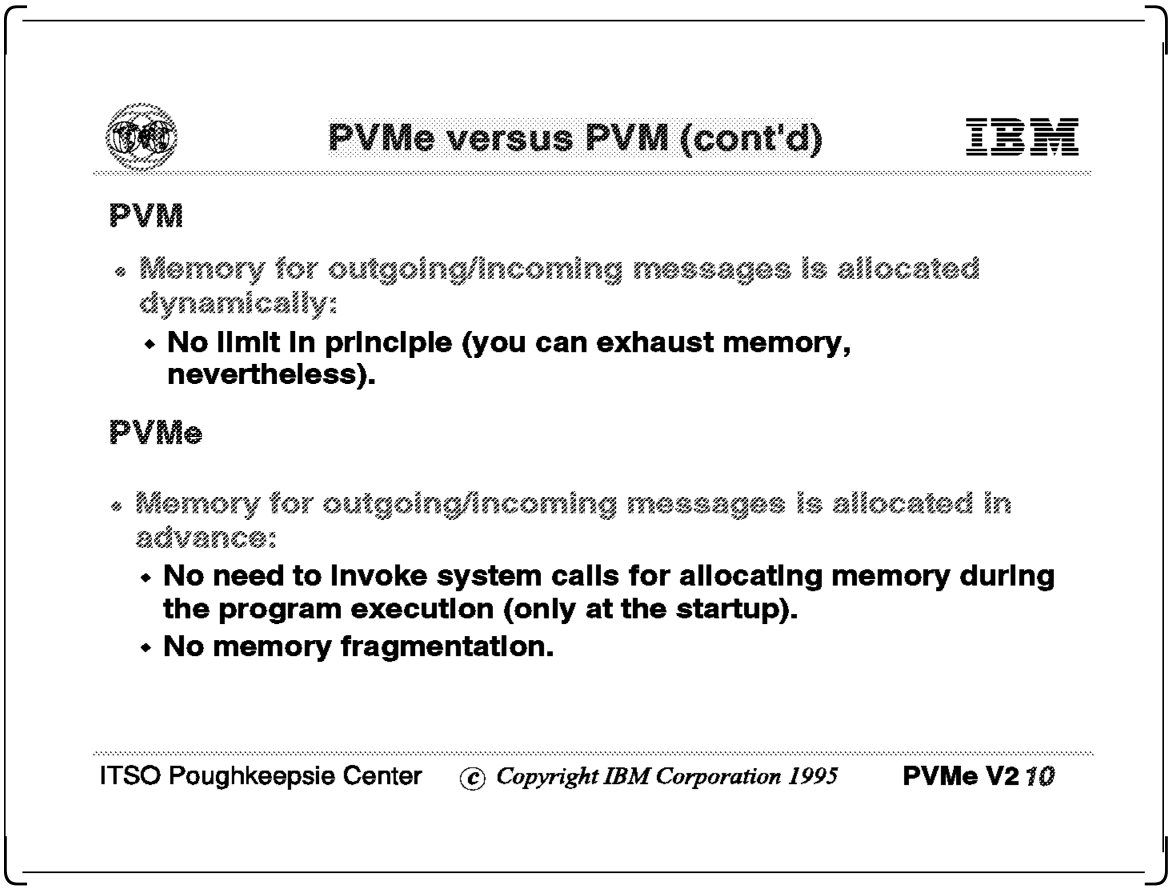
- Direct communication:
  - **Relies on the TCP buffering capability to allow asynchronous message exchange.**
- Communication through the daemon:
  - **The daemon receives messages and delivers them to the tasks when they call the receive function.**

PVMe

- A special software (reader) receives messages on behalf of the user process.
- The reader is an external process if you are using IP. It is an internal function awakened by the adapter when a message arrives if you are using MPCI/US or MPCI/IP.
- Pre-received messages are stored in a memory segment shared between the process and the reader.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V2** *lbf*

- When communication goes through the daemons (not extensively used any more), the daemons receive messages on behalf of the user′s tasks; when the user′s task calls the receive routine, the daemon delivers the message. The more efficient direct communication path uses a TCP connection between sender and receiver, and an asynchronous send operation is nonblocking as long as the buffering capabilities of the underlying protocol are not exhausted.
- In PVMe, a special software plays the role of message *reader*: it is an external process when you use the IP protocol directly; it is an internal function to the user′s task if you are using the MPCI/US or the MPCI/IP protocol. This function is awakened by a signal produced by the interrupt handler of the adapter whenever new packets arrive. This mechanism allows the sender process to complete the call although the recipient is not receiving the message.



PVMe versus PVM (cont'd)

PVM

- Memory for outgoing/incoming messages is allocated dynamically:
  - **No limit in principle (you can exhaust memory, nevertheless).**

PVMe

- Memory for outgoing/incoming messages is allocated in advance:
  - **No need to invoke system calls for allocating memory during the program execution (only at the startup).**
  - **No memory fragmentation.**

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* **PVMe V2** *10*

- PVM daemons and processes allocate memory dynamically during the application execution. Memory allocation is required:

  – To prepare a message (packing data into a buffer)
  – To receive a message

  Memory is freed:

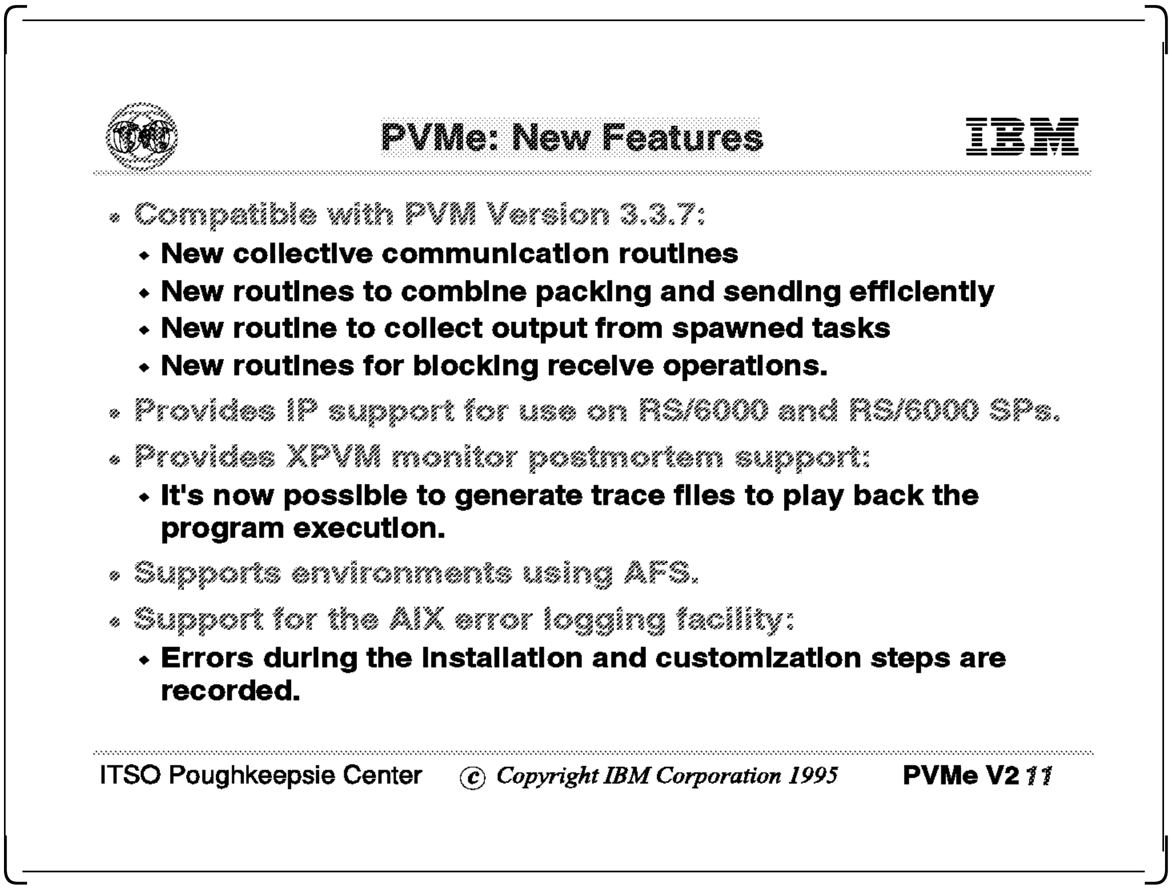
  – When the send buffer is cleared
  – When data is unpacked from a receiving buffer

  If no memory is available from the system, the PVM routines return a specific error code so that the program can take the proper action.

- In PVMe, an entire memory segment is allocated by the user's program at the startup, and memory for outgoing/incoming messages is taken from that segment. The operating system makes a true memory allocation only when a page from the segment is referenced, so that only the needed memory is really allocated, and once it is allocated, it can be reused for multiple messages. This method is efficient, since calls to allocate/deallocate memory are quite time consuming. Moreover, there is no risk of memory fragmentation.

  The memory segment is divided into buffers of different size that are commonly used for small, medium, and large size messages (they can be re-combined if needed).

3.2.5 PVMe: New Features



PVMe: New Features

- Compatible with PVM Version 3.3.7:
  - New collective communication routines
  - New routines to combine packing and sending efficiently
  - New routine to collect output from spawned tasks
  - New routines for blocking receive operations.
- Provides IP support for use on RS/6000 and RS/6000 SPs.
- Provides XPVM monitor postmortem support:
  - It's now possible to generate trace files to play back the program execution.
- Supports environments using AFS.
- Support for the AIX error logging facility:
  - Errors during the installation and customization steps are recorded.

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 11

The programming interface has been extended to reflect the updates of version 3.3.7 of public domain PVM. The extension includes:

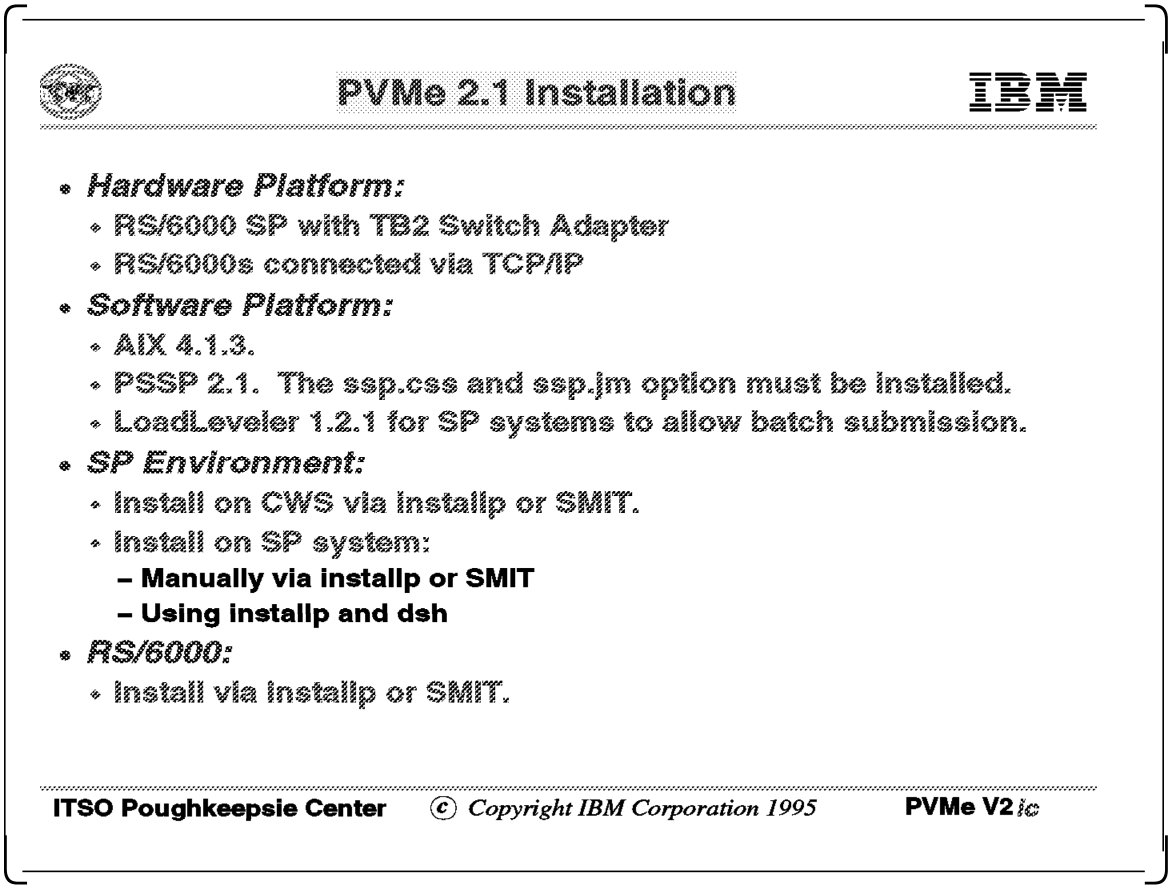
- Collective communication routines (they are already provided by other programming interfaces, such as MPL and MPI), which allow you to perform more sophisticated operations without burden to the programmer.
- Routines that allow you to pack data into a buffer and send the buffer to another task with a single call (and, on the other side, to receive a message and extract data from the receiving buffer with a single call). This is convenient when you send homogeneous data from a given memory region (for example, elements from the same array).
- Routine to collect output from spawned tasks.
- A blocking receive with time-out has been added.

PVMe provides support for running programs on the RS/6000 SP over the MPCI/IP protocol. When using the MPCI/US protocol, you get better performance, but only one user task can access the device (HPS Adapter-2) in user space mode at a time. During the development or testing phase when you are not so much interested in performance results; you probably prefer to share the resources with other users. In direct IP mode, you can run multiple parallel tasks on one SP node or RS/6000 machine.

Moreover you can set up the environment so that other RS/6000 machines participate in the run. Communication with clustered RS/6000 workstations will always be by direct IP no matter what mode is used within the SP.

You can enable the tracing facility so that different events are recorded to trace files during the program execution. After the program completes, these files, can be used by XPVM (public domain software available from the Oak Ridge National Labs, used for run-time and post-mortem monitoring of PVM programs) to play back the program execution. Load unbalancing among the tasks, bottlenecks, and so on, can be discovered and further tuning of the application is easier.

3.3 PVMe 2.1 Installation



PVMe 2.1 Installation

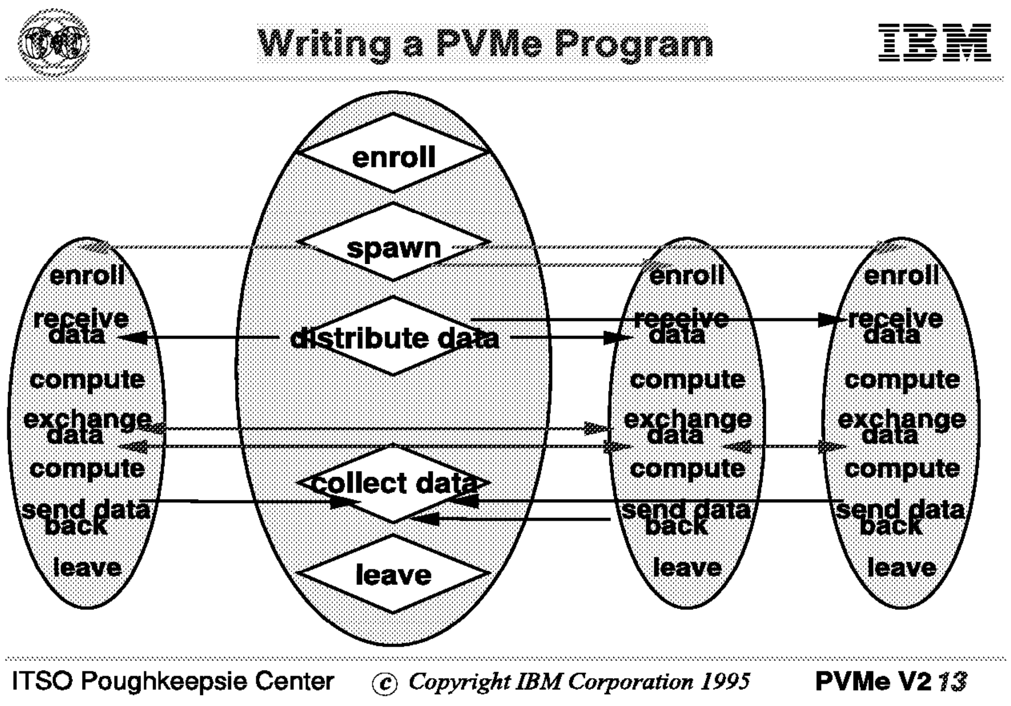
- *Hardware Platform:*
  - RS/6000 SP with TB2 Switch Adapter
  - RS/6000s connected via TCP/IP
- *Software Platform:*
  - AIX 4.1.3.
  - PSSP 2.1. The ssp.css and ssp.jm option must be installed.
  - LoadLeveler 1.2.1 for SP systems to allow batch submission.
- *SP Environment:*
  - Install on CWS via installp or SMIT.
  - Install on SP system:
    - Manually via installp or SMIT
    - Using installp and dsh
- *RS/6000:*
  - Install via installp or SMIT.

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 ic

When you install the product (login as root user), the PVMe directory tree is created under */usr/lpp/pvme*, and it contains the binaries for the daemon and service processes, the libraries, a sample program and makefile, configuration files (you only need to customize them if you want to override the default settings), and a kernel extension. Symbolic links to */usr/lib* are created for the PVMe libraries for your convenience, as well as links to /usr/bin for the daemon and service processes. The kernel extension will be loaded at each boot of the SP node since during the installation, the *rc.net* is updated to do that.

3.4 Writing a PVMe Program

This chapter presents the way you can design and develop a parallel program using the PVMe subroutines. You develop your parallel program using either the SPMD model or the MPMD model. The following graph describes a simple MPMD program with a master and its slaves.



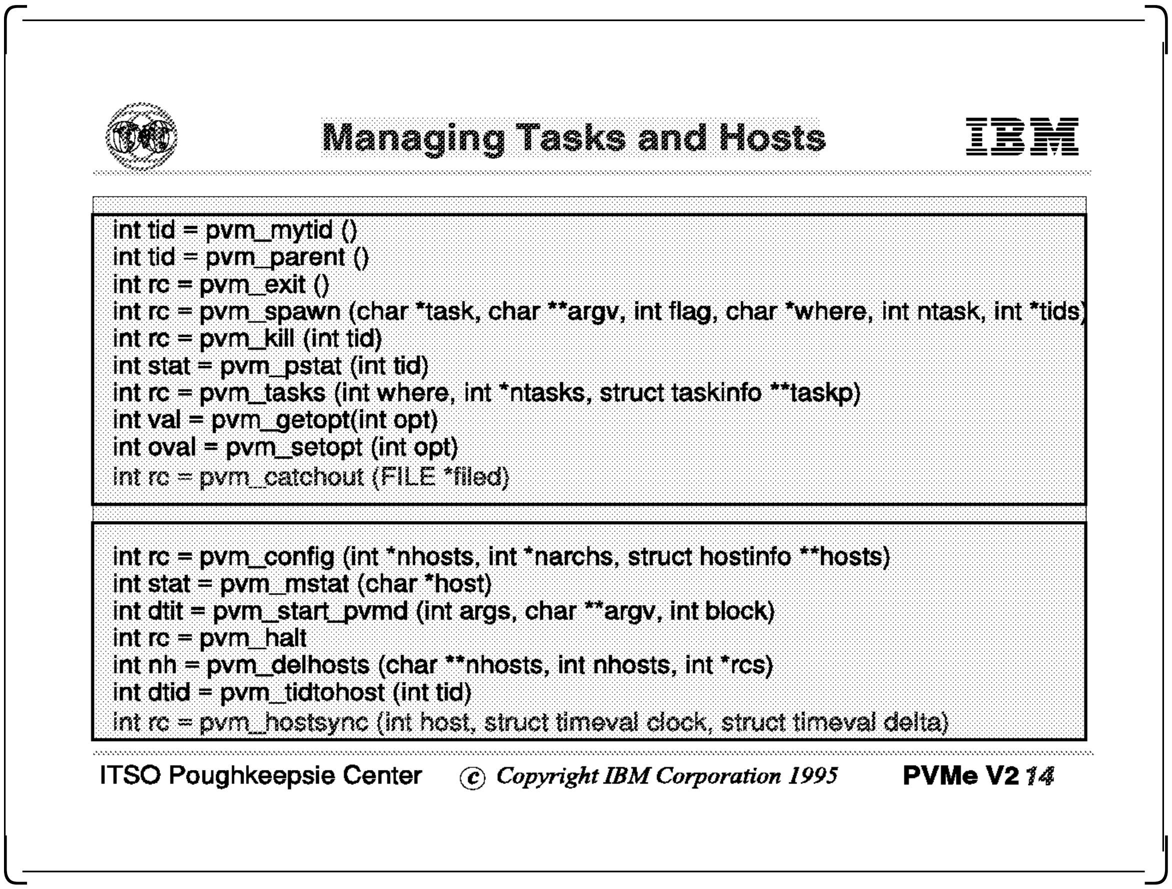
When the master process is started, it executes the following tasks:

- Spawning the slaves on nodes allocated by the resource manager
- Distributing the data to slaves
- Collecting the data from the slaves
- Ending the parallel program

The slave processes execute the following tasks:

- Receive data
- Execute the computation
- If needed, exchange data with other slaves
- Complete the computation
- Send the result to the master

3.4.1 Managing Tasks and Hosts



Managing Tasks and Hosts

```
int tid = pvm_mytid ()
int tid = pvm_parent ()
int rc = pvm_exit ()
int rc = pvm_spawn (char *task, char **argv, int flag, char *where, int ntask, int *tids)
int rc = pvm_kill (int tid)
int stat = pvm_pstat (int tid)
int rc = pvm_tasks (int where, int *ntasks, struct taskinfo **taskp)
int val = pvm_getopt(int opt)
int oval = pvm_setopt (int opt)
int rc = pvm_catchout (FILE *filed)
```

```
int rc = pvm_config (int *nhosts, int *narchs, struct hostinfo **hosts)
int stat = pvm_mstat (char *host)
int dtit = pvm_start_pvmd (int args, char **argv, int block)
int rc = pvm_halt
int nh = pvm_delhosts (char **nhosts, int nhosts, int *rcs)
int dtid = pvm_tidtohost (int tid)
int rc = pvm_hostsync (int host, struct timeval clock, struct timeval delta)
```

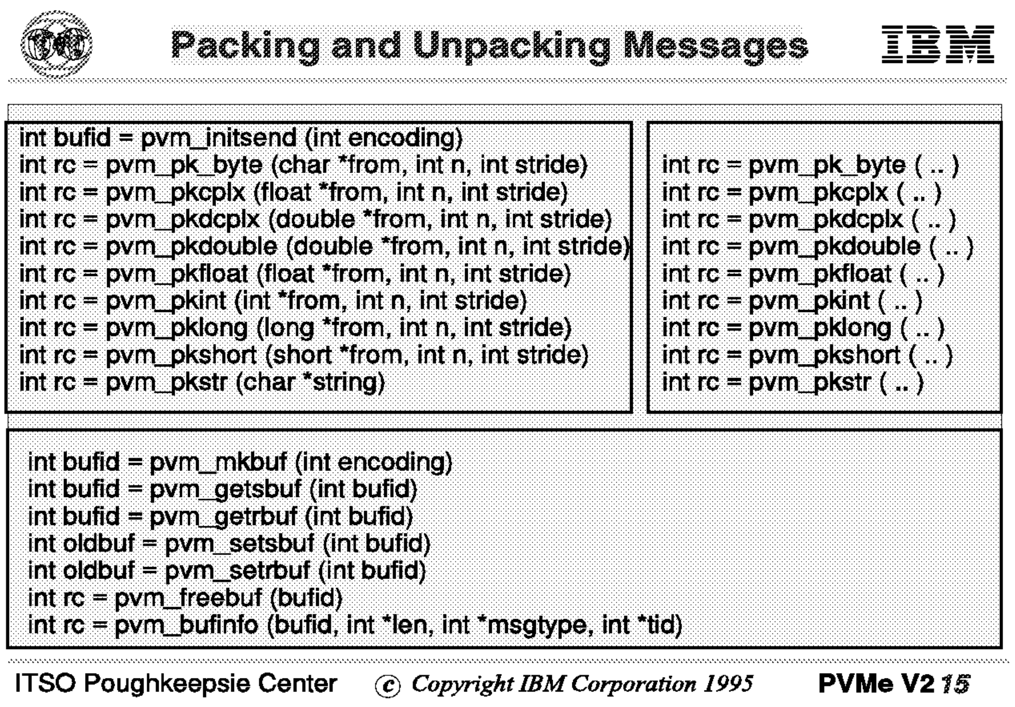
ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V2 14**

A detailed description of each routine can be found in *IBM PVMe for AIX User's Guide and Subroutine Reference Version 2 Release 1, GC23-3884-00*; we only give here a short description of the new features and, where needed, highlight the different behavior of PVMe compared to PVM.

- The *pvm_catchout* routine allows a PVMe process to catch the standard output and the standard error of its children tasks (by default the *stdout* and *stderr* of PVMe tasks are redirected to the *stdout* of the daemon). The *stdout* and *stderr* from the children are collected by the parent task, which prefixes each line with the task's identifier. You can specify a file descriptor as target for the collected outputs and errors.
- The *pvm_hostsync* has been implemented to maintain compatibility with the public domain PVM. It returns the time-of-day of a remote host in the virtual machine and the difference between the time-of-day on the local and remote host. This routine actually relies on the assumption that all the hosts are synchronized (it is mandatory on a RS/6000 SP system that all the nodes and the control workstation be synchronized), so that the time-of-day is the same on the remote and the local host, and the difference above is negligible. Be careful because this assumption is not valid any more if one or more workstations, external to the RS/6000 SP system, participate to the virtual machine.

Note: In PVM, the *pvm_addhosts* is also available. It allows you to add hosts to the virtual machine at any time. This routine is not available in PVMe, because the Resource Manager does not allow a set of nodes allocated for a given parallel job to be extended dynamically.

3.4.2 Packing and Unpacking Messages



Packing and Unpacking Messages

```
int bufid = pvm_initsend (int encoding)
int rc = pvm_pk_byte (char *from, int n, int stride)
int rc = pvm_pkcplx (float *from, int n, int stride)
int rc = pvm_pkdcplx (double *from, int n, int stride)
int rc = pvm_pkdouble (double *from, int n, int stride)
int rc = pvm_pkfloat (float *from, int n, int stride)
int rc = pvm_pkint (int *from, int n, int stride)
int rc = pvm_pklong (long *from, int n, int stride)
int rc = pvm_pkshort (short *from, int n, int stride)
int rc = pvm_pkstr (char *string)
```

```
int rc = pvm_pk_byte ( .. )
int rc = pvm_pkcplx ( .. )
int rc = pvm_pkdcplx ( .. )
int rc = pvm_pkdouble ( .. )
int rc = pvm_pkfloat ( .. )
int rc = pvm_pkint ( .. )
int rc = pvm_pklong ( .. )
int rc = pvm_pkshort ( .. )
int rc = pvm_pkstr ( .. )
```

```
int bufid = pvm_mkbuf (int encoding)
int bufid = pvm_getsbuf (int bufid)
int bufid = pvm_getrbuf (int bufid)
int oldbuf = pvm_setsbuf (int bufid)
int oldbuf = pvm_setrbuf (int bufid)
int rc = pvm_freebuf (bufid)
int rc = pvm_bufinfo (bufid, int *len, int *msgtype, int *tid)
```

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V2 15**

PVMe uses a sending buffer to store data that must be sent to another process. Before sending a message, it must be packed into the sending buffer by using the proper PVMe routines. Then, the correct sequence of operations is:

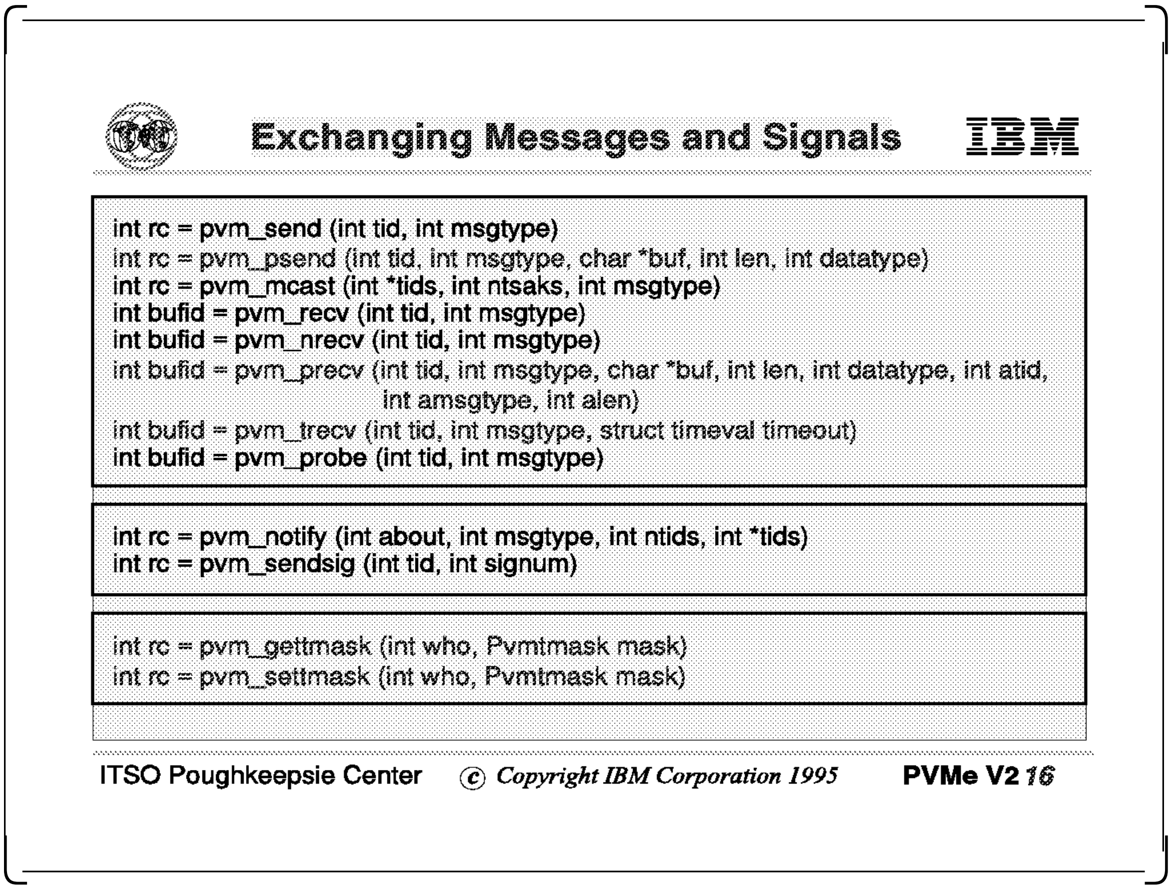
- Initialize the buffer
- Pack data into the buffer
- Send the message

Using the proper routines, you can pack multiple heterogeneous data into the same buffer, and then call the send routine only once. A special option to the *pvm_initsend* and *pvm_mkbuf* routine (also available in PVM) indicates that only the pointer and the size, not the data, must be copied into the send buffer during the packing operation. Data will be copied directly out of your memory only when a send is performed. This option allows you to avoid one data copy, with a substantial improvement in performance. Also, the same data can be sent multiple times, with modifications intervening between the sends, without having to pack it each time.

Once a message has been received, data must be unpacked from the receive buffer in the same order it was packed into the sending buffer.

Multiple buffers can be managed within the same application, but only one of them is active at any time. However, most applications do not need to manage more than one send buffer. If you do not need to manage multiple buffers, you are provided with transparent default send/receive buffers.

3.4.3 Exchanging Messages and Signals



Exchanging Messages and Signals

```
int rc = pvm_send (int tid, int msgtype)
int rc = pvm_psend (int tid, int msgtype, char *buf, int len, int datatype)
int rc = pvm_mcast (int *tids, int ntsaks, int msgtype)
int bufid = pvm_recv (int tid, int msgtype)
int bufid = pvm_nrecv (int tid, int msgtype)
int bufid = pvm_precv (int tid, int msgtype, char *buf, int len, int datatype, int atid,
                int amsgtype, int alen)
int bufid = pvm_trecv (int tid, int msgtype, struct timeval timeout)
int bufid = pvm_probe (int tid, int msgtype)
```

```
int rc = pvm_notify (int about, int msgtype, int ntids, int *tids)
int rc = pvm_sendsig (int tid, int signum)
```

```
int rc = pvm_gettmask (int who, Pvmtmask mask)
int rc = pvm_settmask (int who, Pvmtmask mask)
```

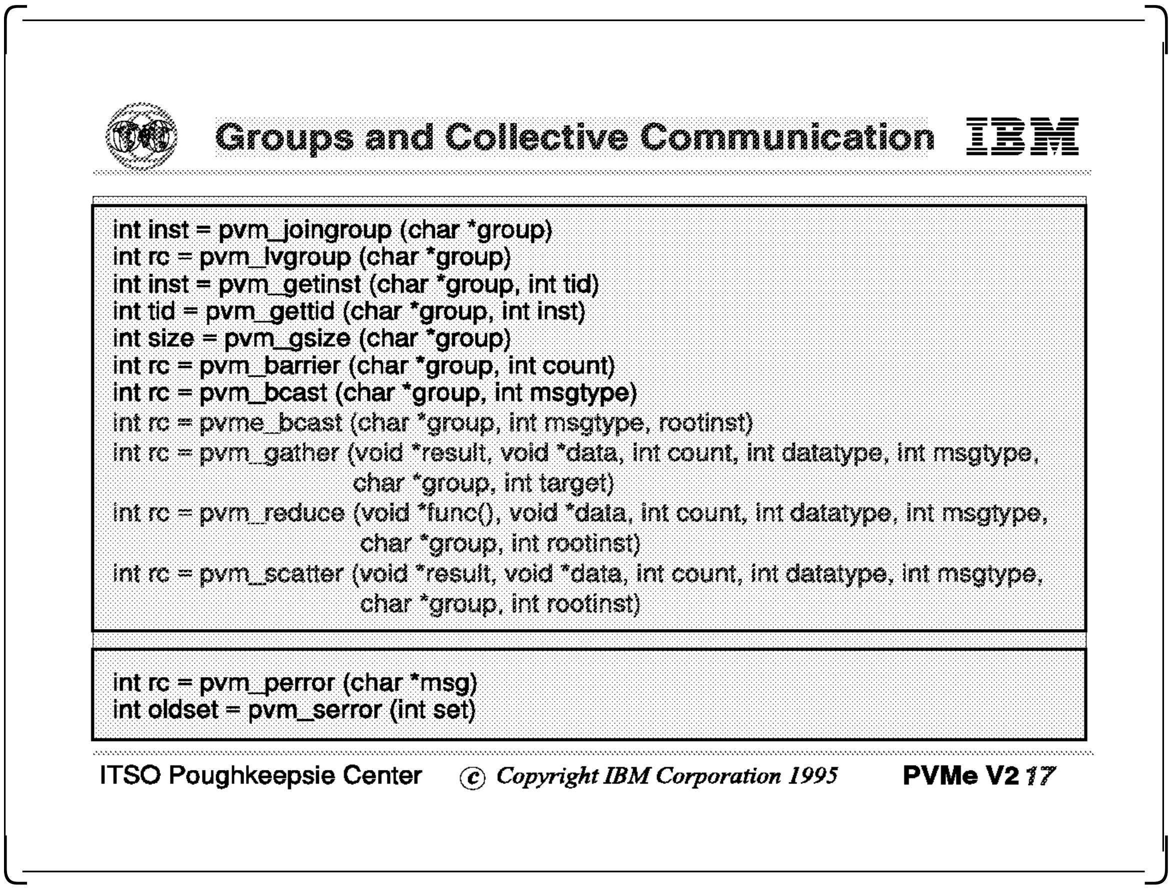
ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V2** *16*

- The *pvm_psend* routine packs data (with the supplied *datatype*, starting from the supplied pointer) into a message and sends the message to the *tid* task with type *msgtype*. Data packing and send are performed within a single call.

  Similarly, the *pvm_precv* routine receives a message with the supplied *msgtype* coming from the supplied *tid*, and also unpacks and copies the message content to a memory region supplied by the user.

- The *pvm_trecv* routine receives a message with the supplied *msgtype* coming from the process with the supplied *tid*, and places it in the current receive buffer. If a message matching the supplied parameters has arrived, it returns soon. If not, it blocks waiting for the proper message. However, if the message does not arrive within a user supplied *timeout*, the routine returns without a message.

3.4.4 Groups and Collective Communications



Groups and Collective Communication

```
int inst = pvm_joingroup (char *group)
int rc = pvm_lvgroup (char *group)
int inst = pvm_getinst (char *group, int tid)
int tid = pvm_gettid (char *group, int inst)
int size = pvm_gsize (char *group)
int rc = pvm_barrier (char *group, int count)
int rc = pvm_bcast (char *group, int msgtype)
int rc = pvme_bcast (char *group, int msgtype, rootinst)
int rc = pvm_gather (void *result, void *data, int count, int datatype, int msgtype,
                     char *group, int target)
int rc = pvm_reduce (void *func(), void *data, int count, int datatype, int msgtype,
                     char *group, int rootinst)
int rc = pvm_scatter (void *result, void *data, int count, int datatype, int msgtype,
                     char *group, int rootinst)
```

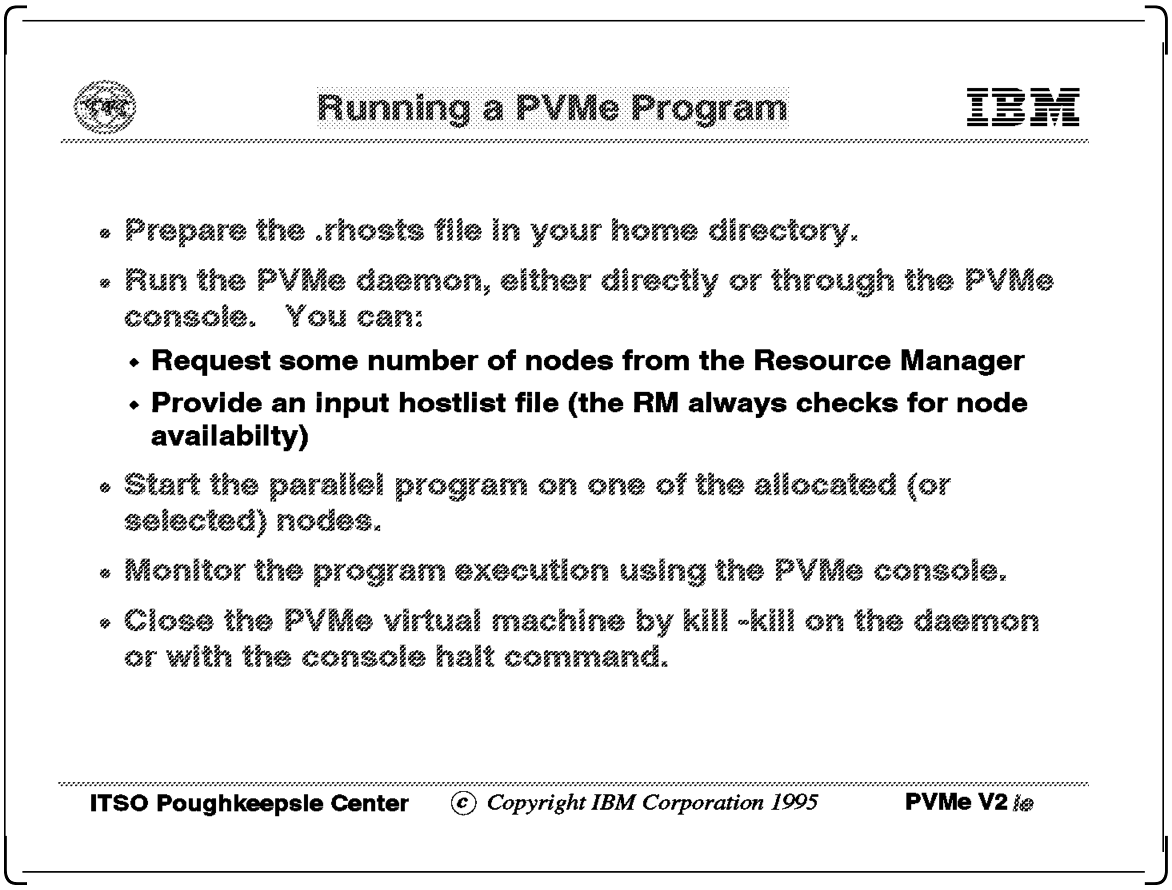
```
int rc = pvm_perror (char *msg)
int oldset = pvm_serror (int set)
```

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 17

- The *pvme_bcast* routine broadcasts the message stored in the active send buffer to all the other members of a given group. This routine is an extension (is not available in PVM) and exploits a more efficient algorithm to distribute data, when compared to the *pvm_bcast*.
- The *pvm_scatter* routine is used to distribute data from a specified root member of a group to all members of the same group (including the root task). Each member of the group must call the routine, and each of them receives a portion of a given data array owned by the root member.
- The *pvm_gather* performs the opposite: it gathers data from each member of a group to a single member of the same group. All group members call the routine; each of them sends a portion of a given data array to the target process. The target process concatenates incoming messages in order on the basis of the group instance number of the sender task. At the end, data from all group members will be stored in a single array on the target process.
- The *pvm_reduce* routine performs a global operation over all the members of a specified group. All group members call the routine, providing the local data, while the final result will be computed and will be available only on the (user supplied) root member. On the root member, the local data will be overwritten with the global operation result.

PVMe supplies a set of predefined global functions (PvmMin, PvmMax, PvmSum, PvmProduct), but user written functions performing other tasks can be used as well (they only have to obey a specific prototype).

3.5 Running a PVMe Program



Running a PVMe Program

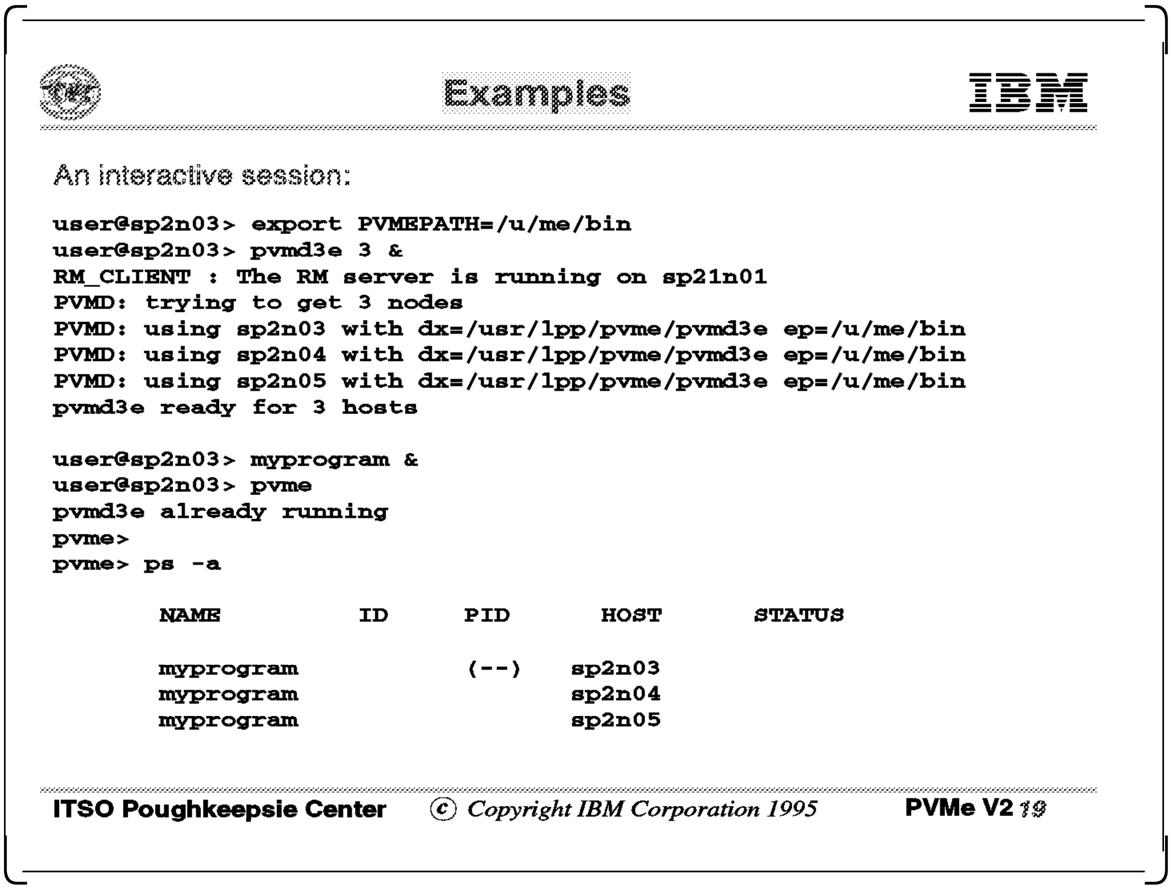
- Prepare the .rhosts file in your home directory.
- Run the PVMe daemon, either directly or through the PVMe console. You can:
  - **Request some number of nodes from the Resource Manager**
  - **Provide an input hostlist file (the RM always checks for node availabilty)**
- Start the parallel program on one of the allocated (or selected) nodes.
- Monitor the program execution using the PVMe console.
- Close the PVMe virtual machine by kill -kill on the daemon or with the console halt command.

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2

- Prepare the *.rhosts* file in your home directory so that you can run remote commands on the SP nodes (since you don′t know which nodes will be allocated for you, you should include in the file all the nodes belonging to the partition or RM pool you are running on).
- To start the daemon, you have to start up the PVMe environment. You run the daemon or the console from the node you are logged into, either supplying the number of nodes you need (so that a request is made to the RM) or supplying a *hostlist* file.
- Start the parallel program from any of the allocated nodes.

  Note: The node you are logged into may be part of the set of nodes allocated for the parallel job, or may not be. If the local node is not part of the set, you need to login into another node, spawn the program using the PVMe console, or start the program with an *rsh*.
- You can use the console to monitor the program execution.
- Once the application completes, the PVMe virtual machine is still active. You can run other applications on the same set of nodes, but if you don′t want to run other applications, don′t forget to halt the PVMe daemon to free the resources. Otherwise they are reserved for you and will not be allocated for other parallel jobs.

3.5.1 Examples



Examples

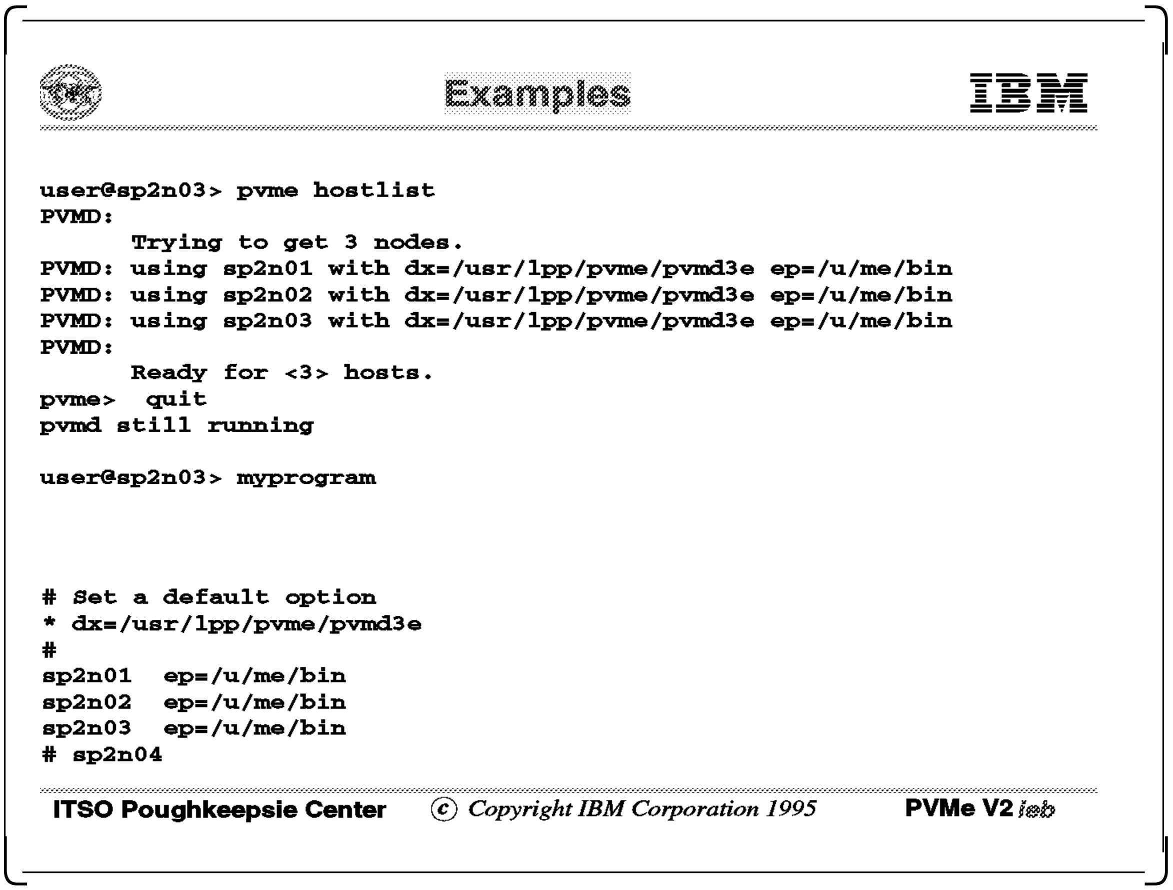
An interactive session:

```
user@sp2n03> export PVMEPATH=/u/me/bin
user@sp2n03> pvmd3e 3 &
RM_CLIENT : The RM server is running on sp21n01
PVMD: trying to get 3 nodes
PVMD: using sp2n03 with dx=/usr/lpp/pvme/pvmd3e ep=/u/me/bin
PVMD: using sp2n04 with dx=/usr/lpp/pvme/pvmd3e ep=/u/me/bin
PVMD: using sp2n05 with dx=/usr/lpp/pvme/pvmd3e ep=/u/me/bin
pvmd3e ready for 3 hosts

user@sp2n03> myprogram &
user@sp2n03> pvme
pvmd3e already running
pvme>
pvme> ps -a

       NAME         ID     PID      HOST      STATUS

       myprogram           (--)   sp2n03
       myprogram                  sp2n04
       myprogram                  sp2n05
```

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 19

In this example you:

1. Run the daemon supplying the number of nodes you need. If nodes are available from the RM, the PVMe daemon will print the list of nodes and a message saying that it is ready.
2. Start the application.
3. Start the console and ask for information about all the processes running in the virtual machine: one instance of the program is active on each node.



Examples

```
user@sp2n03> pvme hostlist
PVMD:
      Trying to get 3 nodes.
PVMD: using sp2n01 with dx=/usr/lpp/pvme/pvmd3e ep=/u/me/bin
PVMD: using sp2n02 with dx=/usr/lpp/pvme/pvmd3e ep=/u/me/bin
PVMD: using sp2n03 with dx=/usr/lpp/pvme/pvmd3e ep=/u/me/bin
PVMD:
      Ready for <3> hosts.
pvme>  quit
pvmd still running

user@sp2n03> myprogram
```

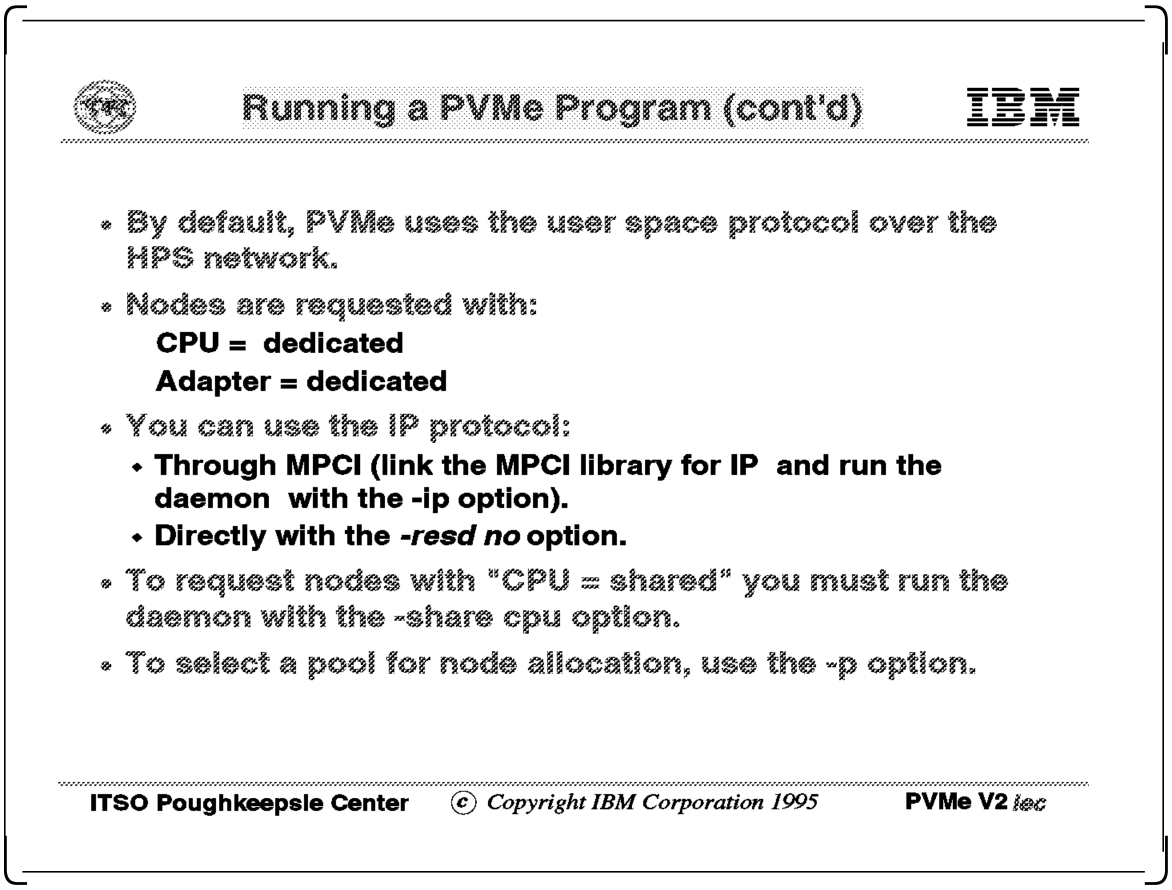
```
# Set a default option
* dx=/usr/lpp/pvme/pvmd3e
#
sp2n01  ep=/u/me/bin
sp2n02  ep=/u/me/bin
sp2n03  ep=/u/me/bin
# sp2n04
```

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 ieb

In this example you:

- Directly start the console, supplying a *hostlist* file. Since there is no PVMe daemon running, the console also starts the daemon and passes the *hostlist* file argument to the daemon. Then the console gives you the prompt.
- With the `quit` command, you stop the console, but the daemon is still running. If you run the console later, it will connect to the running daemon.
- You start the parallel program.
- In the *hostlist* file, you specify the list of nodes you want to use. The RM always verifies the availability of those nodes, and reserves them for you. If nodes are not available, the daemon will print an error message.

  For each host, you can specify a set of options. The most used is the *ep* option, where you can specify the path to the executables of the parallel program. The default value is the *$HOME/pvm3/bin/RS6K* directory; if you are using another location, you have to inform the daemon, so that it can fulfil requests for starting new processes in the future.



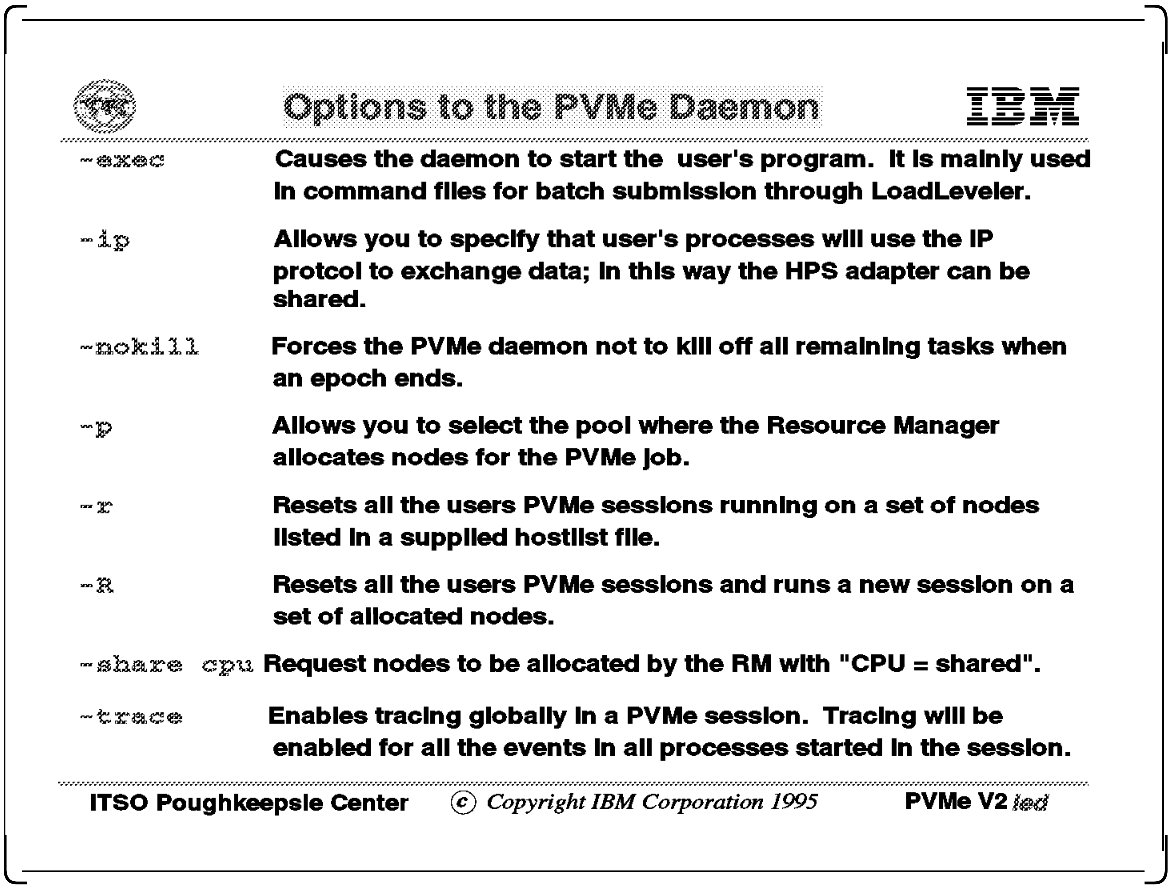
Running a PVMe Program (cont'd)

- By default, PVMe uses the user space protocol over the HPS network.
- Nodes are requested with:
  CPU = dedicated
  Adapter = dedicated
- You can use the IP protocol:
  - **Through MPCI (link the MPCI library for IP and run the daemon with the -ip option).**
  - **Directly with the *-resd no* option.**
- To request nodes with "CPU = shared" you must run the daemon with the -share cpu option.
- To select a pool for node allocation, use the -p option.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PVMe V2**

- By default, PVMe uses the MPCI/US protocol over the HPS. This path gives better performance, but poses the constraint that only one user process can access the device (HPS Adapter-2). Nodes are requested with both CPU and the HPS Adapter-2 adapter *dedicated*.
- To use the MPCI/IP protocol you have to link the proper library and run the daemon with the *-ip* option. Still one process per node is allowed in a PVMe virtual machine, but multiple users can run multiple sessions at the same time.
- You can request nodes to be allocated with shared cpu by specifying *-share cpu*. In this way, even if you program is running over the US, other non-PVMe parallel applications (MPL programs, for example) can run on the same set of nodes using another network adapter (the ethernet adapter, for example).

3.5.2 Options to the PVMe Daemon



Options to the PVMe Daemon

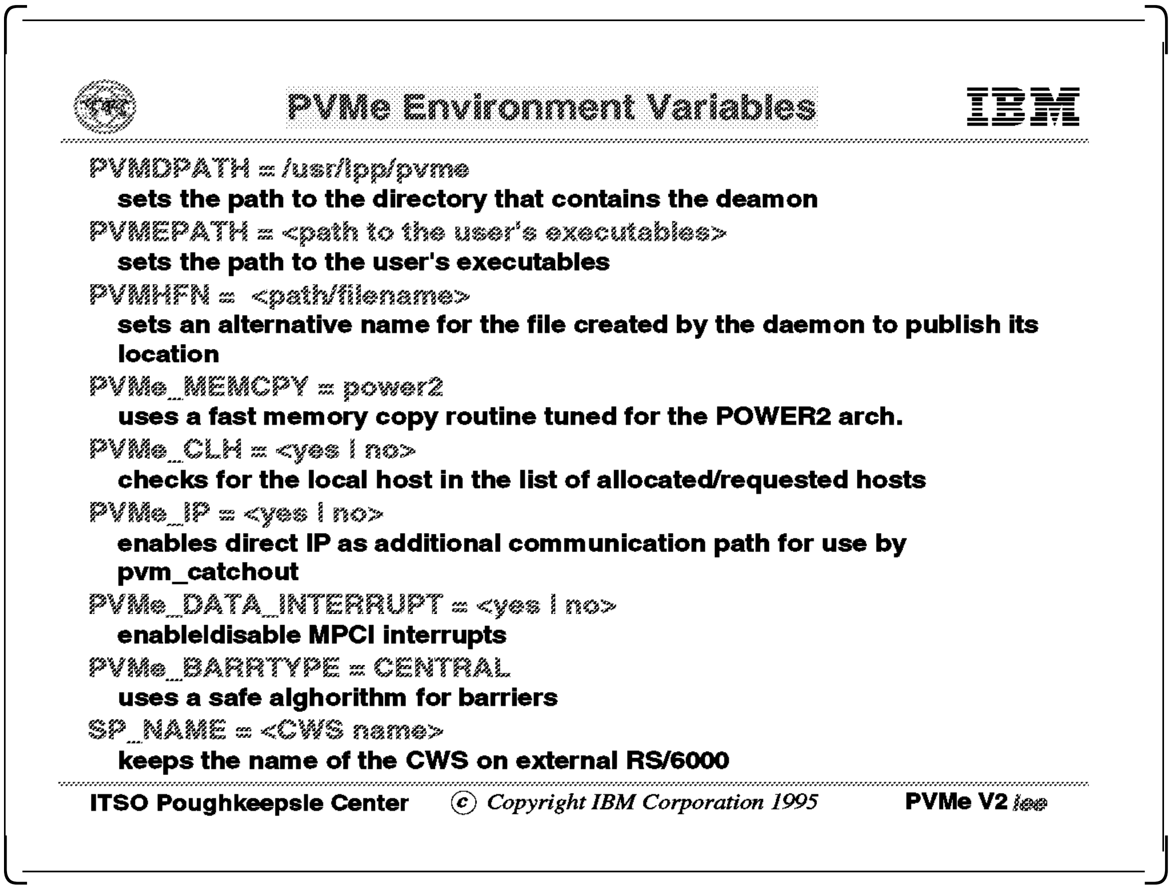
-exec	Causes the daemon to start the user's program. It is mainly used in command files for batch submission through LoadLeveler.
-ip	Allows you to specify that user's processes will use the IP protcol to exchange data; in this way the HPS adapter can be shared.
-nokill	Forces the PVMe daemon not to kill off all remaining tasks when an epoch ends.
-p	Allows you to select the pool where the Resource Manager allocates nodes for the PVMe job.
-r	Resets all the users PVMe sessions running on a set of nodes listed in a supplied hostlist file.
-R	Resets all the users PVMe sessions and runs a new session on a set of allocated nodes.
-share cpu	Request nodes to be allocated by the RM with "CPU = shared".
-trace	Enables tracing globally in a PVMe session. Tracing will be enabled for all the events in all processes started in the session.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PVMe V2 *led*

The -exec option causes the daemon to start up and run the user′s program, and shut down when the program has finished. Specifying -ip and -share cpu allows multiple sessions to run simultaneously on the same nodes. If the -trace option is specified, any jobs that are run will produce trace data that will be stored in the file /tmp/xpvm.trace.$USER on the node on which the daemon is running. This data file can be read by XPVM, a freeware trace visualization tool, to analyze program execution.

XPVM is available on the Internet through anonymous ftp from (netlib2.cs.utk.edu).

3.5.3 PVMe Environment Variables



PVMe Environment Variables

PVMDPATH = /usr/lpp/pvme
sets the path to the directory that contains the deamon

PVMEPATH = <path to the user's executables>
sets the path to the user's executables

PVMHFN = <path/filename>
sets an alternative name for the file created by the daemon to publish its location

PVMe_MEMCPY = power2
uses a fast memory copy routine tuned for the POWER2 arch.

PVMe_CLH = <yes | no>
checks for the local host in the list of allocated/requested hosts

PVMe_IP = <yes | no>
enables direct IP as additional communication path for use by pvm_catchout

PVMe_DATA_INTERRUPT = <yes | no>
enable|disable MPCI interrupts

PVMe_BARRTYPE = CENTRAL
uses a safe alghorithm for barriers

SP_NAME = <CWS name>
keeps the name of the CWS on external RS/6000

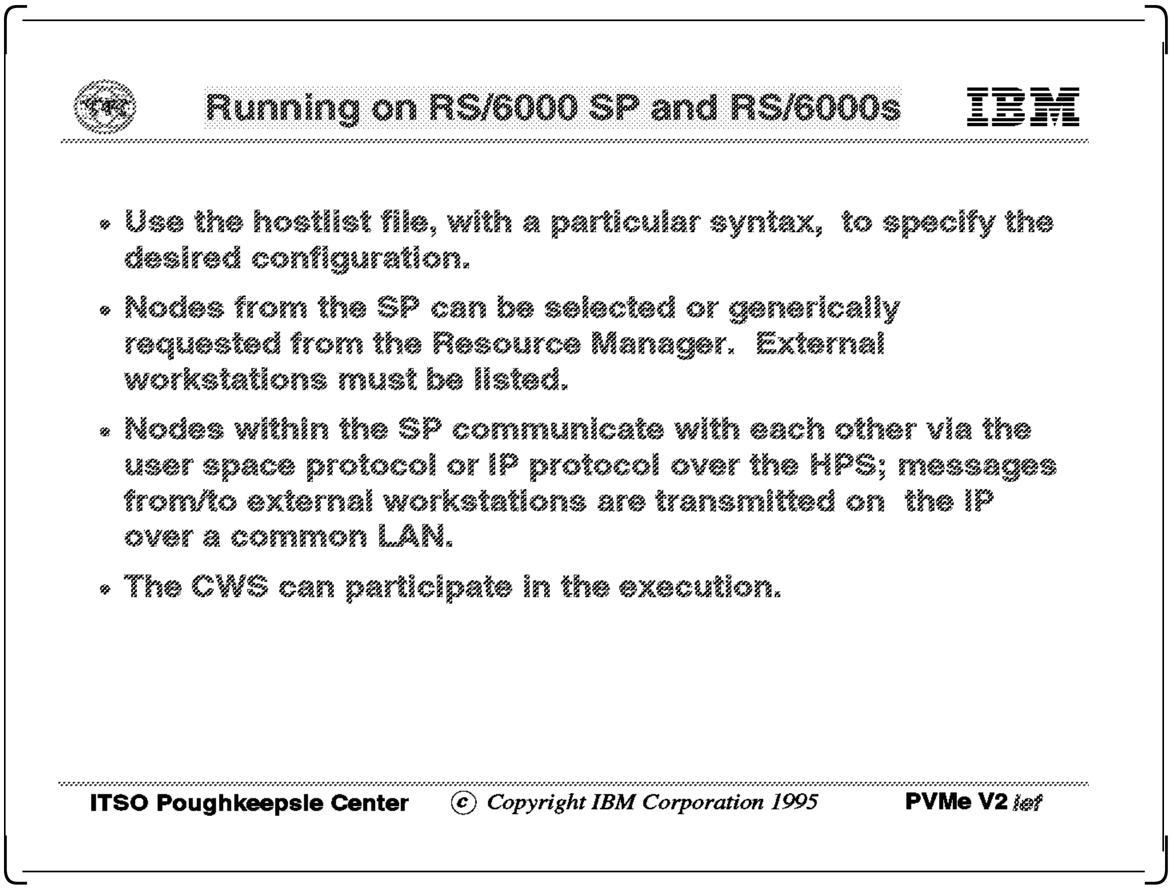
ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PVMe V2 lee

By default, when spawning off copies of a program, PVMe looks for the executable file in the directory $HOME/pvm3/bin/RS6K. If PVMEPATH is set in the environment of the daemon, PVMe will use the value as the name of the directory to search for the executable. PVMHFN specifies the name of a file that is created in the user′s home directory that contains information necessary for user programs to attach to the PVMe daemon. By default, the filename is .pvmdname. To run more than one simultaneous session, it is necessary to set the PVMHFN in the environment of the PVMe daemon and the user program to a unique value for each session.

Enabling the special memory copy routine with PVMe_MEMCPY=power2 requires that all the nodes in the PVMe session be of the POWER2 architecture. In MPCI mode, If PVMe_IP is set to yes before the PVMe daemon is started, the ipproc daemon, which passes redirected output from slave to slave, will also be started. Without this option turned on, the pvm_catchout routine will not function.

PVMe features an enhanced pvm_barrier routine that provides a much more efficient synchronization than PVM. However, it requires that all synchronizations involve every member of a group. If running a program that synchronizes a fraction of a group, setting PVMe_BARRTYPE=CENTRAL before running the program will disable the enhanced barrier algorithm and use a simple algorithm that will work with partial groups.

3.5.4 Running on RS/6000 and RS/6000 SP



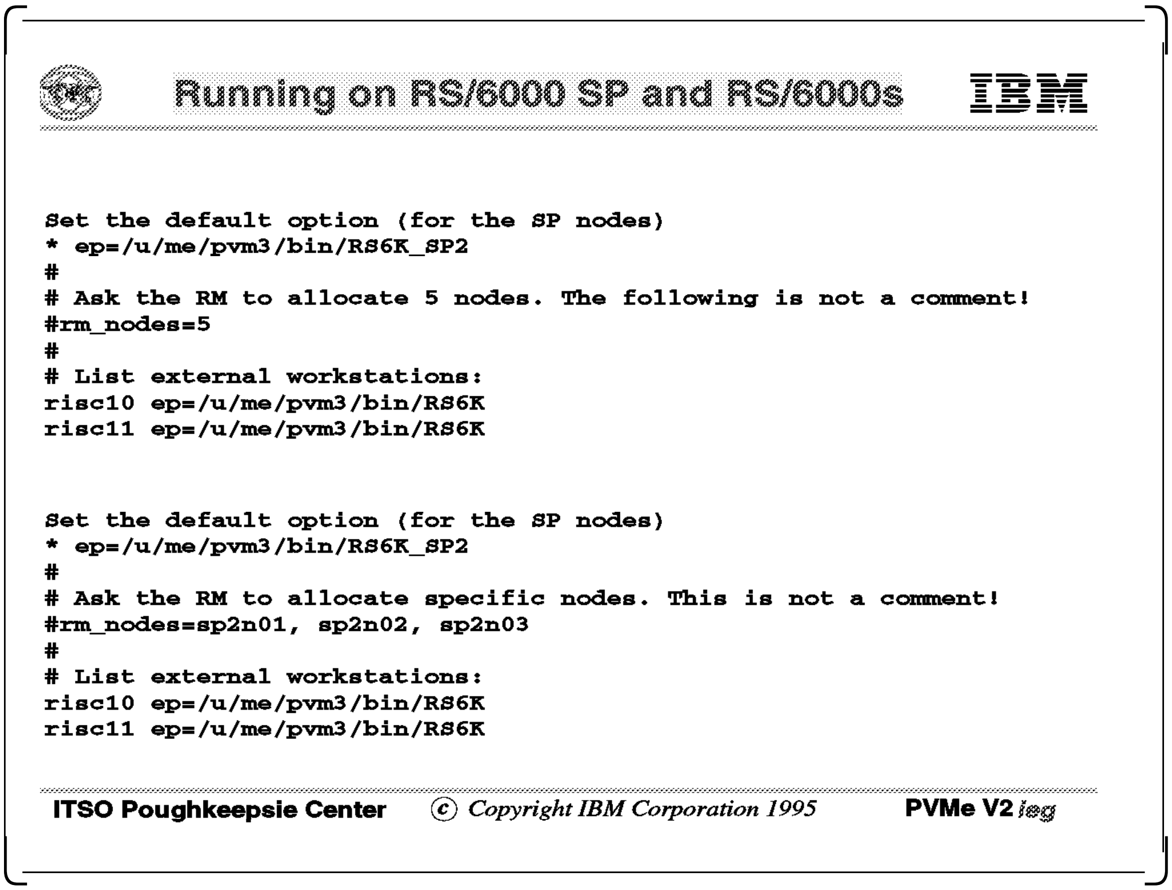
Running on RS/6000 SP and RS/6000s

- Use the hostlist file, with a particular syntax, to specify the desired configuration.
- Nodes from the SP can be selected or generically requested from the Resource Manager. External workstations must be listed.
- Nodes within the SP communicate with each other via the user space protocol or IP protocol over the HPS; messages from/to external workstations are transmitted on the IP over a common LAN.
- The CWS can participate in the execution.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PVMe V2 *lef*

To run PVMe on a mixed group of hosts (SP nodes + external RS/6000s), you have to run the daemon or the console supplying the *hostlist* file, with a special syntax (described in the next foil).

Communication among the SP nodes exploits the MPCI/US or MPCI/IP protocol over the HPS while communication with the external workstations occurs over the IP on the external LAN. This feature is particularly helpful when you isolate a task that doesn′t need frequent data exchange with the other tasks, or that doesn′t have to provide data to the computational nucleus. For example, you may use a graphical workstation to display the progress of your application: the task running on it receives intermediate results from the computational tasks when they are available, and displays them graphically, but is not really part of the execution.



Running on RS/6000 SP and RS/6000s

```
Set the default option (for the SP nodes)
* ep=/u/me/pvm3/bin/RS6K_SP2
#
# Ask the RM to allocate 5 nodes. The following is not a comment!
#rm_nodes=5
#
# List external workstations:
risc10 ep=/u/me/pvm3/bin/RS6K
risc11 ep=/u/me/pvm3/bin/RS6K
```

```
Set the default option (for the SP nodes)
* ep=/u/me/pvm3/bin/RS6K_SP2
#
# Ask the RM to allocate specific nodes. This is not a comment!
#rm_nodes=sp2n01, sp2n02, sp2n03
#
# List external workstations:
risc10 ep=/u/me/pvm3/bin/RS6K
risc11 ep=/u/me/pvm3/bin/RS6K
```

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1995* **PVMe V2**

On this foil, there are two examples of host list files that include RS/6000 SP nodes and clustered RS/6000 workstations:

- Generic request:

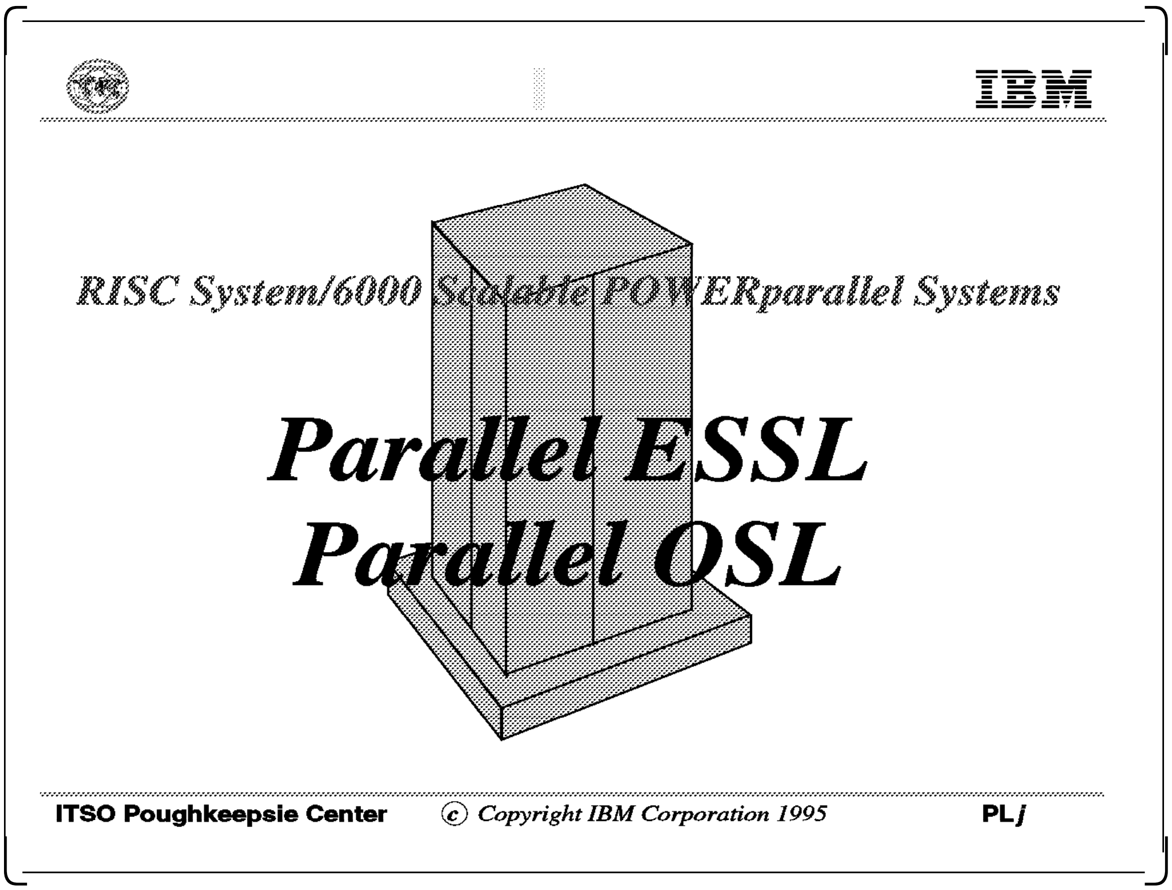
  The statement `#rm_nodes=5` requests five RS/6000 SP nodes that will be allocated by the resource manager. Because the node hostnames are not specified, the resource manager selects available nodes in the interactive or general pool.

- Explicit request:

  The statement `#rm_nodes=sp2n01, sp2n02, sp2n03` requests specific RS/6000 SP nodes. So, the resource manager will allocate these nodes if they are available.

Clustered RS/6000 hostnames are always specified, such as `risc10` and `risc11` in these examples. The RS/6000 used in this way must be in the same IP domain as the SP.

Chapter 4. Parallel ESSL and Parallel OSL



RISC System/6000 Scalable POWERparallel Systems

Parallel ESSL
Parallel OSL

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PLj

Engineering and Scientific Subroutine Library (ESSL/370) was developed to provide a set of mathematical subroutines optimized for the ES/3090 and ES/9000 vector facility feature. When mainframe scientific users migrated to RS/6000 workstations and AIX/6000 operating system, IBM provided them with ESSL/6000, a new version of ESSL subroutines written for an optimized use of the POWER and POWER2 processor architecture.

To allow users to develop portable codes, ESSL uses the de facto standards available on the market, such as BLAS (Basic Linear Algebra Subroutines).

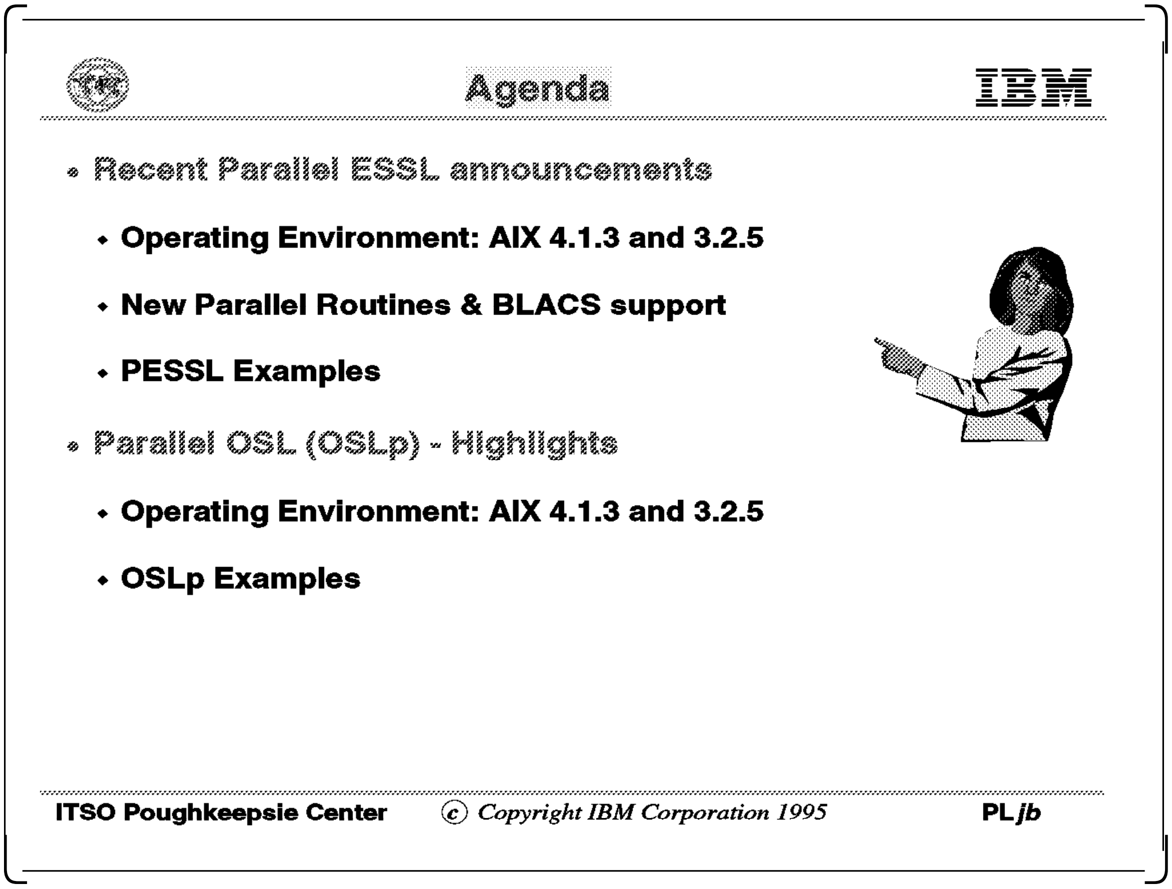
Parallel ESSL includes a new set of subroutines that allow the developer to develop parallel program applications.

Optimization Subroutine Library (OSL) is a set of subroutines developed to solve linear programs with real or integer variables. It is available on many platforms. The parallel version of OSL takes advantage of the distributed parallel structure of RS/6000 SP systems to solve very large linear programs.

This chapter is organized as follows:

- Parallel ESSL is described in Section 4.1, "Parallel ESSL" on page 84.
- Parallel OSL is presented in Section 4.2, "Parallel OSL" on page 111.

4.1 Parallel ESSL



Agenda

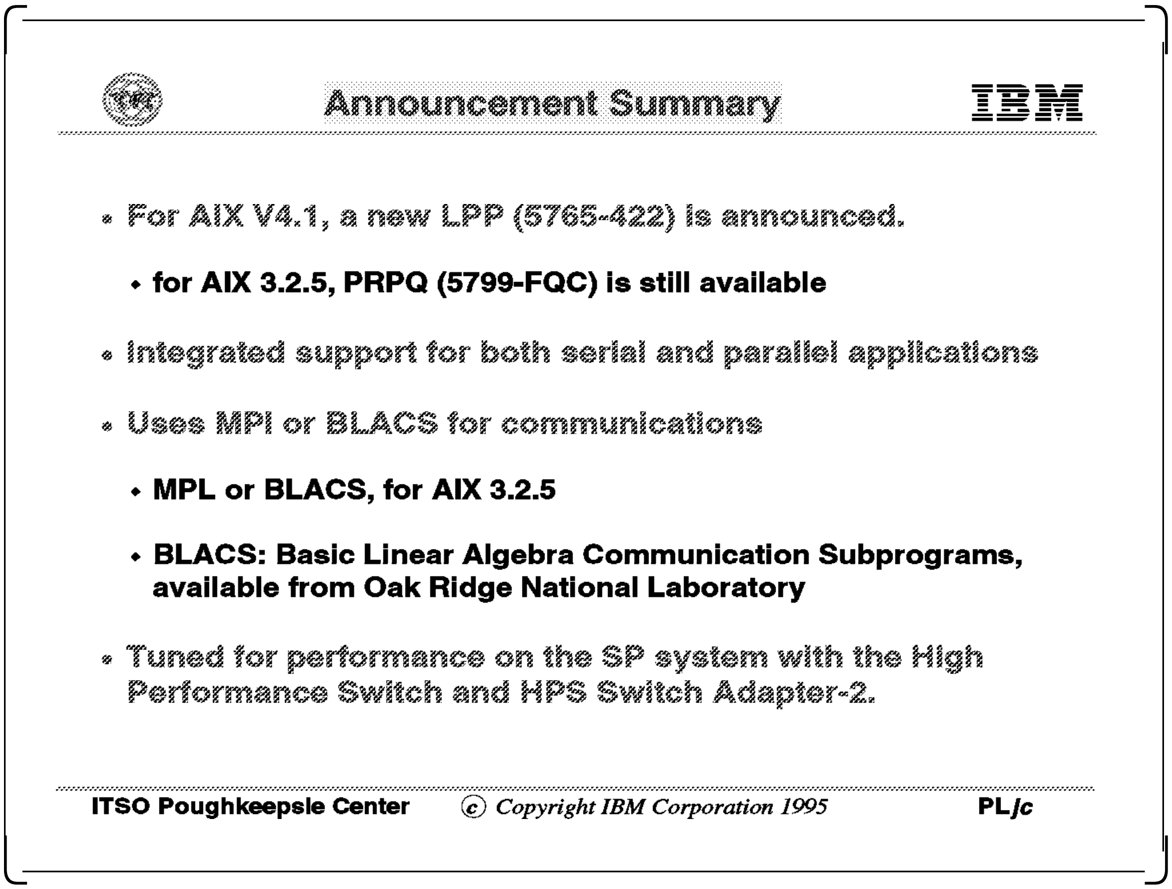
- Recent Parallel ESSL announcements
  - **Operating Environment: AIX 4.1.3 and 3.2.5**
  - **New Parallel Routines & BLACS support**
  - **PESSL Examples**
- Parallel OSL (OSLp) - Highlights
  - **Operating Environment: AIX 4.1.3 and 3.2.5**
  - **OSLp Examples**

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PL*jb***

The following topics will be covered in this section:

- The recent Parallel Engineering and Scientific Subroutine Library for AIX Version 4.1 (Parallel ESSL) announcements and its operating environment
- The new routines that support parallelism and the high-level communications subsystem (BLACS)
- Some examples of Parallel ESSL code
- Highlights of the Parallel Optimization subroutine Library (OSLp) product, and its operating environment
- Some examples of OSLp code

4.1.1 Announcement Summary



Announcement Summary

- For AIX V4.1, a new LPP (5765-422) is announced.
  - for AIX 3.2.5, PRPQ (5799-FQC) is still available
- Integrated support for both serial and parallel applications
- Uses MPI or BLACS for communications
  - MPL or BLACS, for AIX 3.2.5
  - BLACS: Basic Linear Algebra Communication Subprograms, available from Oak Ridge National Laboratory
- Tuned for performance on the SP system with the High Performance Switch and HPS Switch Adapter-2.

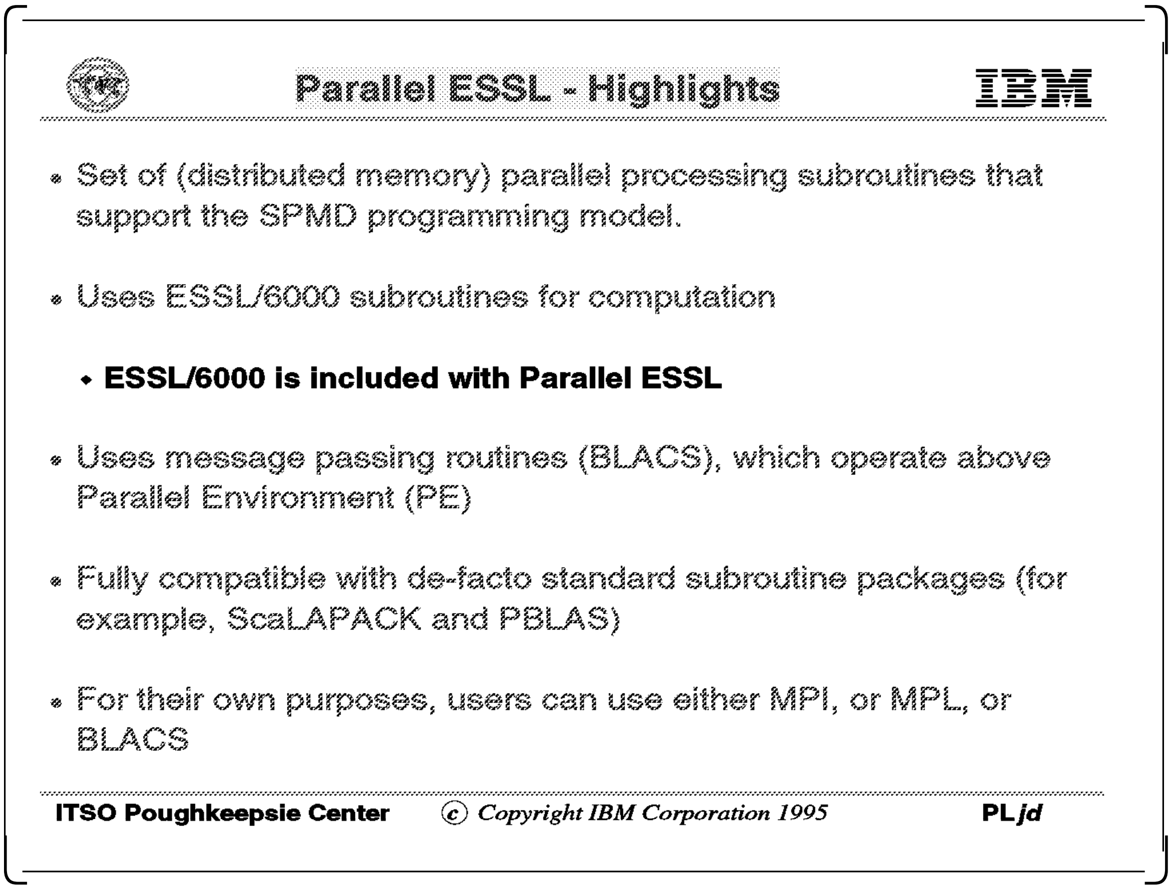
ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL/c

Parallel ESSL improves the performance of engineering and scientific applications on RS/6000 Scalable POWERparallel Systems by using multiple processors instead of single processors. Additional benefits include:

- Provides a parallel library tuned for performance on the SP with the High Performance Switch Adapter-2
- Includes the ESSL/6000 product as part of Parallel ESSL
- Allows licensing on a subset of your RS/6000 SP system
- supports the SPMD programming model under the IBM Parallel Environment (PE)
- Message Passing Interface (MPI) is used for communications through BLACS
- Fully compatible with de facto standard subroutine packages, for example, ScaLAPACK and PBLAS
- Callable from Fortran, C, and C++

For RISC System/6000 Scalable POWERparallel Systems still using AIX Version 3.2.5 and PSSP Version 1.2, the Parallel ESSL PRPQ (5799-FQC) is still available.

4.1.2 Parallel ESSL Highlights



Parallel ESSL - Highlights

- Set of (distributed memory) parallel processing subroutines that support the SPMD programming model.
- Uses ESSL/6000 subroutines for computation
  - **ESSL/6000 is included with Parallel ESSL**
- Uses message passing routines (BLACS), which operate above Parallel Environment (PE)
- Fully compatible with de-facto standard subroutine packages (for example, ScaLAPACK and PBLAS)
- For their own purposes, users can use either MPI, or MPL, or BLACS

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PL*jd***

Parallel ESSL is a set of (distributed memory) parallel processing subroutines that have been designed to provide high performance for numerically intensive computing jobs running on the RISC System/6000 Scalable POWERparallel System.

Parallel ESSL is built on the ESSL/6000 product, which is part of this offering. When you license Parallel ESSL, you can take advantage of over 400 serial subroutines available in ESSL/6000. Both Parallel ESSL and ESSL/6000 accelerate applications by replacing comparable subroutines and inline code with high-performance, tuned subroutines. ESSL uses algorithms tailored to the specific operational characteristics of the RISC System/6000 processors.

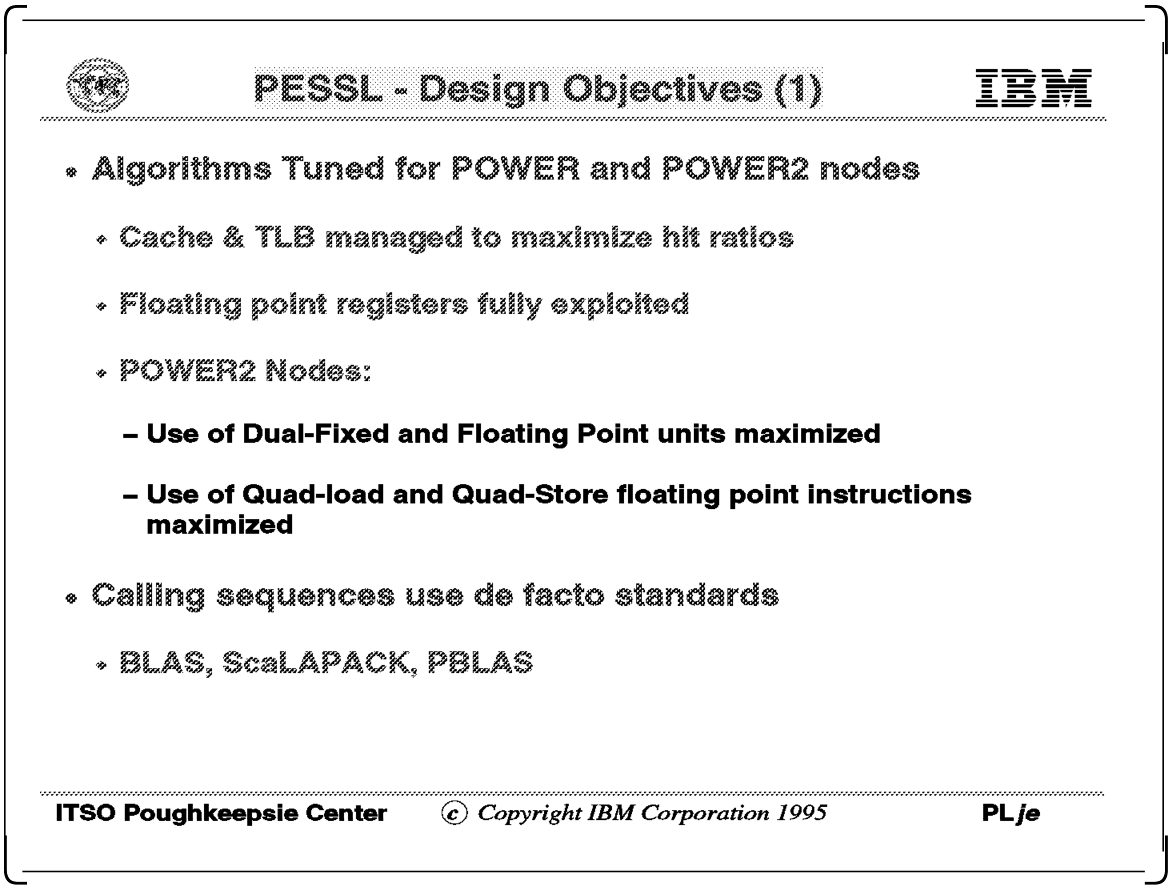
Parallel ESSL subroutines assume that your program uses the SPMD programming model where tasks running your parallel tasks on each processor are identical. The tasks, however work on different sets of data.

For interprocess communication, Parallel ESSL delivers the Basic Linear Algebra Communications subprograms (BLACS), which, in turn, use the AIX Parallel Environment (PE) Message Passing Interface (MPI).

For their own purposes, users can use either MPL, or MPI, or BLACS.

Note: The BLACS (de facto) standard was developed by the University of Tennessee and the Oak Ridge National Laboratory.

4.1.3 PESSL Design Objectives



PESSL - Design Objectives (1)

- Algorithms Tuned for POWER and POWER2 nodes
  - Cache & TLB managed to maximize hit ratios
  - Floating point registers fully exploited
  - POWER2 Nodes:
    - Use of Dual-Fixed and Floating Point units maximized
    - Use of Quad-load and Quad-Store floating point instructions maximized
- Calling sequences use de facto standards
  - BLAS, ScaLAPACK, PBLAS

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* PL*je*

Parallel ESSL was designed to exploit the strengths of IBM RISC System/6000 POWER and POWER2 processors, and the parallel subroutines exploit the high performance provided by ESSL/6000 Version 2.2 for use on each processor of a Parallel ESSL job. Most of the ESSL/6000 subroutines use the following techniques to optimize performance.

- Manage the cache and TLB efficiently so the hit ratios are maximized — that is, data is blocked so that it stays in the cache or TLB for its computation. This reduces the cost of accessing data (when it is found in the cache, which avoids a memory access), and the number of virtual to real memory translations (TLB data).
- Access data that is stored contiguously; that is, use stride 1 computations.
- Exploit the large number of available floating point registers.
- On POWER processors:
  - Use algorithms that balance floating operations with loads in the innermost loop.
  - Use algorithms that minimize stores in the innermost loop.
- On POWER2 processors:
  - Use algorithms that fully use the dual fixed-point and floating-point units. Two Multiply-Add instructions can be executed each cycle, neglecting

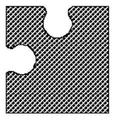
overhead. This allows execution of four floating-point operations per cycle.

– Use algorithms that fully exercise the load and store floating-point Quadword instructions. For example, in one cycle, two Load Floating-Point Quadword instructions can be executed. Neglecting overhead, this allows four doublewords to be loaded per cycle.

  The Quadword load instructions move two double-precision storage operands into two adjacent floating-point registers.

PESSL - Design Objectives (2)

- **Tuning for Parallel Processing on SP Systems**
  - **Minimizing communications: exchange large blocks of data**
  - **Block data relative to processor cache sizes**
  - **User-defined:**
    - **Data block sizes**
    - **Processor grids**
    - **Processor types**



ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* **PL*jf***

Tuning to optimize the parallel algorithms further increases the actual throughput capabilities of the Parallel ESSL library. These tuning design points include the following:

- Minimizing the impact of communications by exchanging large blocks of data when possible.
- Blocking data to match the processor cache size.
- Permitting programmer experimentation to obtain favorable block sizes for the distribution of problem data.
- Permitting programmer experimentation to determine favorable processor grid dimensions for a variety of problem sizes and types.
- Parallel ESSL can be installed on combinations of POWERparallel thin and wide nodes.

Shape of the Process Grid:

- For most subroutines, programmers are urged to use a two-dimensional (square or as close to square as possible) grid.

Number and Types of Processor Nodes:

- The optimal number of processors depends primarily on the program size.
- It is reasonable to increase the number of processors if the problem size increases sufficiently to keep the amount of local data per process at a reasonable size.

- When users increase the number of processors, they must keep in mind the shape of the process grid. For example, using 17 processors instead of 16, would most likely result in performance degradation.

Blocksize:

- The optimal blocksize depends on underlying node computations, load balancing, communications, system buffering requirements, problem size, and dimension and shape of the process grid.
- Blocksize specifications require experimentation to achieve optimal performance.

  The following values provide good performance in most cases:

  – POWER nodes:

    - 24 for Level 2 PBLAS, eigensystems and singular value analysis
    - 40 for Level 3 PBLAS and linear algebraic equations
    - $\frac{\mathit{data\_cache\_size}}{2}$ for random numbers

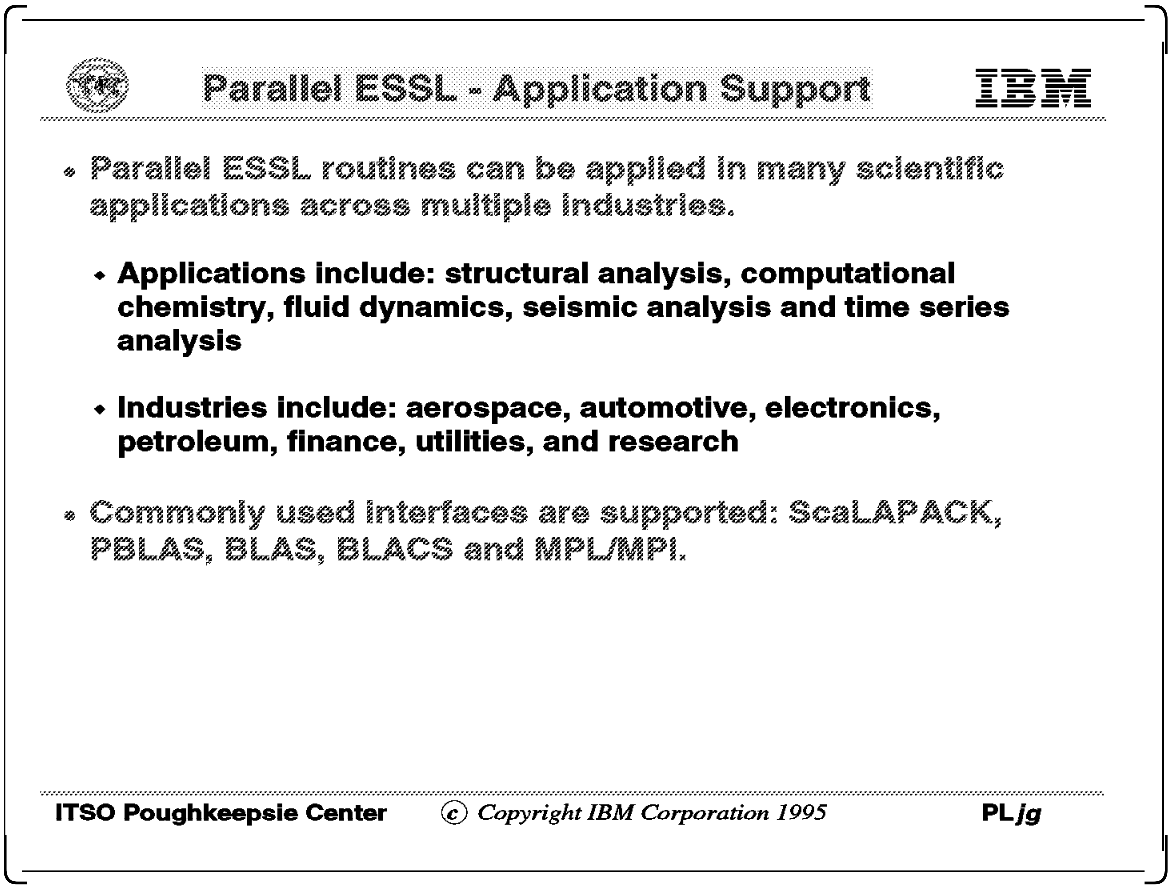
  – POWER2 nodes:

    - 24 for Level 2 PBLAS, eigensystems and singular value analysis
    - 70 for Level 3 PBLAS and linear algebraic equations
    - $\frac{\mathit{data\_cache\_size}}{2}$ for random numbers

Note: The data cache size can be obtained with this command:

```
lsattr -E -H -l sys0
```

4.1.4 PESSL Application Support



Parallel ESSL - Application Support

- Parallel ESSL routines can be applied in many scientific applications across multiple industries.
  - **Applications include: structural analysis, computational chemistry, fluid dynamics, seismic analysis and time series analysis**
  - **Industries include: aerospace, automotive, electronics, petroleum, finance, utilities, and research**
- Commonly used interfaces are supported: ScaLAPACK, PBLAS, BLAS, BLACS and MPL/MPI.

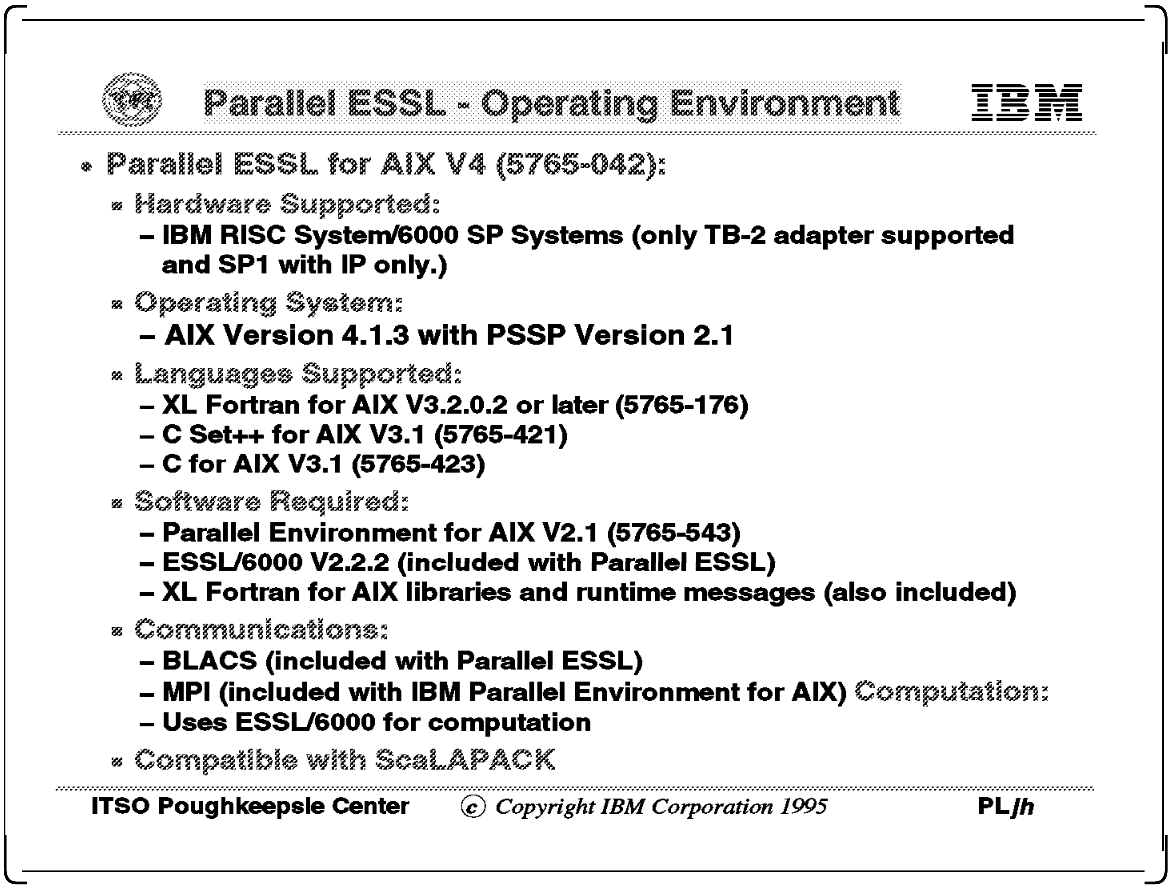
ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PL*jg***

Parallel ESSL can provide solutions to problems in many scientific and technical applications. A number of enhancements have been implemented that make Parallel ESSL easy to use. These include:

- Comprehensive, high-quality documentation that is usable by a wide class of programmers.
- Validity checks for parameters, when technically possible, and when the impact to performance is not significant.
- Calling sequences compatible with existing ESSL conventions, conventions of widely used subroutine libraries, and conventions familiar to typical users. In the areas of Linear Algebraic Equations, Parallel BLAS, and Eigensystem Analysis, the calling sequences match those of ScaLAPACK.

Note: ScaLAPACK extends the LAPACK library to run scalably on MIMD, distributed memory, concurrent computers. The ScaLAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy. The fundamental building blocks of the ScaLAPACK library are distributed memory versions (PBLAS) of the Level 1, 2 and 3 BLAS, and a set of Basic Linear Algebra Communication subprograms (BLACS) for communication tasks that arise frequently in parallel linear algebra computations. More information on ScaLAPACK can be obtained from the `netlib` homepage on the World Wide Web at `http://www.netlib.org/scalapack/index.html`.

4.1.5 PESSL Operating Environments



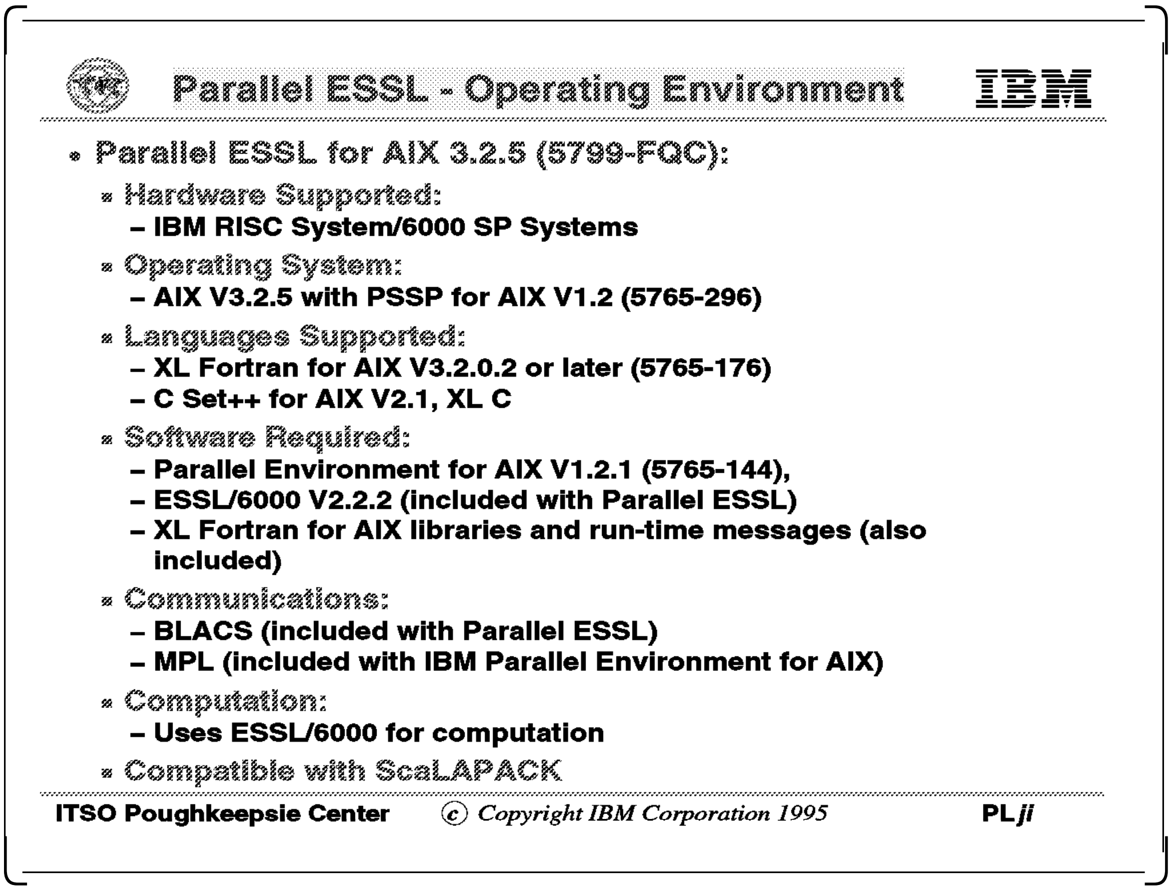
Parallel ESSL - Operating Environment

- Parallel ESSL for AIX V4 (5765-042):
  - Hardware Supported:
    - IBM RISC System/6000 SP Systems (only TB-2 adapter supported and SP1 with IP only.)
  - Operating System:
    - AIX Version 4.1.3 with PSSP Version 2.1
  - Languages Supported:
    - XL Fortran for AIX V3.2.0.2 or later (5765-176)
    - C Set++ for AIX V3.1 (5765-421)
    - C for AIX V3.1 (5765-423)
  - Software Required:
    - Parallel Environment for AIX V2.1 (5765-543)
    - ESSL/6000 V2.2.2 (included with Parallel ESSL)
    - XL Fortran for AIX libraries and runtime messages (also included)
  - Communications:
    - BLACS (included with Parallel ESSL)
    - MPI (included with IBM Parallel Environment for AIX) Computation:
    - Uses ESSL/6000 for computation
  - Compatible with ScaLAPACK

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL*jh*

This is a summary of the software required to install and use Parallel ESSL on a RISC System/6000 Scalable POWERparallel System in an AIX V4.1 environment. These products must be on every node where it is intended to use Parallel ESSL.

Note: Only The XL Fortran libraries and messages are required by Parallel ESSL. If applications are to be developed in Fortran, then the XL Fortran compiler is also required. These libraries and messages can be shipped as a separate feature of Parallel ESSL.



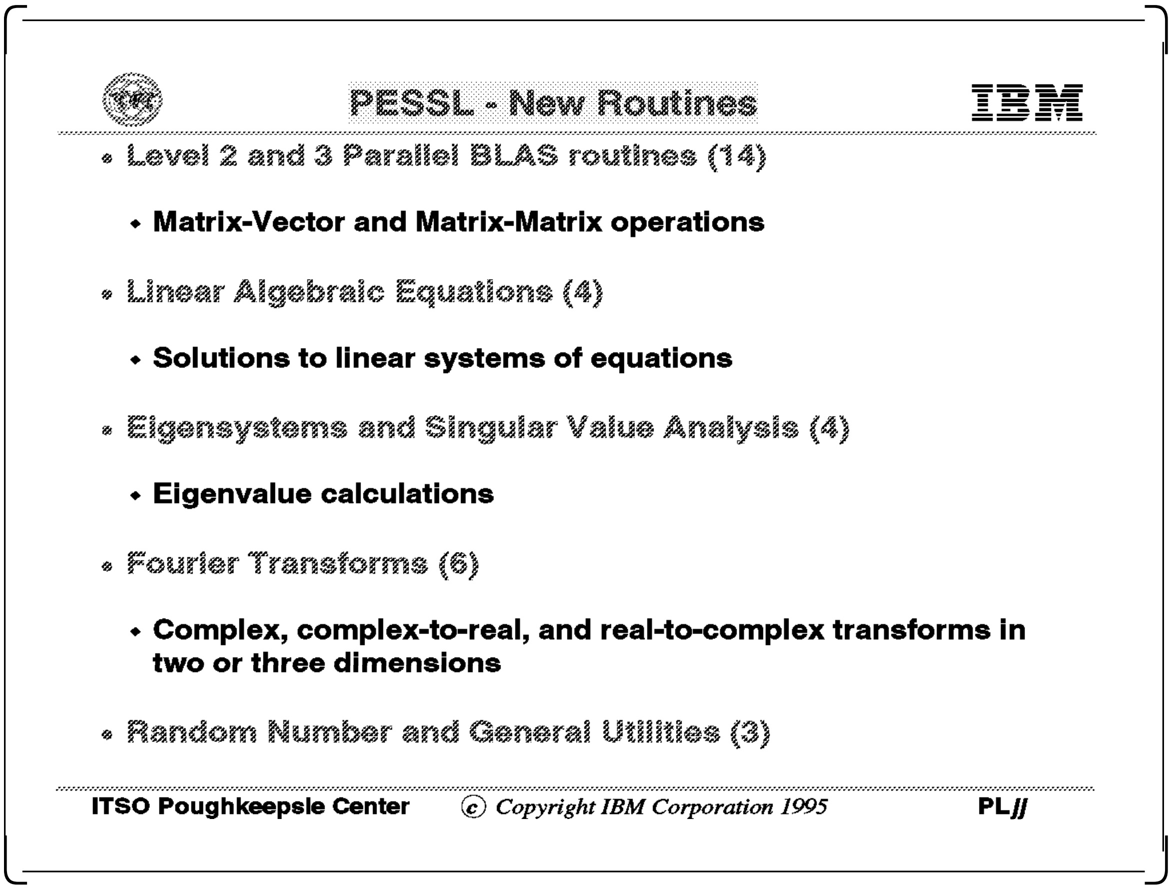
Parallel ESSL - Operating Environment

- Parallel ESSL for AIX 3.2.5 (5799-FQC):
  - Hardware Supported:
    - IBM RISC System/6000 SP Systems
  - Operating System:
    - AIX V3.2.5 with PSSP for AIX V1.2 (5765-296)
  - Languages Supported:
    - XL Fortran for AIX V3.2.0.2 or later (5765-176)
    - C Set++ for AIX V2.1, XL C
  - Software Required:
    - Parallel Environment for AIX V1.2.1 (5765-144),
    - ESSL/6000 V2.2.2 (included with Parallel ESSL)
    - XL Fortran for AIX libraries and run-time messages (also included)
  - Communications:
    - BLACS (included with Parallel ESSL)
    - MPL (included with IBM Parallel Environment for AIX)
  - Computation:
    - Uses ESSL/6000 for computation
  - Compatible with ScaLAPACK

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* PL*ji*

This foil presents the information related to the Parallel ESSL version supported by AIX 3.2.5 and PSSP 1.2.

4.1.6 Parallel ESSL - New Routines



PESSL - New Routines

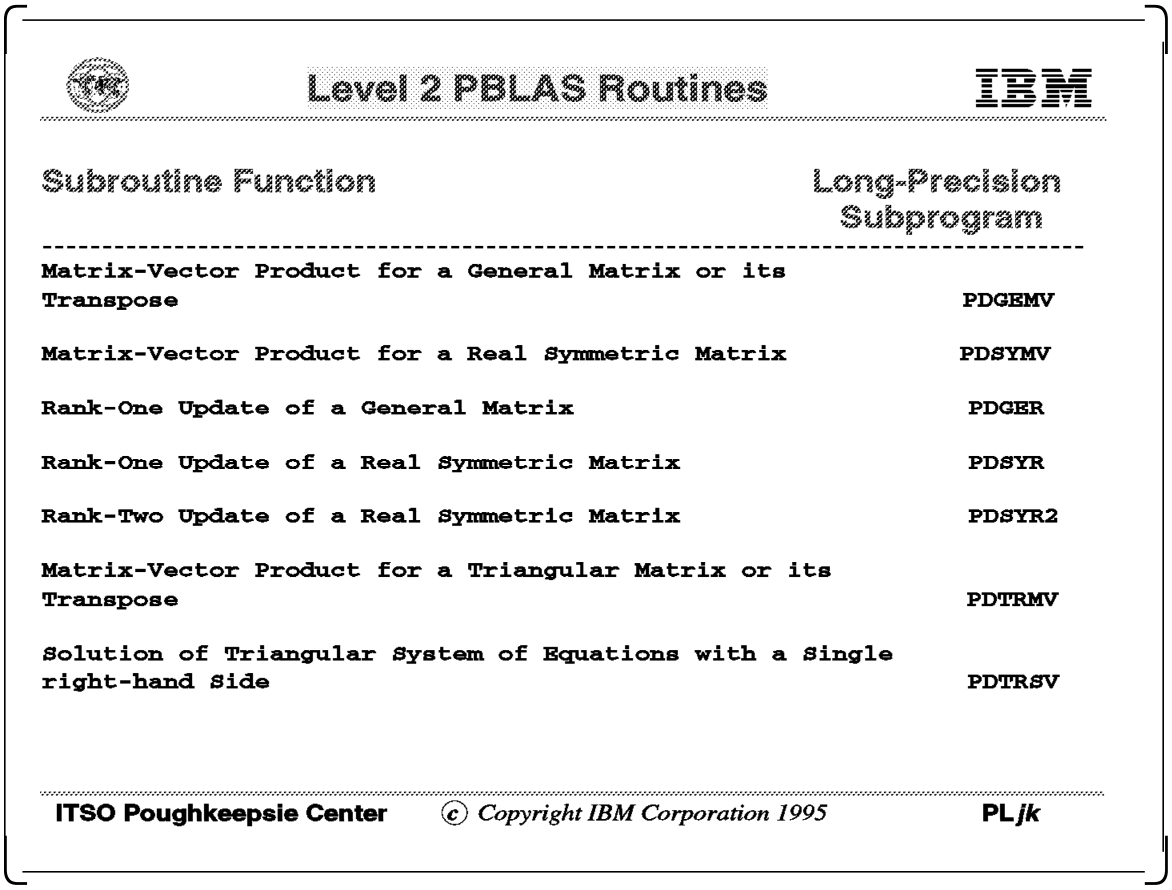
- Level 2 and 3 Parallel BLAS routines (14)
  - Matrix-Vector and Matrix-Matrix operations
- Linear Algebraic Equations (4)
  - Solutions to linear systems of equations
- Eigensystems and Singular Value Analysis (4)
  - Eigenvalue calculations
- Fourier Transforms (6)
  - Complex, complex-to-real, and real-to-complex transforms in two or three dimensions
- Random Number and General Utilities (3)

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   PLJJ

Discussions with marketing and IBM customers resulted in the delivery of the following parallel subroutines into the Parallel ESSL product:

- Level 2 Parallel BLAS Routines. These routines involve matrix-vector operations.
- Level 3 Parallel BLAS Routines. These routines involve matrix-matrix operations.
- Linear Algebraic Equations. These subroutines provide solutions to linear systems of equations for real general matrices and real symmetric positive definite matrices.
- Eigensystems and Singular Value Analysis. Parallel eigensystem analysis subroutines provide solutions to the algebraic eigensystem analysis problem for real symmetric matrices and the capability to reduce real symmetric and real general matrices to condensed form.
- Fourier Transforms, Convolutions and Correlations. Parallel ESSL Fourier transforms subroutines perform mixed-radix complex, complex-to-real, and real-to-complex transforms in two and three dimensions.
- The Random number generation subroutine generates uniformly distributed random numbers. Utility subroutines perform general service functions that support Parallel ESSL rather than perform mathematical computations.

4.1.6.1 PBLAS Level 2 Routines



Level 2 PBLAS Routines

Subroutine Function	Long-Precision Subprogram
Matrix-Vector Product for a General Matrix or its Transpose	PDGEMV
Matrix-Vector Product for a Real Symmetric Matrix	PDSYMV
Rank-One Update of a General Matrix	PDGER
Rank-One Update of a Real Symmetric Matrix	PDSYR
Rank-Two Update of a Real Symmetric Matrix	PDSYR2
Matrix-Vector Product for a Triangular Matrix or its Transpose	PDTRMV
Solution of Triangular System of Equations with a Single right-hand Side	PDTRSV

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PL*jk*

PDGEMV: This subroutine performs one of the matrix-vector products:

$$y{:=}\alpha*A*x+\beta*y$$
or
$$y{:=}\alpha*A'*x+\beta*y$$

where α and β are scalars, x and y are vectors and A is a general matrix.

PDSYMV: This subroutine performs the matrix-vector product:

$$y{:=}\alpha*A*x+\beta*y$$

where α and β are scalars, x and y are vectors and A is a symmetric matrix.

PDGER: This subroutine performs the rank 1 update:

$$A{:=}\alpha*x*y'+A$$

where α is a scalar, x and y are vectors and A is a general matrix.

PDSYR: This subroutine performs the rank 1 update:

$$A{:=}\alpha*x*x'+A$$

where α is a scalar, x is a vector and A is a symmetric matrix.

PDSYR2: This subroutine performs the rank 2 update:

$$A{:=}\alpha*x*y'+\alpha*y*x'+A$$

where α is a scalar, x and y are vectors and A is a symmetric matrix.

PDTRMV: This subroutine performs one of the matrix-vector products:

```
x:=A*x
or
x:= A′*x
```

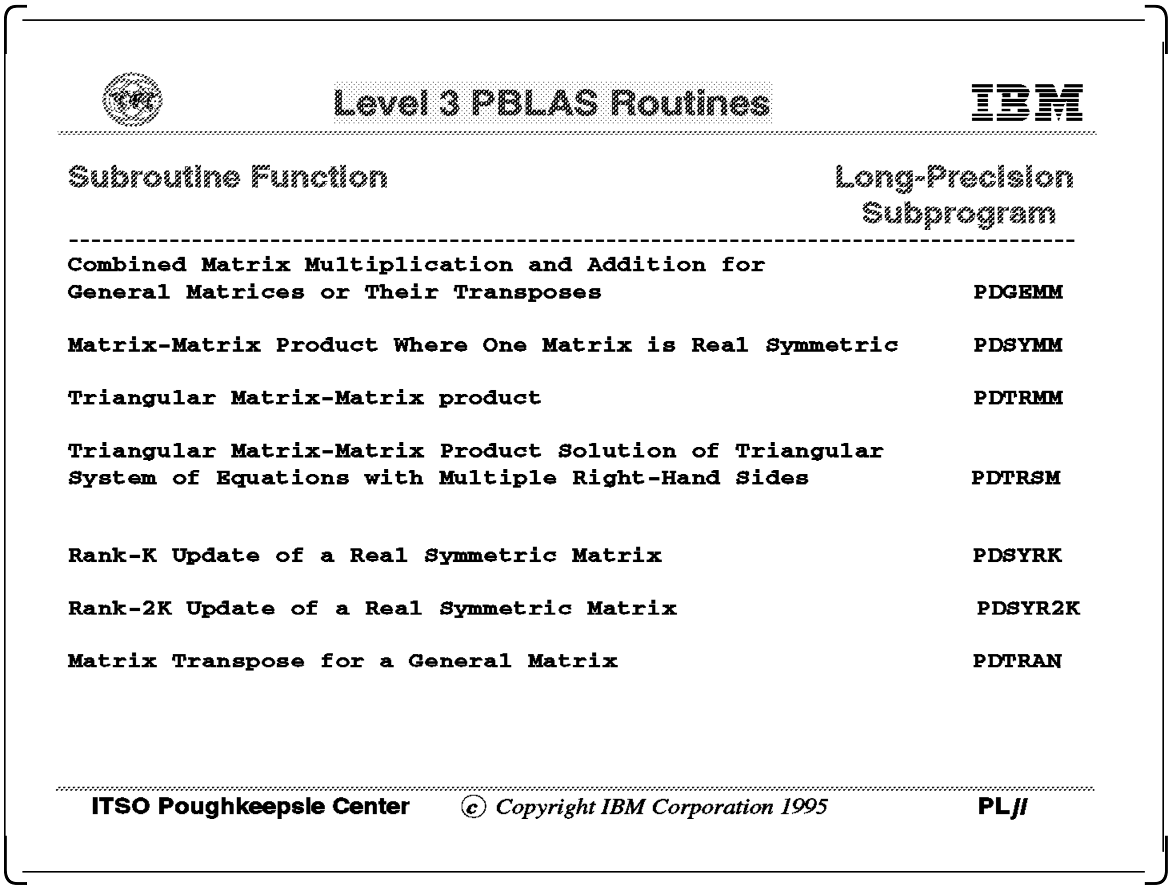
where x is a vector and A is a unit, or non-unit, upper or lower triangular matrix.

PDTRSV: This subroutine performs one of the following solves for a triangular system of equations with a single right-hand side:

```
A′x=b
  or
Ax=b
```

where x and b are vectors, and A is a unit, or non-unit, upper or lower triangular matrix.

4.1.6.2 Level 3 PBLAS Routines



Level 3 PBLAS Routines

Subroutine Function	Long-Precision Subprogram
Combined Matrix Multiplication and Addition for General Matrices or Their Transposes	PDGEMM
Matrix-Matrix Product Where One Matrix is Real Symmetric	PDSYMM
Triangular Matrix-Matrix product	PDTRMM
Triangular Matrix-Matrix Product Solution of Triangular System of Equations with Multiple Right-Hand Sides	PDTRSM
Rank-K Update of a Real Symmetric Matrix	PDSYRK
Rank-2K Update of a Real Symmetric Matrix	PDSYR2K
Matrix Transpose for a General Matrix	PDTRAN

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PL/I

PDGEMM: This subroutine performs one of the matrix-matrix products:

$$C:=\alpha*A*B+\beta*C$$
or
$$C:=\alpha*A*B'+\beta*C$$
or
$$C:=\alpha*A'*B+\beta*C$$
or
$$C:=\alpha*A'*B'+\beta*C$$

where α and β are scalars, and A, B, and C are general matrices.

PDSYMM: This subroutine performs one of the matrix-matrix products:

$$C:=\alpha*A*B+\beta*C$$
or
$$C:=\alpha*B*A+\beta*C$$

where α and β are scalars, A is a symmetric matrix and B and C are general matrices.

PDTRMM: This subroutine performs one of the matrix-matrix products:

```
B:= α*A*B
      or
B:= α*B*A
      or
B:= α*A′*B
      or
B:= α*B*A′
```

where α is a scalar, B is a general matrix and A is a unit, or non-unit, upper or lower triangular matrix.

PDTRSM: This subroutine performs one of the following solves for a triangular system of equations with multiple right hand sides:

```
A*X  = α*B
    or
X*A  = α*B
    or
A′*X = α*B
    or
X*A′ = α*B
```

where α is a scalar, X and B are general matrices, and A is a unit or non-unit, upper or lower triangular matrix.

PDSYRK: This subroutines performs one of the rank k updates:

```
C:=α*A*A′+β*C
        or
C:=α*A′*A+β*C
```

where α and β are scalars, C is a symmetric matrix and A is a general matrix.

PDSYR2K: This subroutine performs one of the rank 2-k updates:

```
C:= α*A*B′+α*B*A′+β*C
              or
C:= α*A′*B+α*B′*A+β*C
```

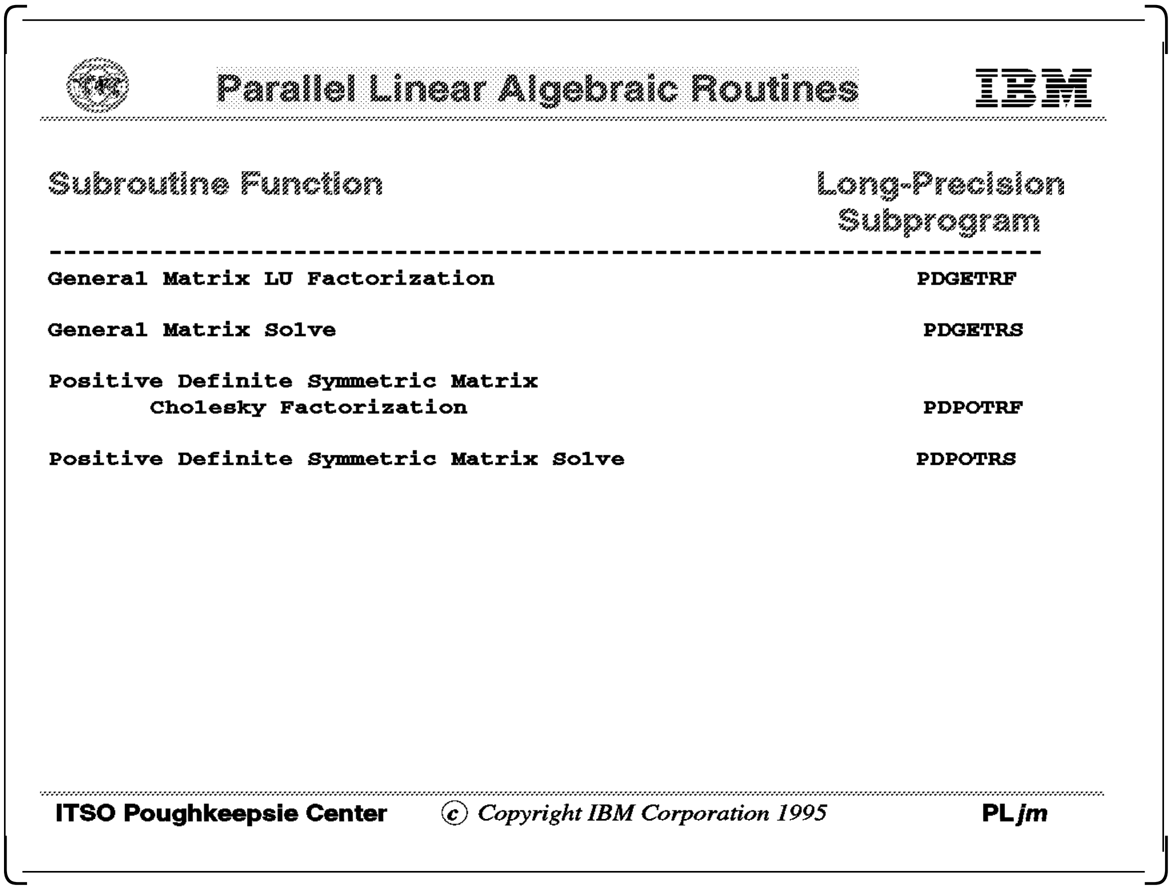
where α and β are scalars, C is a symmetric matrix and A and B are general matrices.

PDTRAN: This subroutine performs the following matrix computation:

```
C:= βC+αA′
```

where α and β are scalars and A and C are general matrices.

4.1.6.3 Parallel Linear Algebraic Routines



Parallel Linear Algebraic Routines

Subroutine Function	Long-Precision Subprogram
General Matrix LU Factorization	PDGETRF
General Matrix Solve	PDGETRS
Positive Definite Symmetric Matrix Cholesky Factorization	PDPOTRF
Positive Definite Symmetric Matrix Solve	PDPOTRS

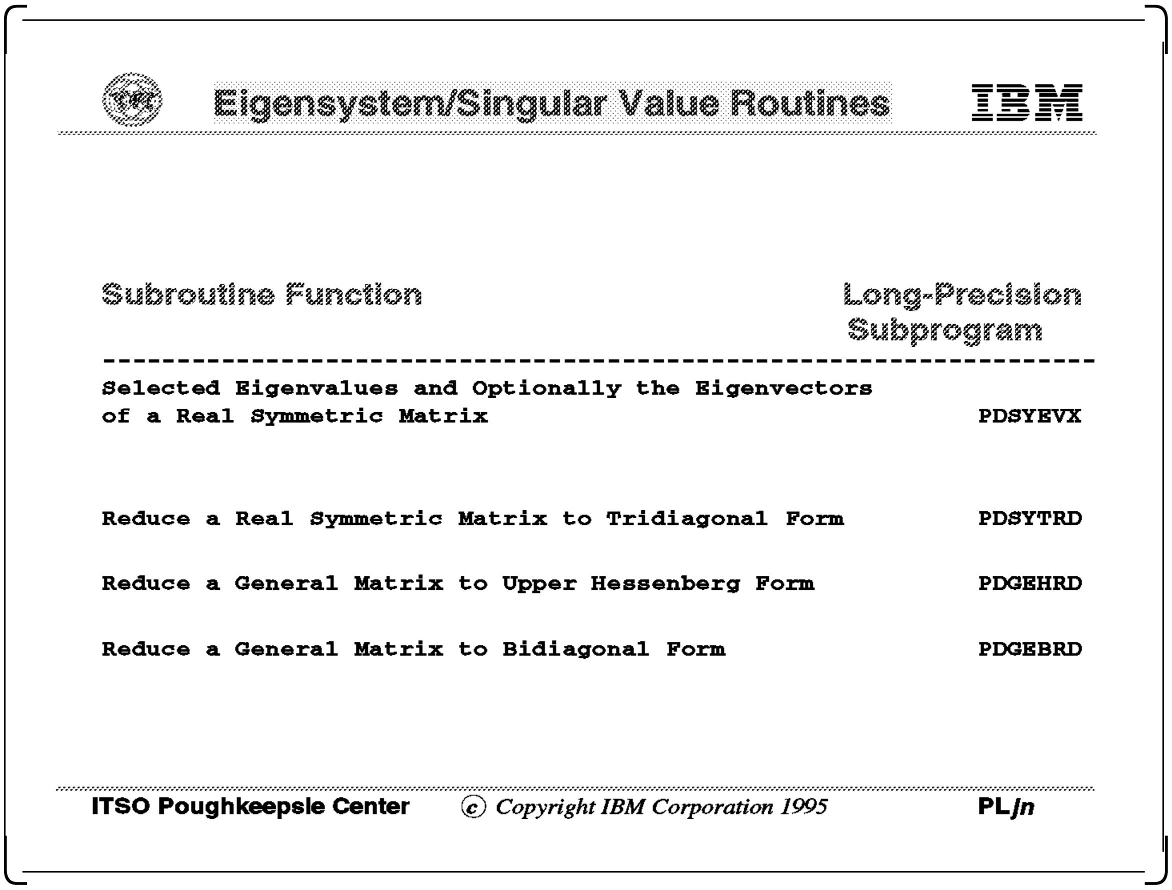
ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PLjm

Parallel Linear Algebra Equation subroutines provide solutions to linear systems of equations for real general matrices and real symmetric positive definite matrices.

PDGETRF and PDGETRS: PDGETRF computes an LU factorization of a general matrix A using Gaussian elimination with partial pivoting. PDGETRS solves a system of equations, AX=B or A′X=B, with general matrices A and B using the LU factorization computed by a prior call to PDGETRF.

PDPOTRF and PDPOTRS: PDPOTRF factors a real, positive definite, symmetric matrix by the Cholesky factorization method. PDPOTRS solves a system of equations, AX=B, where A is a positive definite, symmetric matrix using the Cholesky factorization computed by a previous call to PDPOTRF.

4.1.6.4 Eigensystem/Singular Value Routines



Eigensystem/Singular Value Routines

Subroutine Function	Long-Precision Subprogram
Selected Eigenvalues and Optionally the Eigenvectors of a Real Symmetric Matrix	PDSYEVX
Reduce a Real Symmetric Matrix to Tridiagonal Form	PDSYTRD
Reduce a General Matrix to Upper Hessenberg Form	PDGEHRD
Reduce a General Matrix to Bidiagonal Form	PDGEBRD

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PL/jn

PDSYEVX: This subroutine computes selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix A:

$$Az = wz \quad \text{where } A = A'$$

PDSYTRD: This subroutine reduces a real symmetric matrix A to symmetric tri-diagonal form T by an orthogonal similarity transformation:

$$T = Q' * A * Q$$

PDGEHRD: This subroutine reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation:

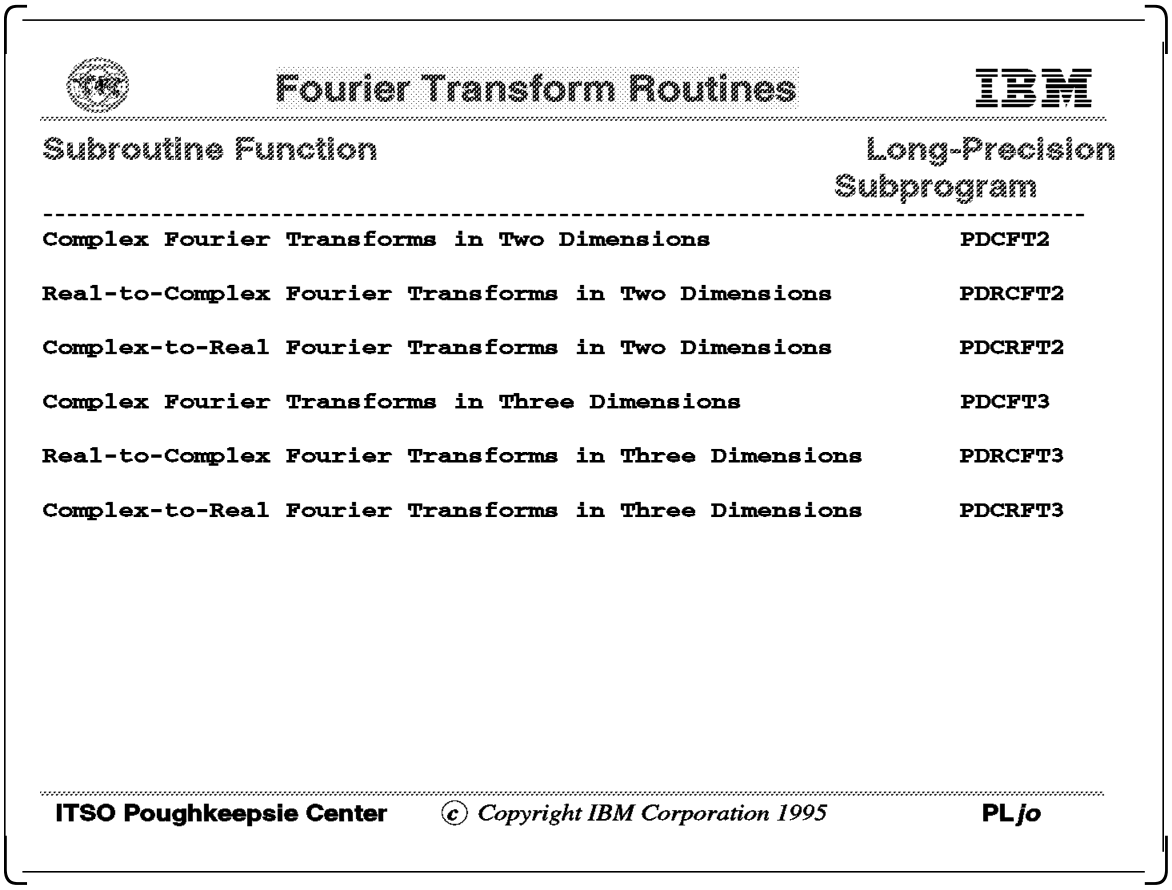
$$H = Q' * A * Q$$

Note: An upper matrix is one with non-zero elements in the upper triangle and the first subdiagonal.

PDGEBRD: This subroutine reduces a real general matrix A to upper or lower bi-diagonal form B by an orthogonal transformation:

$$B = Q' * A * P$$

4.1.6.5 Fourier Transform Routines



Fourier Transform Routines

Subroutine Function	Long-Precision Subprogram
Complex Fourier Transforms in Two Dimensions	PDCFT2
Real-to-Complex Fourier Transforms in Two Dimensions	PDRCFT2
Complex-to-Real Fourier Transforms in Two Dimensions	PDCRFT2
Complex Fourier Transforms in Three Dimensions	PDCFT3
Real-to-Complex Fourier Transforms in Three Dimensions	PDRCFT3
Complex-to-Real Fourier Transforms in Three Dimensions	PDCRFT3

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PLjo

The following FFTs are available as parallel subroutines.

PDCFT2: This subroutine computes the two-dimensional discrete Fourier transform of long precision complex data.

PDRCFT2: This subroutine computes the two-dimensional complex-conjugate even discrete Fourier transform of real data. The output data has complex conjugate even format.

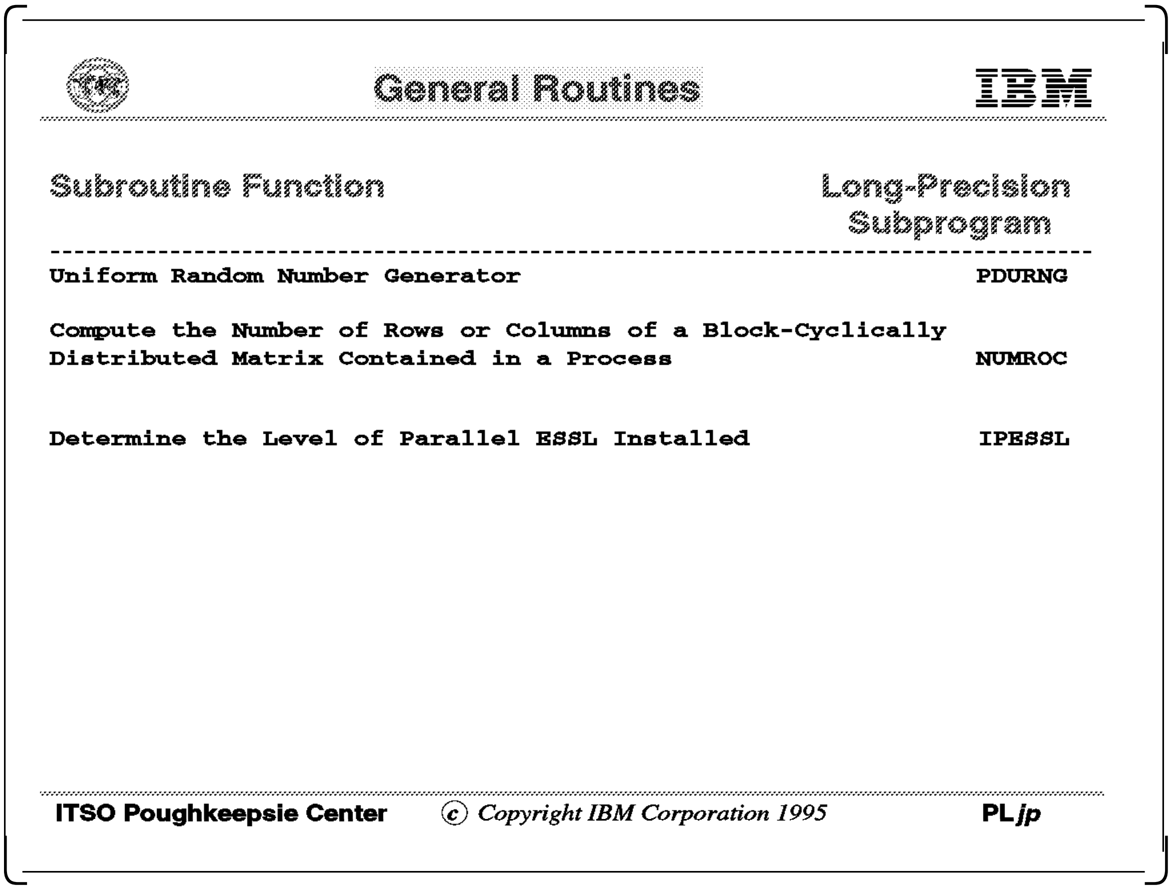
PDCRFT2: This subroutine computes the two-dimensional real discrete Fourier transform of long precision complex-conjugate even data. The input data is stored in compact format, taking advantage of the symmetry of the input data. The output data is long precision real.

PDCFT3: This subroutine computes the three-dimensional discrete Fourier transform of long precision complex data.

PDRCFT3: This subroutine computes the three-dimensional complex-conjugate even discrete Fourier transform of real data. The output data has complex-conjugate even format.

PDCRFT3: This subroutine computes the three-dimensional real discrete Fourier transform of long precision complex-conjugate even data. The input data is stored in compact format, taking advantage of the symmetry of the input data. The output data is long precision real.

4.1.6.6 General Routines



General Routines

Subroutine Function	Long-Precision Subprogram
Uniform Random Number Generator	PDURNG
Compute the Number of Rows or Columns of a Block-Cyclically Distributed Matrix Contained in a Process	NUMROC
Determine the Level of Parallel ESSL Installed	IPESSL

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PLjp

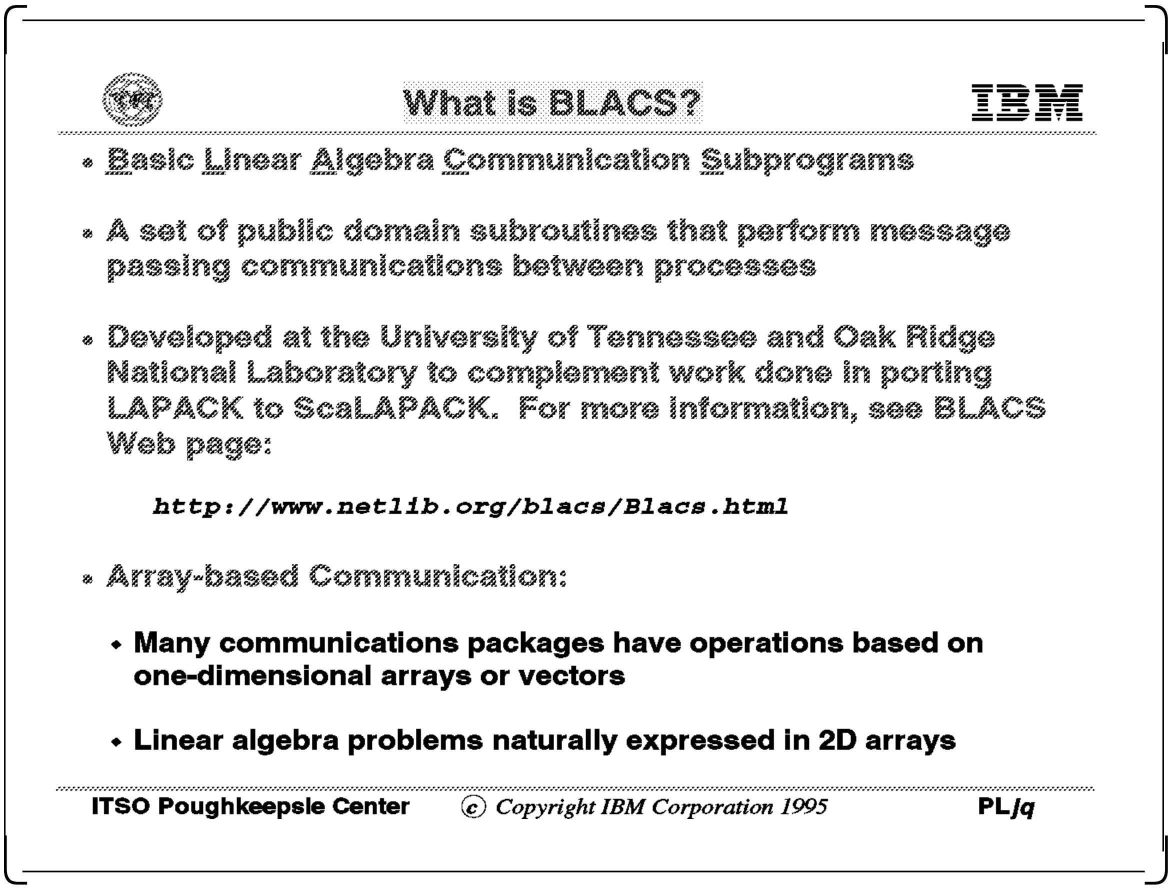
PDURNG: PDURNG generates a vector of uniform pseudo-random numbers in the ranges [0,1] or [-1,1]. The random numbers are generated using the multiplicative congruential method with a user-specified seed.

NUMROC: NUMROC computes the number of rows or columns of a block cyclically distributed matrix contained in any one process.

Note: Block distribution breaks up a matrix into blocks of data, whereby each processor participating in a parallel computation handles only one block of matrix elements. This is appropriate if matrix element computations involve neighboring elements. Cyclic distribution breaks up the matrix into strips, and each processor participating in a parallel computation handles several strips of data. This is commonly used to provide better load balancing.

IPESSL: IPESSL is a query utility which returns the Parallel ESSL version number, release number, modification number, and fix number (for the last PTF installed).

4.1.7 What is BLACS?



What is BLACS?

- Basic Linear Algebra Communication Subprograms
- A set of public domain subroutines that perform message passing communications between processes
- Developed at the University of Tennessee and Oak Ridge National Laboratory to complement work done in porting LAPACK to ScaLAPACK. For more information, see BLACS Web page:

  http://www.netlib.org/blacs/Blacs.html
- Array-based Communication:
  - **Many communications packages have operations based on one-dimensional arrays or vectors**
  - **Linear algebra problems naturally expressed in 2D arrays**

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* **PL*jq***

The Basic Linear Algebra Communication subprograms (BLACS) package is a linear algebra oriented message passing interface that is implemented efficiently and uniformly across a large range of distributed memory platforms. The BLACS makes linear algebra programs both easier to write and more portable. It was developed jointly at the University of Tennessee and the Oak Ridge National Laboratory as the communications layer for the ScaLAPACK project, which involved implementing the LAPACK library on distributed memory MIMD machines

The BLACS was written specifically for linear algebra programming. Many communication packages (including IBM′s MPL and the new MPI standard) can be classified as having operations based on one dimensional arrays or vectors; that is, they require an address and length to be sent. In programming linear algebra problems, however, it is more natural to express all operations in terms of matrices. Vectors and scalars are, of course, simply subclasses of matrices. On computers, a linear algebra matrix is represented by a two-dimensional array (2D array), and therefore the BLACS operates on 2D arrays.

One of the main strengths of the BLACS is that code that uses the BLACS for its communication layer can run unchanged on any supported platform. The BLACS has been written on top of the following message passing layers:

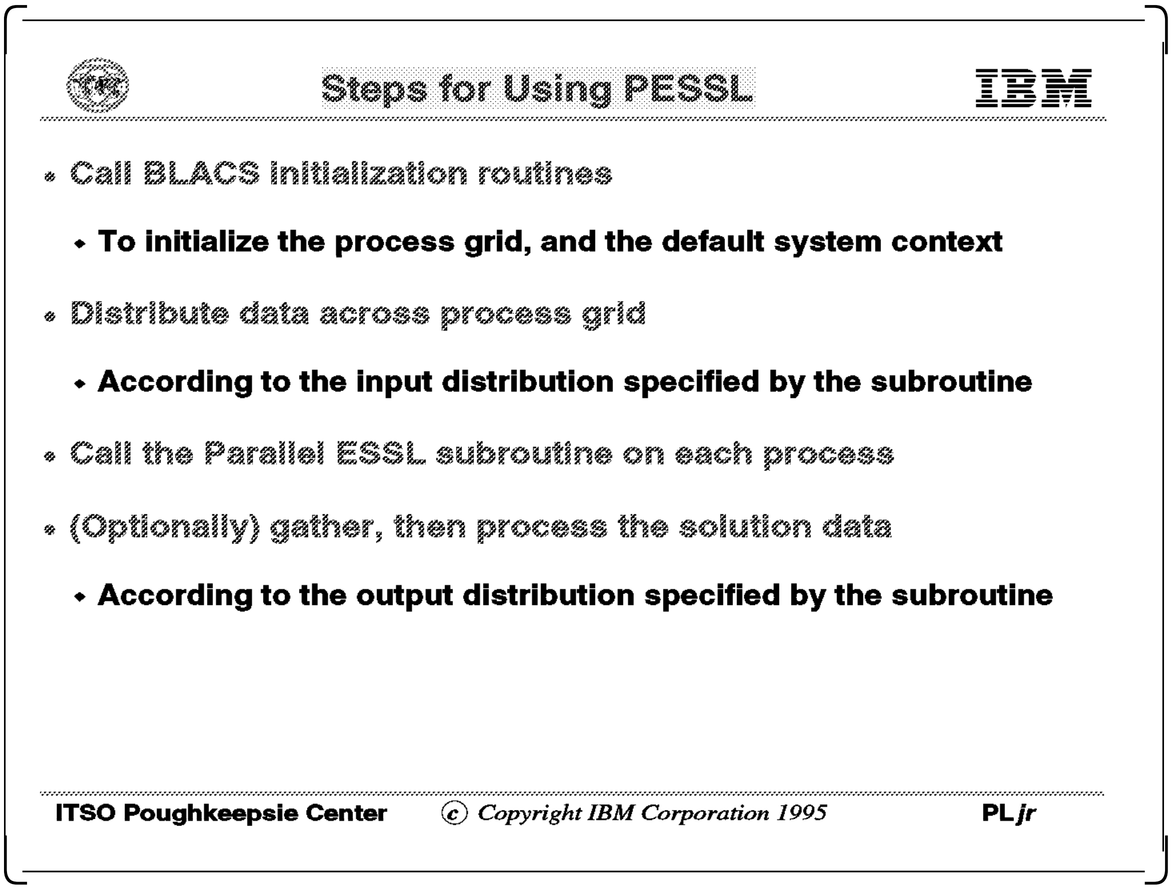
Message Passing Layer	**Machines**
CMMD	Thinking Machine′s CM-5.

MPL/MPI	IBM′s RISC System/6000 series.
NX	Intel′s supercomputer series (iPSC2, iPSC860, DELTA and PARAGON).
PVM	Anywhere PVM is supported, which includes most UNIX systems.

Parallel ESSL includes both the BLACS and the ESSL/6000 BLAS, compatible with the public domain interfaces; therefore, any applications that call these can be used compatibly with Parallel ESSL.

More information on the BLACS can be obtained from the BLACS Web page at `http://www.netlib.org/blacs/Blacs.html`, maintained by the Oak Ridge National Laboratory.

4.1.8 Steps for Using PESSL



Steps for Using PESSL

IBM

- Call BLACS initialization routines
  - To initialize the process grid, and the default system context
- Distribute data across process grid
  - According to the input distribution specified by the subroutine
- Call the Parallel ESSL subroutine on each process
- (Optionally) gather, then process the solution data
  - According to the output distribution specified by the subroutine

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* PL*jr*

- Call the BLACS initialization subroutines. These routines (BLACS_PINFO, BLACS_GET, and BLACS_GRIDINIT or BLACS_GRIDMAP) define the following:

  1. The number of processes (nodes) available.
  2. The default system context; the context can be imagined to be a "message passing universe," in which all processes involved in a particular parallel computation can communicate safely with each other.
  3. The size of the process grid, which is a mapping of the nodes (processes) onto a "process grid," which is a 1- or 2-dimensional representation of the available nodes; the grid is defined to be as close to square as possible.

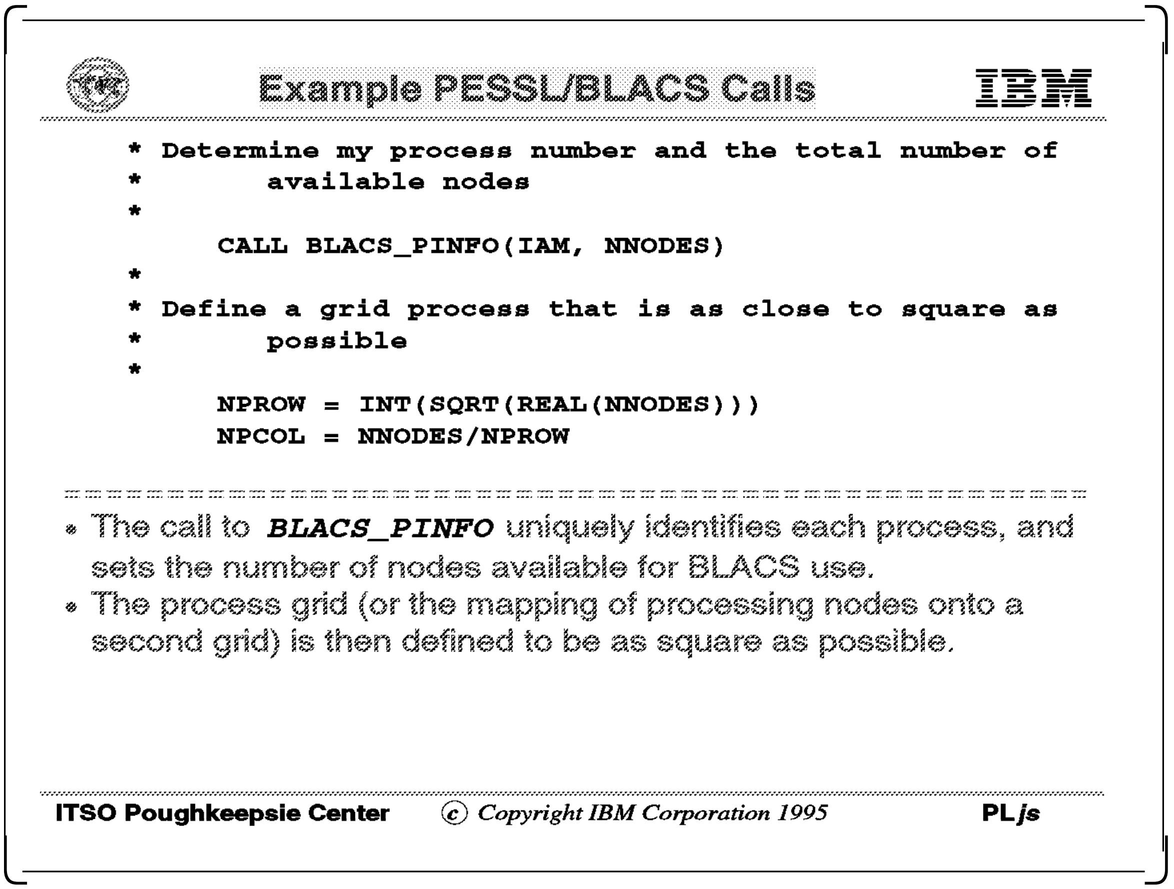
- Distribute the data across the process grid. Since the Parallel ESSL subroutines support the SPMD (single-program, multiple-data) programming model, an application′s global data structures (for example, vectors, matrices, or sequences) must be distributed across the processes that are members of the process grid, before calling the Parallel ESSL subroutine.

  Each Parallel ESSL subroutine specifies how the data is to be distributed to the nodes that are part of a particular process grid. The size and shape of the process grid and the way the global data structures are distributed over the processes are important factors to be considered.

- Call the Parallel ESSL subroutine on each process defined on the process grid.

- The solution data should then be processed according to the output distribution specified by the Parallel ESSL subroutine.

4.1.9 PESSL/BLACS Call Examples



Example PESSL/BLACS Calls

```
* Determine my process number and the total number of
*       available nodes
*
      CALL BLACS_PINFO(IAM, NNODES)
*
* Define a grid process that is as close to square as
*       possible
*
      NPROW = INT(SQRT(REAL(NNODES)))
      NPCOL = NNODES/NPROW
```

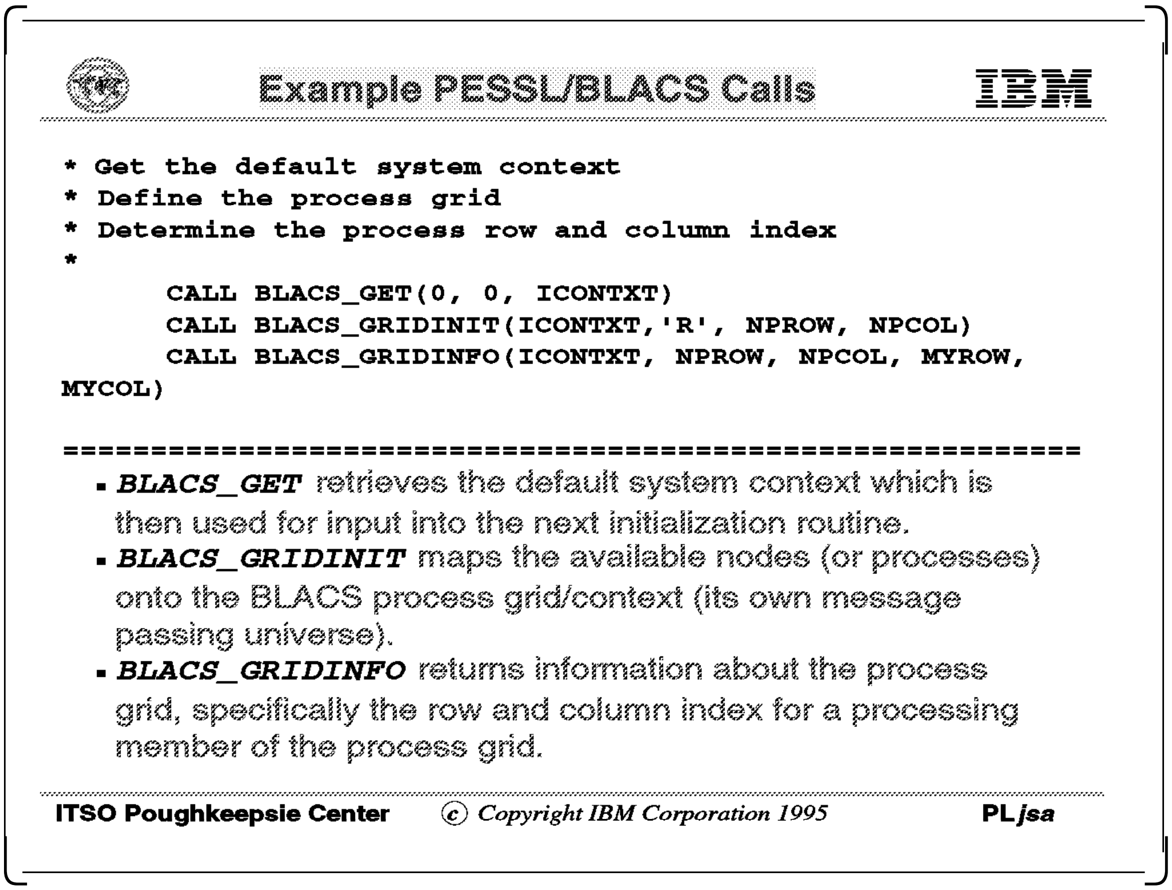
- The call to ***BLACS_PINFO*** uniquely identifies each process, and sets the number of nodes available for BLACS use.
- The process grid (or the mapping of processing nodes onto a second grid) is then defined to be as square as possible.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL*js*

The call to BLACS_PINFO uniquely identifies each process, and sets the number of nodes available for BLACS use.

The process grid (or the mapping of processing nodes onto a 2-dimensional grid) is defined to be as square as possible. For example, if six processors are available, then the process grid would be defined to have 2 rows and 3 columns, and a processor would be referenced by its coordinates in the grid, as shown in the diagram below, rather than as a single number.

(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)



Example PESSL/BLACS Calls

```
* Get the default system context
* Define the process grid
* Determine the process row and column index
*
      CALL BLACS_GET(0, 0, ICONTXT)
      CALL BLACS_GRIDINIT(ICONTXT,'R', NPROW, NPCOL)
      CALL BLACS_GRIDINFO(ICONTXT, NPROW, NPCOL, MYROW,
MYCOL)

==========================================================
```

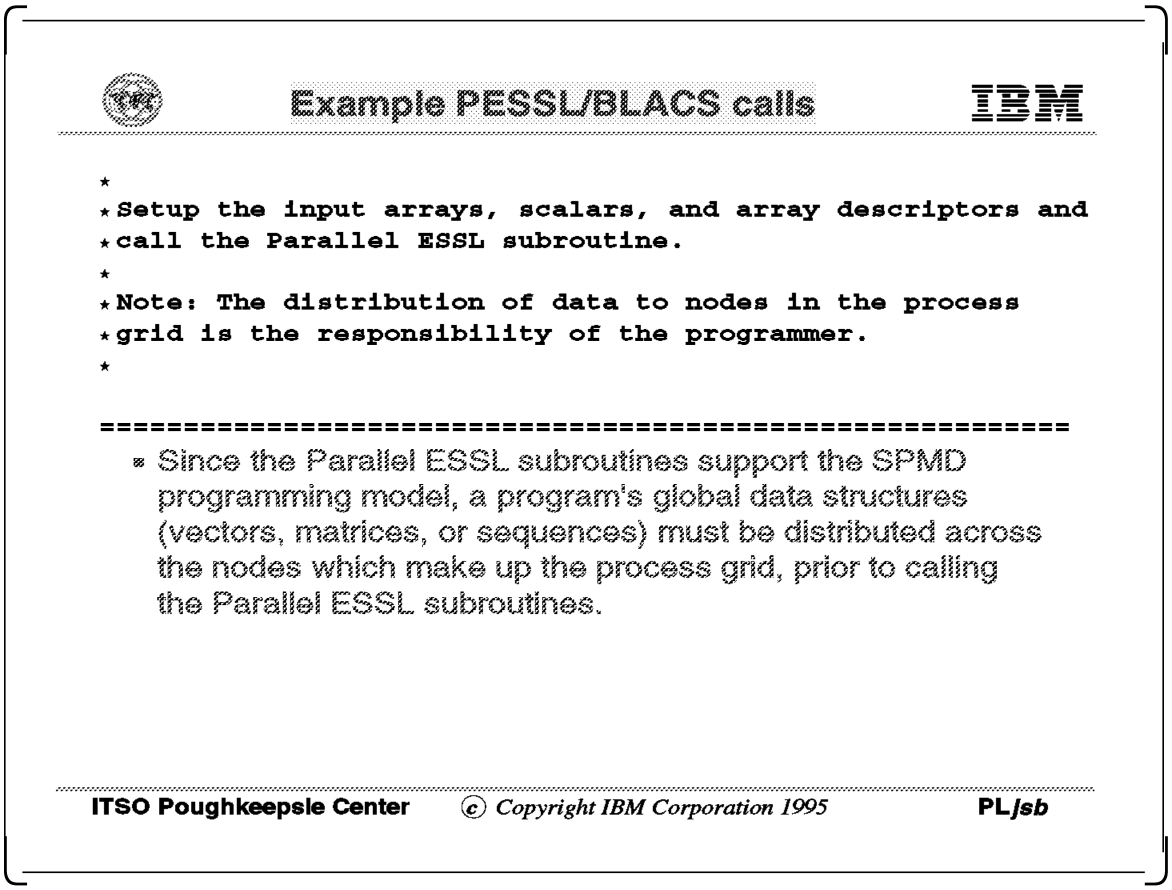
- ***BLACS_GET*** retrieves the default system context which is then used for input into the next initialization routine.
- ***BLACS_GRIDINIT*** maps the available nodes (or processes) onto the BLACS process grid/context (its own message passing universe).
- ***BLACS_GRIDINFO*** returns information about the process grid, specifically the row and column index for a processing member of the process grid.

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1995* **PL*jsa***

BLACS_GET retrieves the default system context to be input into the following initialization routines.

BLACS_GRIDINIT maps the available nodes (or processes) onto the BLACS process grid, thus establishing how the BLACS coordinate system will map into the native machine's process numbering system. Each BLACS grid is contained in a context (its own message passing universe) within which most computation takes place). This means that a grid does not interfere with distributed operations that occur within other (possibly overlapping) grids/contexts.

BLACS_GRIDINFO returns information about the process grid, specifically the row and column index for a processing member of the process grid.



Example PESSL/BLACS calls

```
*
*Setup the input arrays, scalars, and array descriptors and
*call the Parallel ESSL subroutine.
*
*Note: The distribution of data to nodes in the process
*grid is the responsibility of the programmer.
*

==========================================================
```

- Since the Parallel ESSL subroutines support the SPMD programming model, a program's global data structures (vectors, matrices, or sequences) must be distributed across the nodes which make up the process grid, prior to calling the Parallel ESSL subroutines.

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 PL/sb

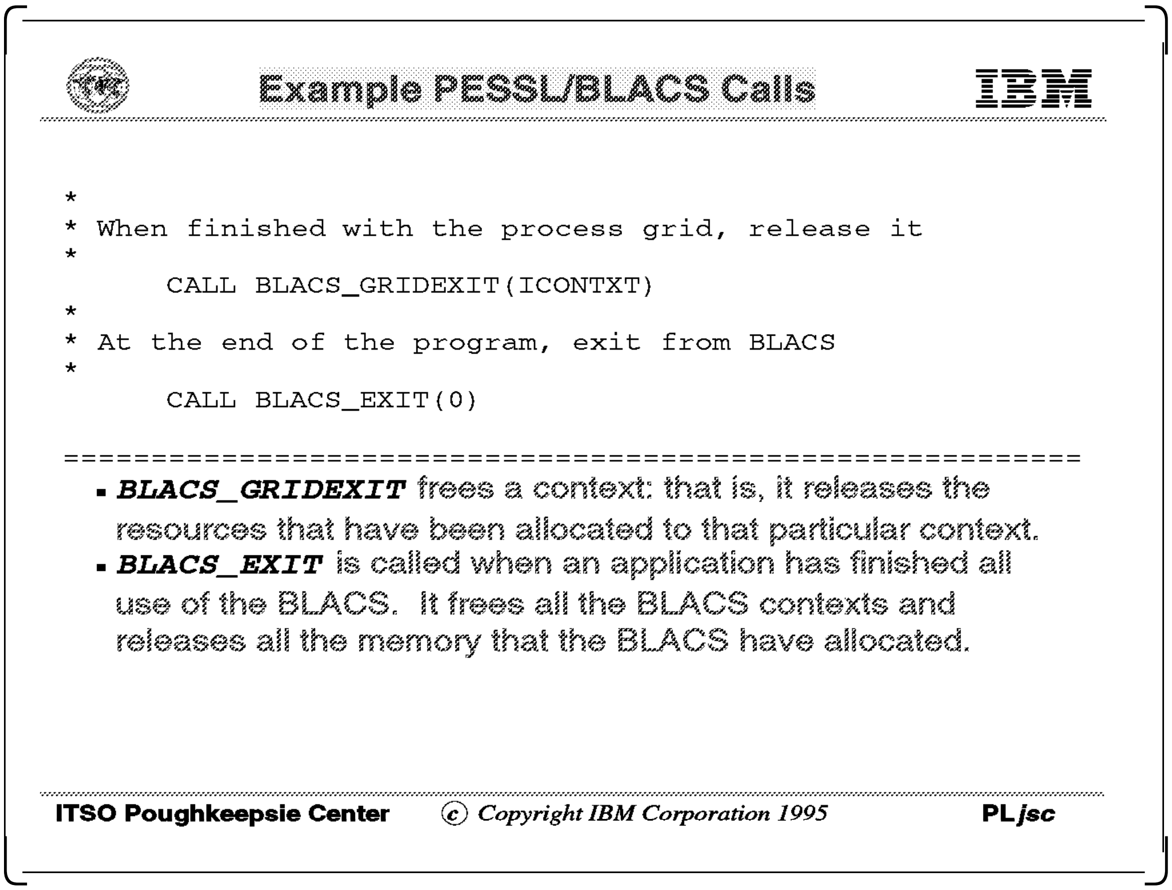
Since the Parallel ESSL subroutines support the SPMD programming model, a program′s global data structures (vectors, matrices, or sequences) must be distributed across the nodes that make up the process grid, before calling the Parallel ESSL subroutines.

Block distribution breaks up a matrix into blocks of data, whereby each processor participating in a parallel computation handles only one block of matrix elements. This is appropriate if matrix element computations involve neighboring elements.

Cyclic distribution breaks up the matrix into strips, and each processor participating in a parallel computation handles several strips of data. This is commonly used to provide better load balancing.

Block Cyclic distribution is a generalization of the block and cyclic distributions, in which blocks of r consecutive data objects are distributed cyclically over p nodes.

Methods for distributing data over one or two-dimensional process grids are described in *Parallel ESSL Guide and Reference*.



Example PESSL/BLACS Calls

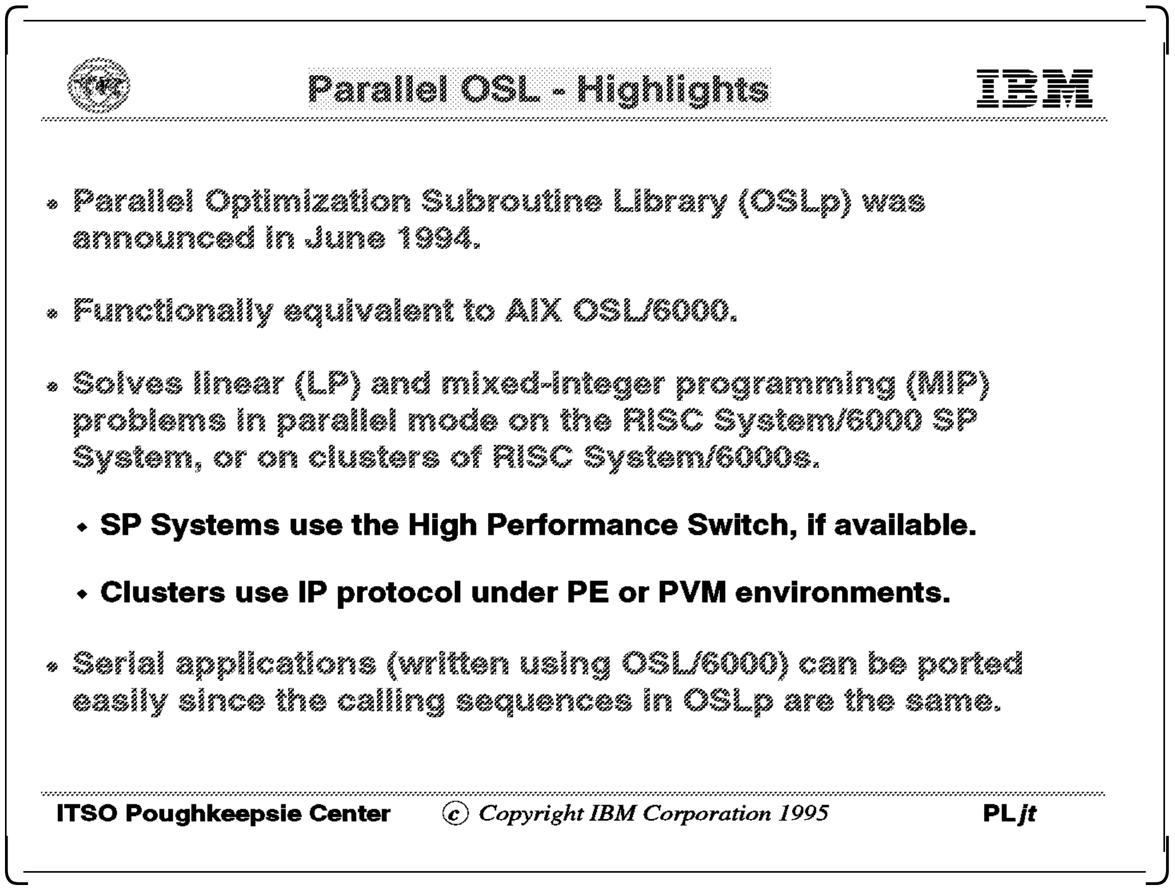
```
*
* When finished with the process grid, release it
*
      CALL BLACS_GRIDEXIT(ICONTXT)
*
* At the end of the program, exit from BLACS
*
      CALL BLACS_EXIT(0)

==========================================================
```

- ***BLACS_GRIDEXIT*** frees a context: that is, it releases the resources that have been allocated to that particular context.
- ***BLACS_EXIT*** is called when an application has finished all use of the BLACS. It frees all the BLACS contexts and releases all the memory that the BLACS have allocated.

BLACS_GRIDEXIT frees a context; that is, it releases the resources that have been allocated to that particular context.

BLACS_EXIT is called when an application has finished all use of the BLACS. It frees all the BLACS contexts and releases all the memory that the BLACS has allocated.

4.2 Parallel OSL



Parallel OSL - Highlights

- Parallel Optimization Subroutine Library (OSLp) was announced in June 1994.
- Functionally equivalent to AIX OSL/6000.
- Solves linear (LP) and mixed-integer programming (MIP) problems in parallel mode on the RISC System/6000 SP System, or on clusters of RISC System/6000s.
  - **SP Systems use the High Performance Switch, if available.**
  - **Clusters use IP protocol under PE or PVM environments.**
- Serial applications (written using OSL/6000) can be ported easily since the calling sequences in OSLp are the same.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PL*jt***

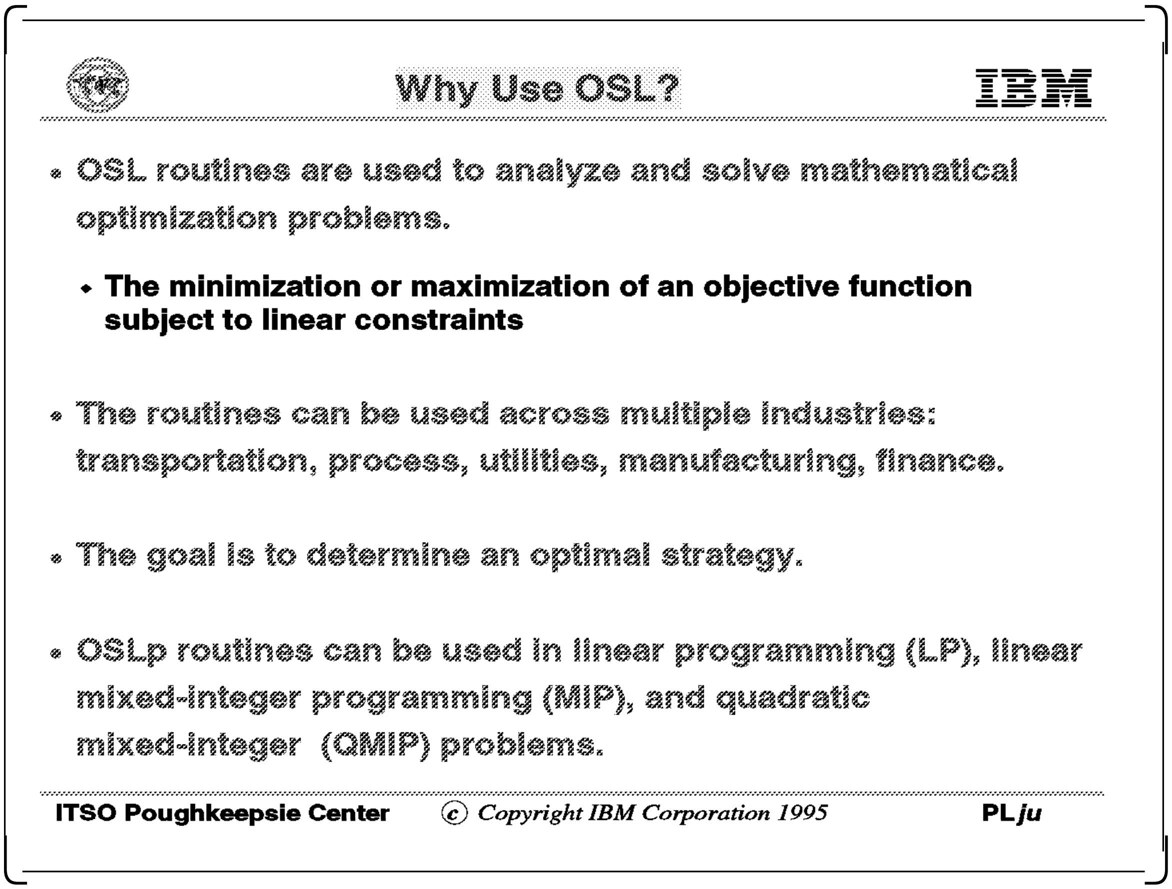
The Parallel Optimization subroutine Library (OSLp) can solve linear and mixed-integer programming problems in a parallel mode on the IBM RISC System/6000 Scalable POWERparallel SP System. This set of subroutines, which permit the transformation of serial applications to parallel applications, was originally announced in June 1994. Recent announcements have updated this program product to support the AIX Version 4.1.3 operating system on both RISC System/6000 SP systems and clusters.

Problems using AIX OSL/6000 are portable to OSLp because calling sequences are identical. Making minor changes to existing serial OSL application programs will enable them to run in a parallel environment.

Parallel OSL includes all of the functions of AIX OSL/6000, with the replacement of two major subroutines, EKKMSLV and EKKBSLV. These subroutines solve mixed integer programming (MIP) and linear programming (LP) problems, respectively.

Parallel OSL runs in any of the three parallel environments: IBM AIX PE, PVM, and IBM AIX PVMe.

4.2.1 Why use OSLp?



Why Use OSL?

- OSL routines are used to analyze and solve mathematical optimization problems.
  - The minimization or maximization of an objective function subject to linear constraints
- The routines can be used across multiple industries: transportation, process, utilities, manufacturing, finance.
- The goal is to determine an optimal strategy.
- OSLp routines can be used in linear programming (LP), linear mixed-integer programming (MIP), and quadratic mixed-integer (QMIP) problems.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL*ju*

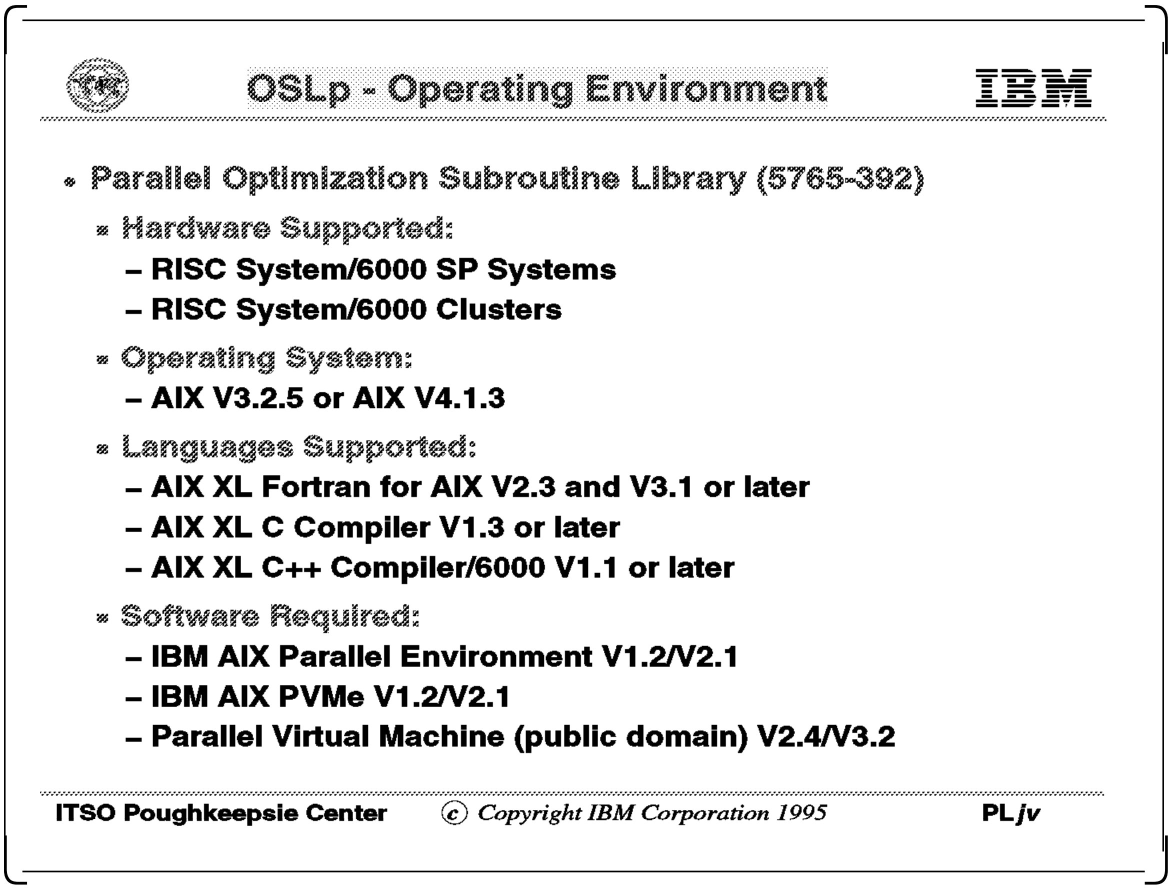
Mathematical optimization is concerned with the problem of finding an optimal allocation of limited resources by choosing an alternative that maximizes payoff, or minimizes cost, from among those alternatives that satisfy the given constraints.

The range of problems that can be addressed using OSL include the following:

1. Linear programming (LP) problems, where both the objective function and the constraints are linear. An example of this type of problem would be determining the optimum allocation of limited resources to meet objectives, while minimizing costs:. Here, both the allocation and cost equations would be described in linear equations.
2. Mixed-integer programming (MIP) problems, which are LPs where some of the variables are constrained to be integers. Some examples of this type of problem include the fare allocation of airline seats on a particular flight, or the number and type of aircraft on a particular route. Also decision variables, whose values determine which of two alternatives are to be implemented, are often modeled as [0,1] integer variables.
3. Quadratic MIP problems where the objective function to be minimized is quadratic, the constraints are linear, and some variables are constrained to be integer. This type of problem arises naturally in financial applications.

OSLp routines achieve *significant* speedups on LP and MIP problems.

4.2.2 OSLp Operating Environment



OSLp - Operating Environment

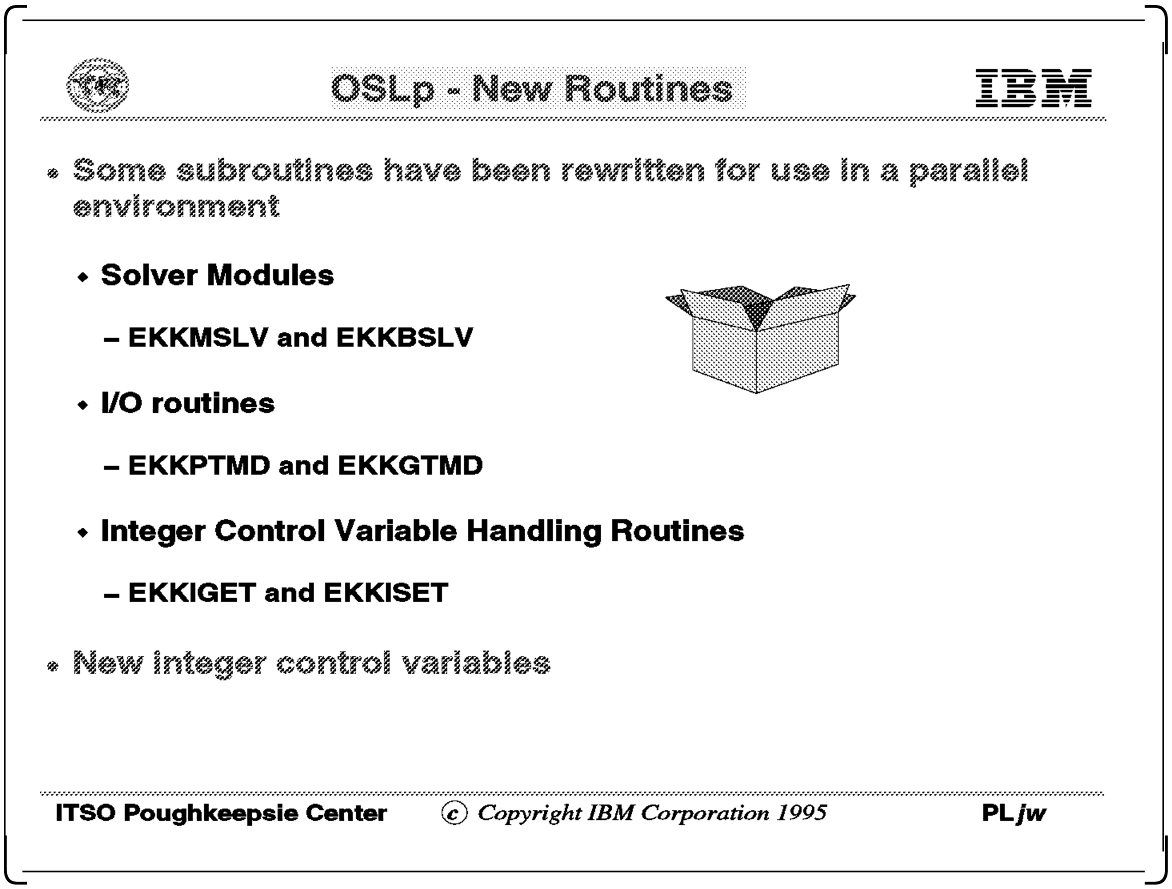
- Parallel Optimization Subroutine Library (5765-392)
  - Hardware Supported:
    - RISC System/6000 SP Systems
    - RISC System/6000 Clusters
  - Operating System:
    - AIX V3.2.5 or AIX V4.1.3
  - Languages Supported:
    - AIX XL Fortran for AIX V2.3 and V3.1 or later
    - AIX XL C Compiler V1.3 or later
    - AIX XL C++ Compiler/6000 V1.1 or later
  - Software Required:
    - IBM AIX Parallel Environment V1.2/V2.1
    - IBM AIX PVMe V1.2/V2.1
    - Parallel Virtual Machine (public domain) V2.4/V3.2

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* PL*jv*

This is a summary of the software required to install and use Parallel OSL on a RISC System/6000 Scalable POWERparallel System in an AIX V4.1.3 environment. These products must be installed on every node where it is intended to use Parallel OSL.

Note: OSLp must be installed on each node on which it is required to run OSLp code. The minimum number of nodes that can be licensed is four.

4.2.3 OSLp - New Routines


OSLp - New Routines

- Some subroutines have been rewritten for use in a parallel environment
  - Solver Modules
    - EKKMSLV and EKKBSLV
  - I/O routines
    - EKKPTMD and EKKGTMD
  - Integer Control Variable Handling Routines
    - EKKIGET and EKKISET
- New integer control variables

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL*jw*

All of the features and capabilities of the serial OSL product are available in Parallel OSL. The OSLp calls are compatible with the serial OSL, making it simple to adapt existing programs. Further, users do not need to know how to write parallel code to use Parallel OSLp: adding minor changes to existing serial programs allows these programs to be run in a parallel environment.

The new solver routines, EKKMSLV (for MIP problems) and EKKBSLV (for LP problems), have been modified to share the work of solving their problems among multiple processors.

The I/O routines EKKPTMD and EKKGTMD have been modified so that if the FORTRAN unit number supplied to these routines is greater than 100, they generate and receive messages passed between OSLp application processes. If these unit numbers are 99 or less, the routines function exactly as the serial versions do.

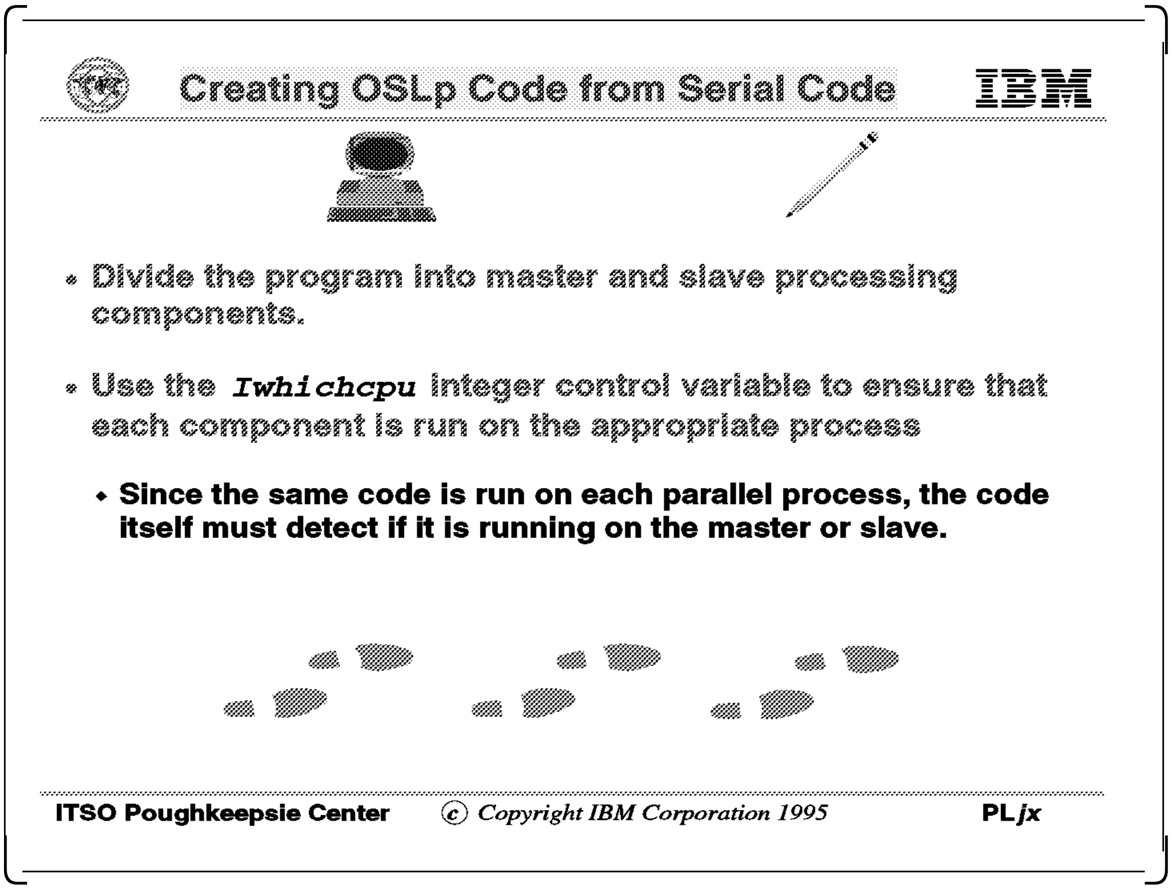
The integer control variable handling routines EKKIGET and EKKISET access and modify, respectively, all OSLp integer control variables (which are a superset of the OSL variables). The new OSLp control variables are:

- `Inumcpu` is used to initiate and terminate parallel processing. This control variable can have different values in different OSLp processes. When the master process resets `Inumcpu` from zero to some value other than minus one (-1), EKKISET initiates parallel processing. EKKISET terminates parallel processing when the master and slave processes reset Inumcpu to zero.

- Iwhichcpu provides the mechanism for identifying the process number associated with a particular instance of parallel code:

  Iwhichcpu = 0 identifies the master process.
  Iwhichcpu = [1,...,n] identifies the slave processes.

4.2.4 Creating OSLp Code from Serial OSL Code

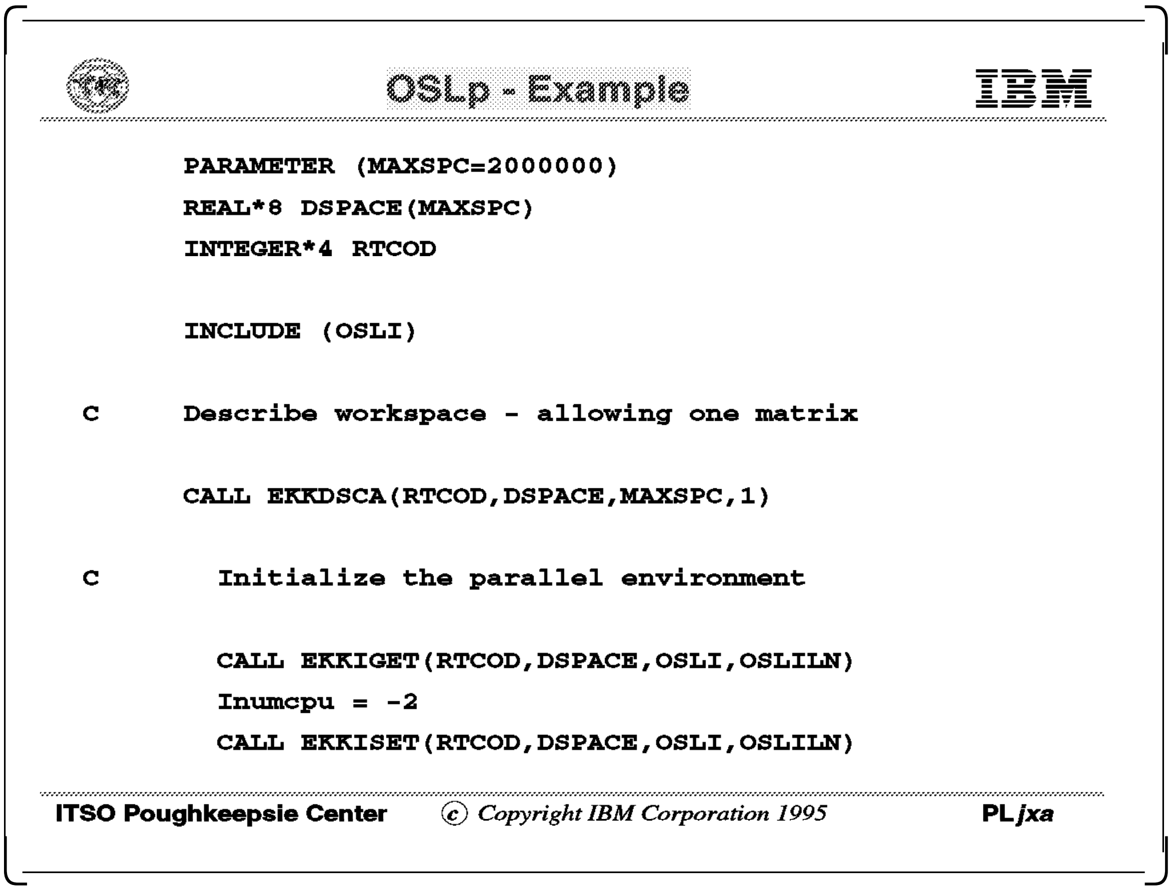


Creating OSLp Code from Serial Code

- Divide the program into master and slave processing components.
- Use the *Iwhichcpu* integer control variable to ensure that each component is run on the appropriate process
  - **Since the same code is run on each parallel process, the code itself must detect if it is running on the master or slave.**

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL*jx*

To create an effective Parallel OSL application program, the components of the program that will be called by the master process need to be separated from those that will be called by the slave processes. This could be done by coding distinct MASTER and SLAVE subroutines, or by separating the master and slave components within the same routine. In either case, since all parallel OSL processes run the same code, the code itself must detect which process is running. This is done by checking the value of Iwhichcpu.

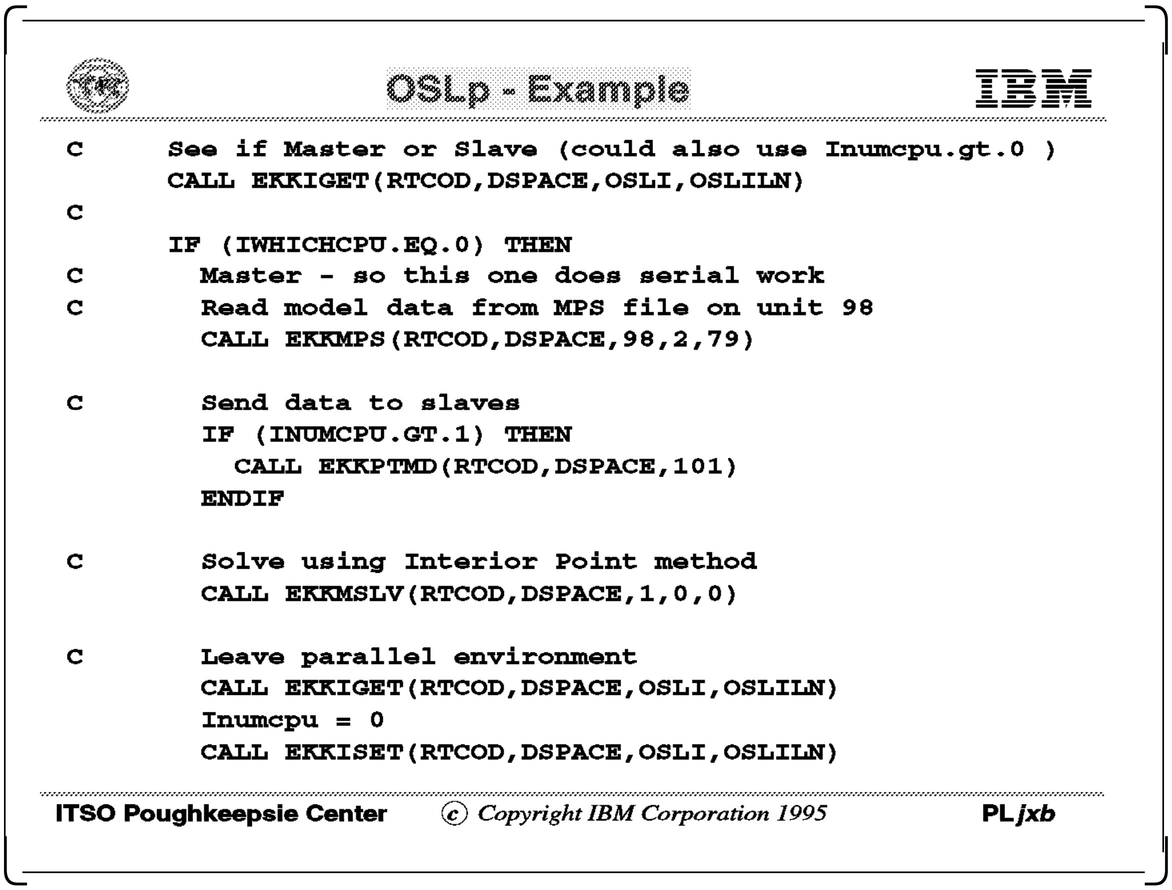


OSLp - Example

```
      PARAMETER (MAXSPC=2000000)
      REAL*8 DSPACE(MAXSPC)
      INTEGER*4 RTCOD

      INCLUDE (OSLI)

C     Describe workspace - allowing one matrix

      CALL EKKDSCA(RTCOD,DSPACE,MAXSPC,1)

C       Initialize the parallel environment

        CALL EKKIGET(RTCOD,DSPACE,OSLI,OSLILN)
        Inumcpu = -2
        CALL EKKISET(RTCOD,DSPACE,OSLI,OSLILN)
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1995   PL*jxa*

In the initial part of the program, the size of the OSL work space is defined, and the application is described to OSL.



OSLp - Example

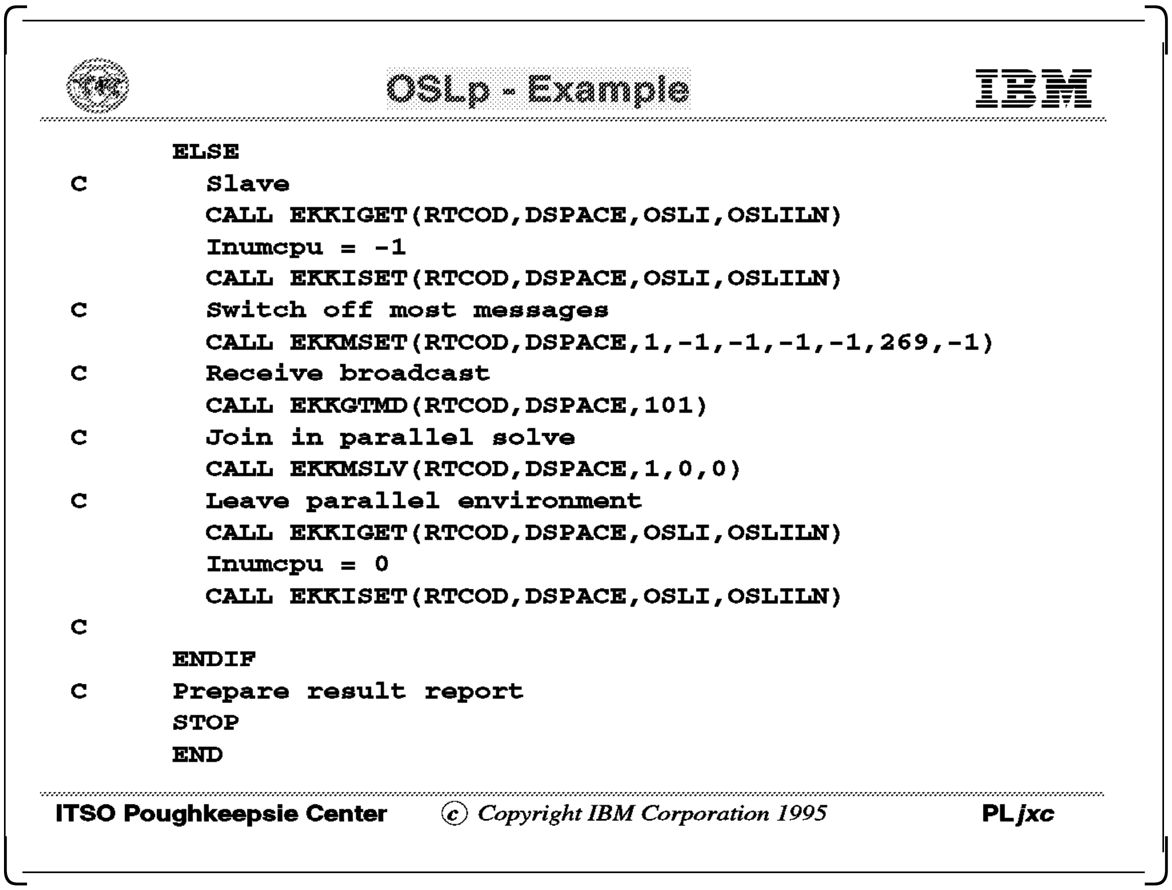
```
C      See if Master or Slave (could also use Inumcpu.gt.0 )
       CALL EKKIGET(RTCOD,DSPACE,OSLI,OSLILN)
C
       IF (IWHICHCPU.EQ.0) THEN
C        Master - so this one does serial work
C        Read model data from MPS file on unit 98
         CALL EKKMPS(RTCOD,DSPACE,98,2,79)

C        Send data to slaves
         IF (INUMCPU.GT.1) THEN
           CALL EKKPTMD(RTCOD,DSPACE,101)
         ENDIF

C        Solve using Interior Point method
         CALL EKKMSLV(RTCOD,DSPACE,1,0,0)

C        Leave parallel environment
         CALL EKKIGET(RTCOD,DSPACE,OSLI,OSLILN)
         Inumcpu = 0
         CALL EKKISET(RTCOD,DSPACE,OSLI,OSLILN)
```

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* PL*jxb*

Each process then determines if it is the master or a slave by retrieving the Iwhichcpu control variable through a call to the EKKIGET routine. In the master process, Iwhichcpu will be zero, and Inumcpu will be the number of slave processes.

The master process reads the input data (call to EKKMPS) and sends the data out to the slave processes (call to EKKPTMD; recall that unit numbers greater than 100 are used to communicate between OSLp processes running on different nodes). The master process then calls EKKMSLV to solve its portion of this MIP problem.

When this is completed, the master process terminates parallel processing by resetting Inumcpu to zero.



OSLp - Example

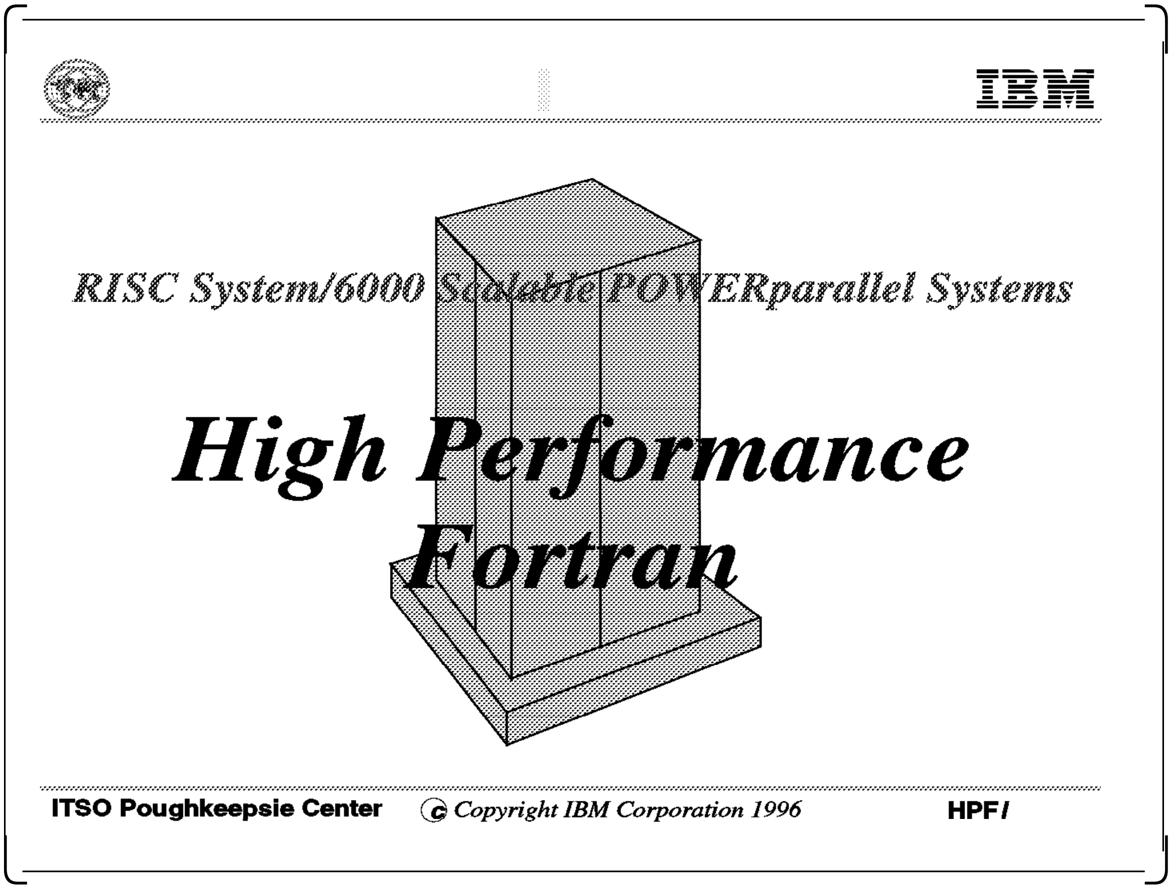
```
      ELSE
C       Slave
        CALL EKKIGET(RTCOD,DSPACE,OSLI,OSLILN)
        Inumcpu = -1
        CALL EKKISET(RTCOD,DSPACE,OSLI,OSLILN)
C       Switch off most messages
        CALL EKKMSET(RTCOD,DSPACE,1,-1,-1,-1,-1,269,-1)
C       Receive broadcast
        CALL EKKGTMD(RTCOD,DSPACE,101)
C       Join in parallel solve
        CALL EKKMSLV(RTCOD,DSPACE,1,0,0)
C       Leave parallel environment
        CALL EKKIGET(RTCOD,DSPACE,OSLI,OSLILN)
        Inumcpu = 0
        CALL EKKISET(RTCOD,DSPACE,OSLI,OSLILN)
C
      ENDIF
C     Prepare result report
      STOP
      END
```

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* **PL*jxc***

In the slave processes, `Iwhichcpu` will be set to [1, ... ,n-1], and `Inumcpu` will be set to -1 to indicate that this process is a slave processes. The slave processes then read the input data sent to them by the master process (call to `EKKGTMD`; recall that unit numbers greater than 100 are used to communicate between OSLp processes running on different nodes). The slave processes then call `EKKMSLV` to solve their portions of the MIP problem.

When these are completed, the slaves also close the parallel processing by resetting `Inumcpu` to zero.

Chapter 5. High Performance Fortran



RISC System/6000 Scalable POWERparallel Systems

High Performance Fortran

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF/

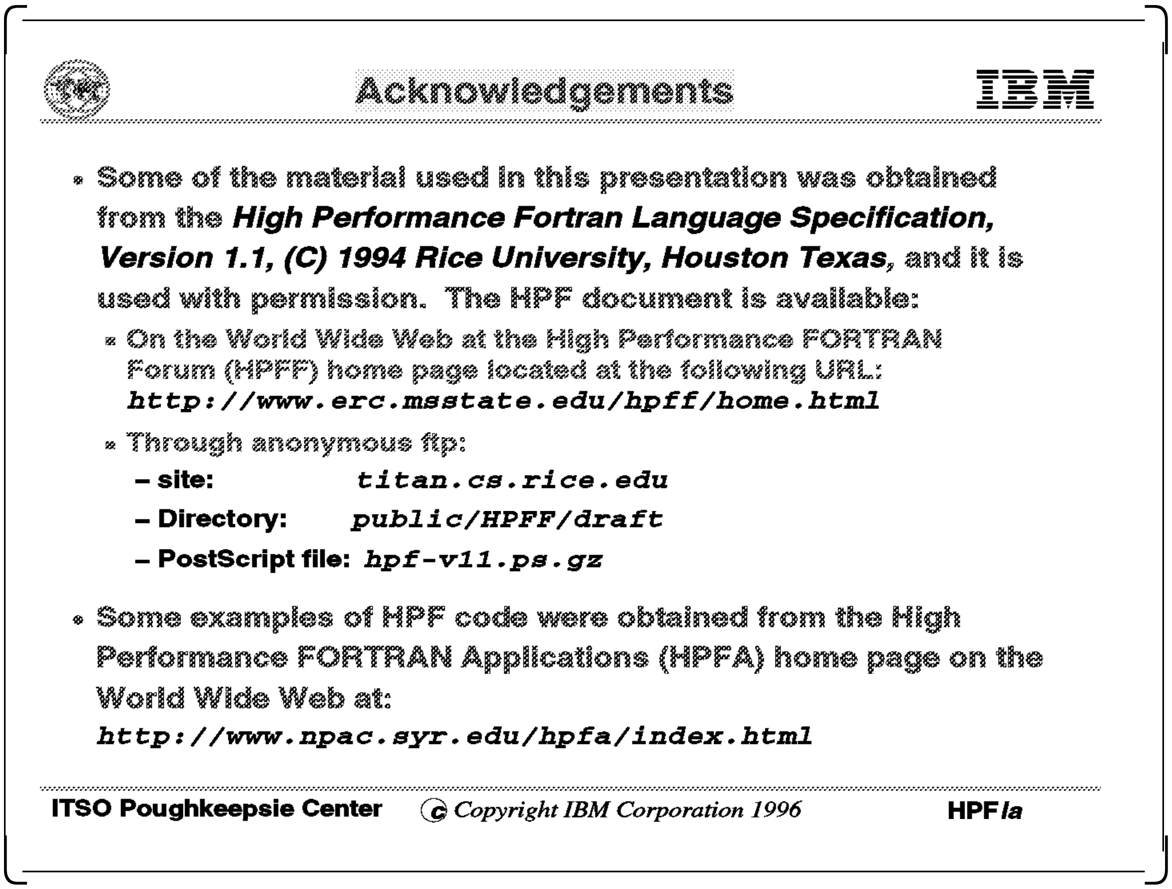
On December 5, 1995, IBM announced its High Performance Fortran (HPF) compiler to provide scientific customers with a HPF compiler supporting the HPF standard defined by Fortran committees. The consortium that defined the HPF standard also defined a subset of HPF features that must be supported in order to be HPF branded. In fact, the IBM HPF includes the HPF subset and adds several HPF features not included in the subset yet.

HPF was designed to help researchers and NIC application programmers develop performing parallel applications without any message passing programming. So, the message passing functions are implicitly generated by the HPF compiler with respect to specific information given by the developer about the data mapping, that is, the way data is logically stored in memory, and the way it is distributed between the processors.

This chapter provides the following information:

- New HPF statements and constructs
- HPF directives
- Compile-time flags

5.1.1 Acknowledgements


Acknowledgements

- Some of the material used in this presentation was obtained from the ***High Performance Fortran Language Specification, Version 1.1, (C) 1994 Rice University, Houston Texas,*** and it is used with permission. The HPF document is available:
  - On the World Wide Web at the High Performance FORTRAN Forum (HPFF) home page located at the following URL: ***http://www.erc.msstate.edu/hpff/home.html***
  - Through anonymous ftp:
    - **site:** ***titan.cs.rice.edu***
    - **Directory:** ***public/HPFF/draft***
    - **PostScript file:** ***hpf-v11.ps.gz***
- Some examples of HPF code were obtained from the High Performance FORTRAN Applications (HPFA) home page on the World Wide Web at: ***http://www.npac.syr.edu/hpfa/index.html***

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1996* **HPF*la***

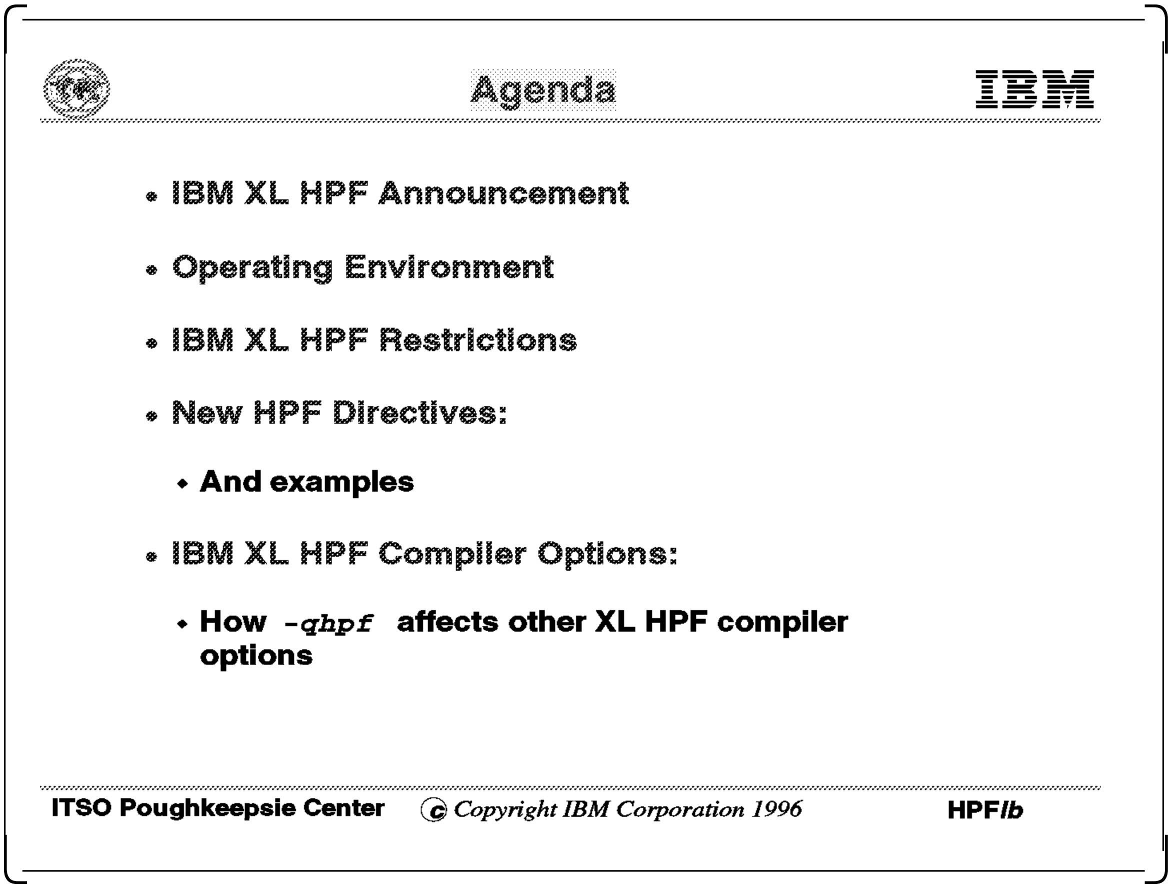
Some of the material used in this presentation was obtained from the *High Performance Fortran Language Specification*, Version 1.1, (C) 1994 Rice University, Houston Texas, and it is used with permission. The HPF language specification document is available on the World Wide Web at the High Performance Fortran Forum (HPFF) home page located at the following URL: `http://www.erc.msstate.edu/hpff/home.html`

Additional information could be obtain through the anonymous ftp site *titan.cs.rice.edu* The directory is *public/HPFF/draft* and the PostScript file is *hpf-v11.ps.gz*.

Some examples of HPF code were obtained from the High Performance Fortran Applications (HPFA) home page on the World Wide Web at `http://www.npac.syr.edu/hpfa/index.html`.

The IBM documentation was also used to provide input to this presentation and its examples. See *IBM XL High Performance Fortran User's Guide* and *IBM XL High Performance Language Reference* for more information.

5.2 Topics Covered


Agenda

IBM

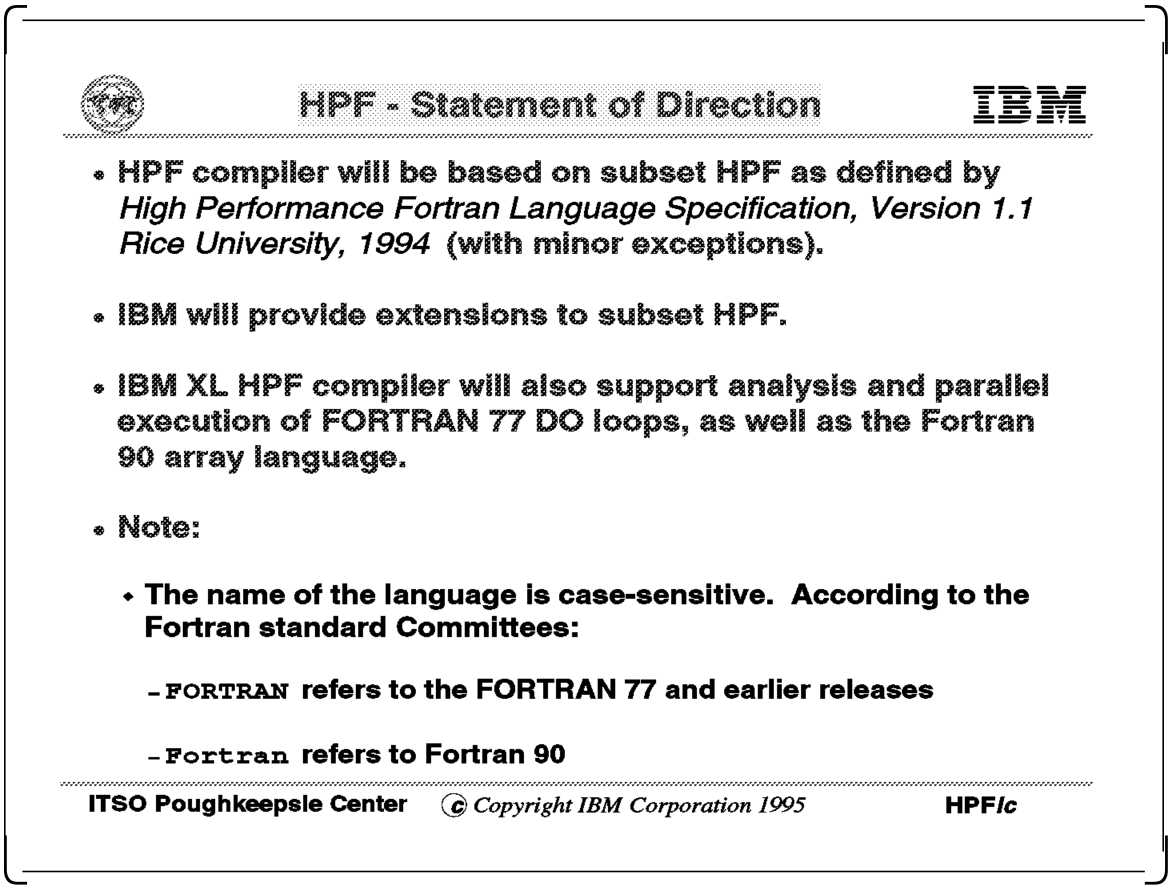
- IBM XL HPF Announcement
- Operating Environment
- IBM XL HPF Restrictions
- New HPF Directives:
  - And examples
- IBM XL HPF Compiler Options:
  - How *-qhpf* affects other XL HPF compiler options

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1996* HPF*lb*

The following topics will be covered:

- IBM′s Announcement for High Performance Fortran (December 5, 1995).
- Summary of the operating environment, and the description of some restrictions in the HPF compiler.
- Summary of the new HPF directives with examples illustrating their use.
- Short Guide to new and/or changed compiler options.

5.2.1 HPF Announcement (December 5, 1995)



HPF - Statement of Direction

- HPF compiler will be based on subset HPF as defined by *High Performance Fortran Language Specification, Version 1.1 Rice University, 1994* (with minor exceptions).
- IBM will provide extensions to subset HPF.
- IBM XL HPF compiler will also support analysis and parallel execution of FORTRAN 77 DO loops, as well as the Fortran 90 array language.
- Note:
  - The name of the language is case-sensitive. According to the Fortran standard Committees:
    - FORTRAN refers to the FORTRAN 77 and earlier releases
    - Fortran refers to Fortran 90

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1995* HPF*lc*

IBM announced in June 1995 that it intended to develop and deliver to the marketplace, a High Performance Fortran (HPF) compiler based on the Subset HPF, as defined by the *High Performance Fortran Language Specification, Version 1.1, Rice University, 1994.* The IBM HPF compiler was announced in December 1995 with these specifications, and the following extensions:

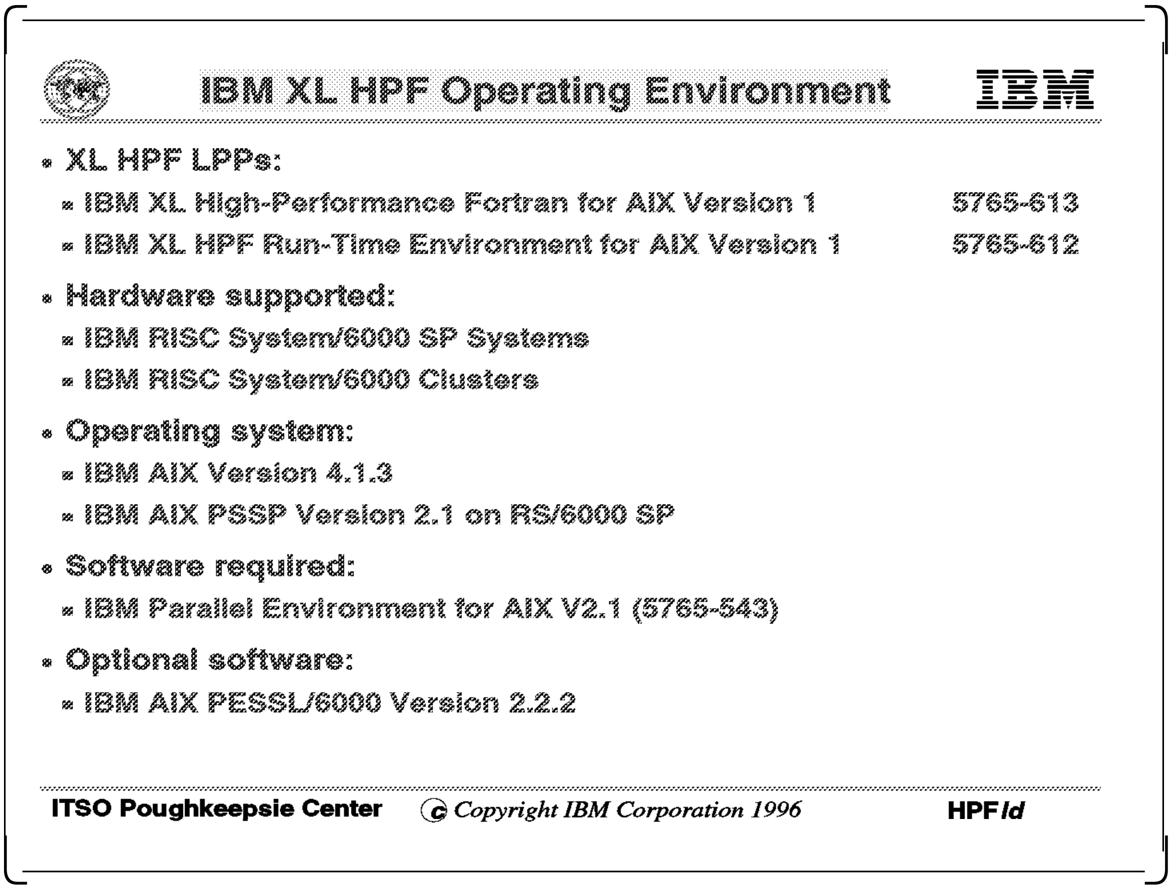
- PURE procedures
- FORALL construct
- Storage and sequence association, including the SEQUENCE directive (but not supporting mapping of sequenced variables)
- HPF_LOCAL and HPF_SERIAL extrinsic kinds (on subroutines and functions only)
- The HPF_LOCAL_LIBRARY module

  The IBM HPF compiler supports the following HPF_LOCAL_LIBRARY subroutines:

  - ABSTRACT_TO_PHYSICAL
  - GLOBAL_ALIGNMENT
  - GLOBAL_DISTRIBUTION
  - GLOBAL_TEMPLATE
  - GLOBAL_TO_LACAL
  - LOCAL_TO_GLOBAL
  - MY_PROCESSOR

  - PHYSICAL_TO_ABSTRACT
- The HPF_DISTRIBUTION, HPF_TEMPLATE, and HPF_ALIGNMENT inquiry subroutines of the HPF_LIBRARY module
- The DIM argument of the MAXLOC and MINLOC intrinsic functions which is part of the subset.

5.2.2 HPF - Operating Environment



IBM XL HPF Operating Environment

- XL HPF LPPs:
  - IBM XL High-Performance Fortran for AIX Version 1 5765-613
  - IBM XL HPF Run-Time Environment for AIX Version 1 5765-612
- Hardware supported:
  - IBM RISC System/6000 SP Systems
  - IBM RISC System/6000 Clusters
- Operating system:
  - IBM AIX Version 4.1.3
  - IBM AIX PSSP Version 2.1 on RS/6000 SP
- Software required:
  - IBM Parallel Environment for AIX V2.1 (5765-543)
- Optional software:
  - IBM AIX PESSL/6000 Version 2.2.2

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPFld

The objective of the HPF standard is to help programmers decompose data parallel problems for all parallel machines. High Performance Fortran will allow programmers:

- To identify scalar and array variables that will be distributed across a parallel machine
- To specify how the scalar and array data will be distributed: in strips, blocks, or in another format
- To specify the alignment between these variables on each other

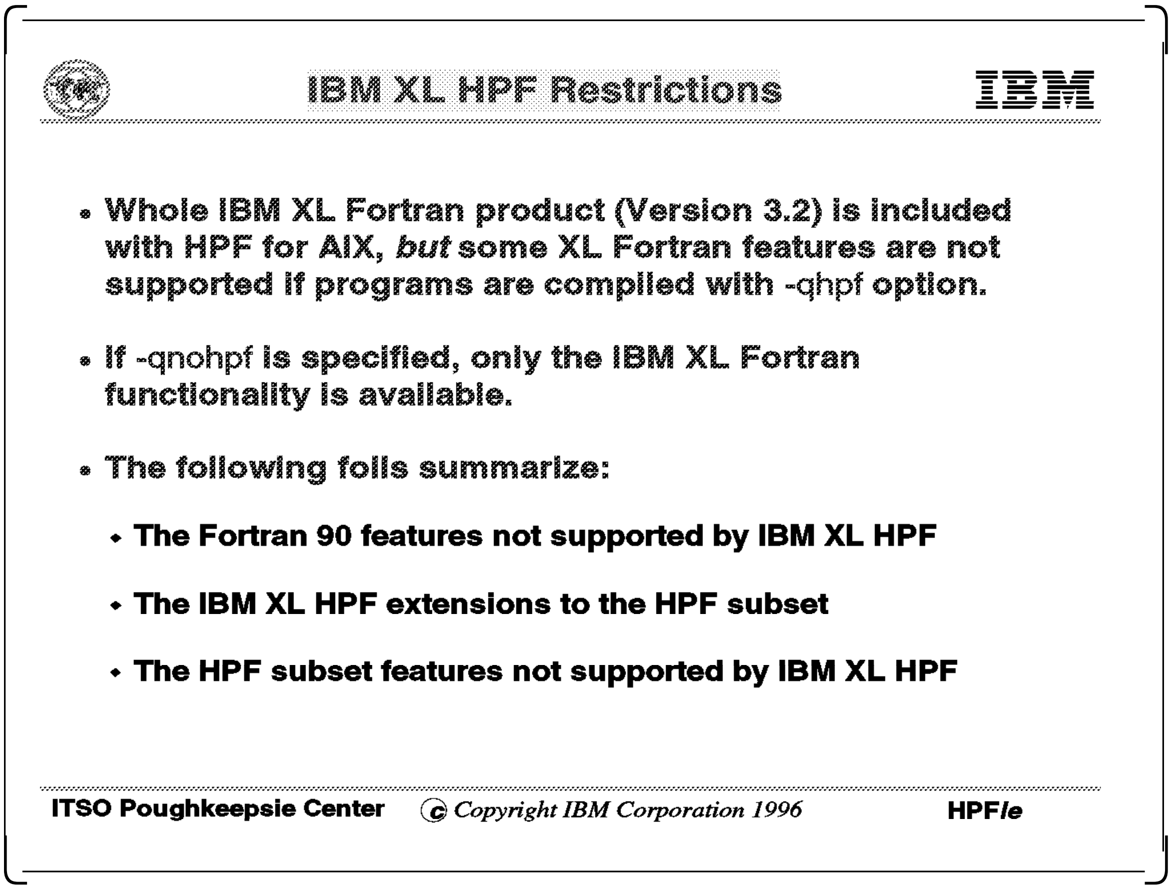
The required operating environment for the XL HPF product will be:

- RISC System/6000 Scalable POWERparallel (SP) Systems, or RISC System/6000 clusters
- AIX Version 4.1 and IBM Parallel Environment for AIX Version 2.1.

  IBM PE provides the MPI library that includes the MPI subroutines called by the executable generated by HPF.

In support of HPF for AIX, IBM intends to provide the Parallel Engineering and Scientific Subroutine Library for AIX (Parallel ESSL for AIX) in the second quarter of 1996. This product will offer mathematical subroutines that can be easily called by XL HPF for AIX programs.

5.2.3 HPF Restrictions



IBM XL HPF Restrictions

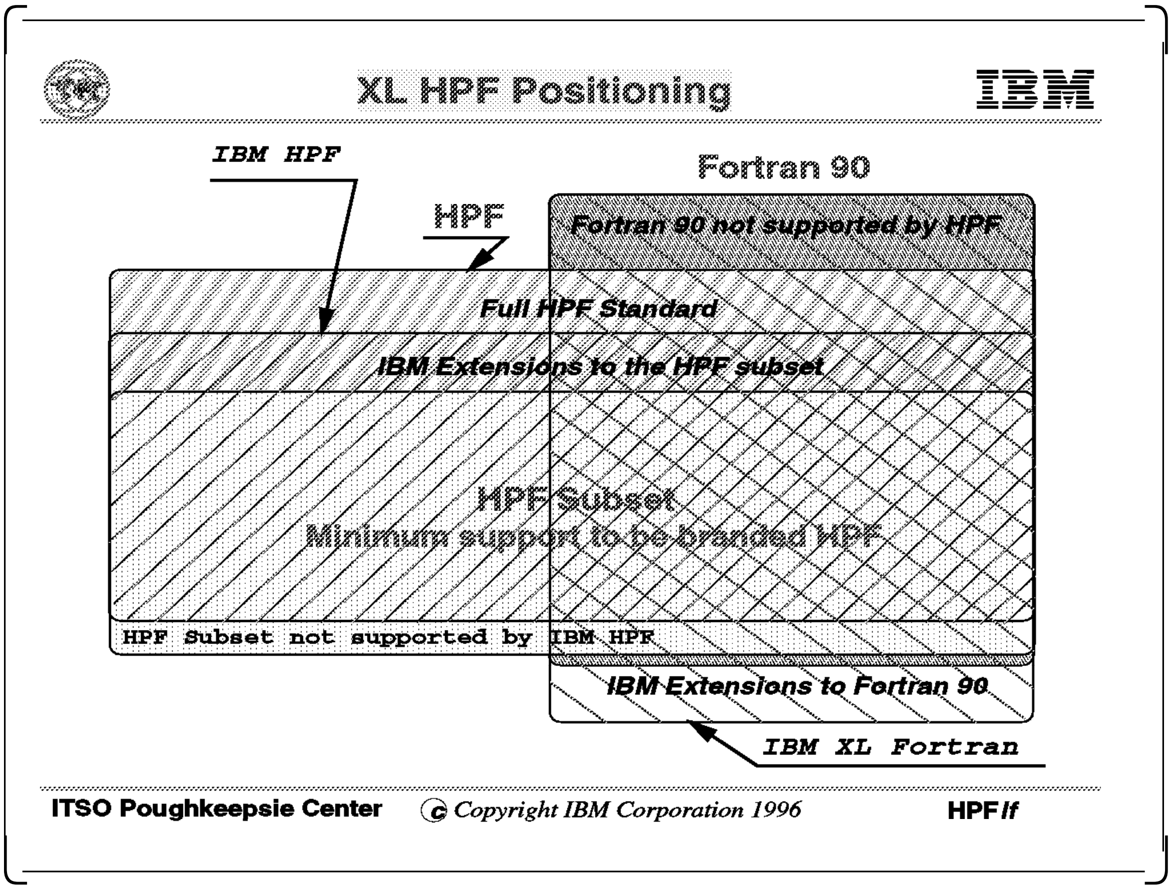
- Whole IBM XL Fortran product (Version 3.2) is included with HPF for AIX, *but* some XL Fortran features are not supported if programs are compiled with -qhpf option.
- If -qnohpf is specified, only the IBM XL Fortran functionality is available.
- The following foils summarize:
  - The Fortran 90 features not supported by IBM XL HPF
  - The IBM XL HPF extensions to the HPF subset
  - The HPF subset features not supported by IBM XL HPF

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*le*

- If the *-qhpf* option is used, then not all of the XL Fortran features are supported.
- If the *-qnohpf* option is used, then only the XL Fortran functionality is available.
- A number of features of Subset HPF as defined in the standard are *not* supported. These include:
  - CYCLIC(n) data distribution (CYCLIC(n) is treated as CYCLIC(1) unless it appears in an HPF_LOCAL interface.
  - The ENTRY statement

    In FORTRAN 77, the ENTRY statement is put into functions and subroutines when you want to access them different ways.
  - Multiple processor arrangements

5.2.4 HPF vs XLF


XL HPF Positioning
IBM
IBM HPF
Fortran 90
HPF
Fortran 90 not supported by HPF
Full HPF Standard
IBM Extensions to the HPF subset
HPF Subset
Minimum support to be branded HPF
HPF Subset not supported by IBM HPF
IBM Extensions to Fortran 90
IBM XL Fortran
ITSO Poughkeepsie Center
© Copyright IBM Corporation 1996
HPFlf

This diagram summarizes the functional relationship between HPF and XL Fortran.

IBM Extension to Fortran 90

IBM XL Fortran for AIX, Version 3 Release 2 conforms to the Fortran 90 standard. It includes several extensions, such as:

- Directives lines

  IBM XL Fortran allows users to specify compile-time options directly in the source file with the @PROCESS directive. SOURCEFORM is another directive used to indicate which form the source is written in.

- Typeless literal constants

  IBM XL Fortran supports several typeless constants, such as hexadecimal, octal, binary, and Hollerith constants.

- Storage classes for variables

  IBM XL Fortran assigns, implicitly or explicitly, a storage class to variables, such as, automatic, static, common. So, several statements have additional optional parameters to set up the storage class of variables.

- BYTE

This type declaration statement specifies the attributes of objects and functions of type byte with length of 1.

- DOUBLE COMPLEX

  Each component of this complex number is a REAL(16) number.

- Intrinsic Procedures

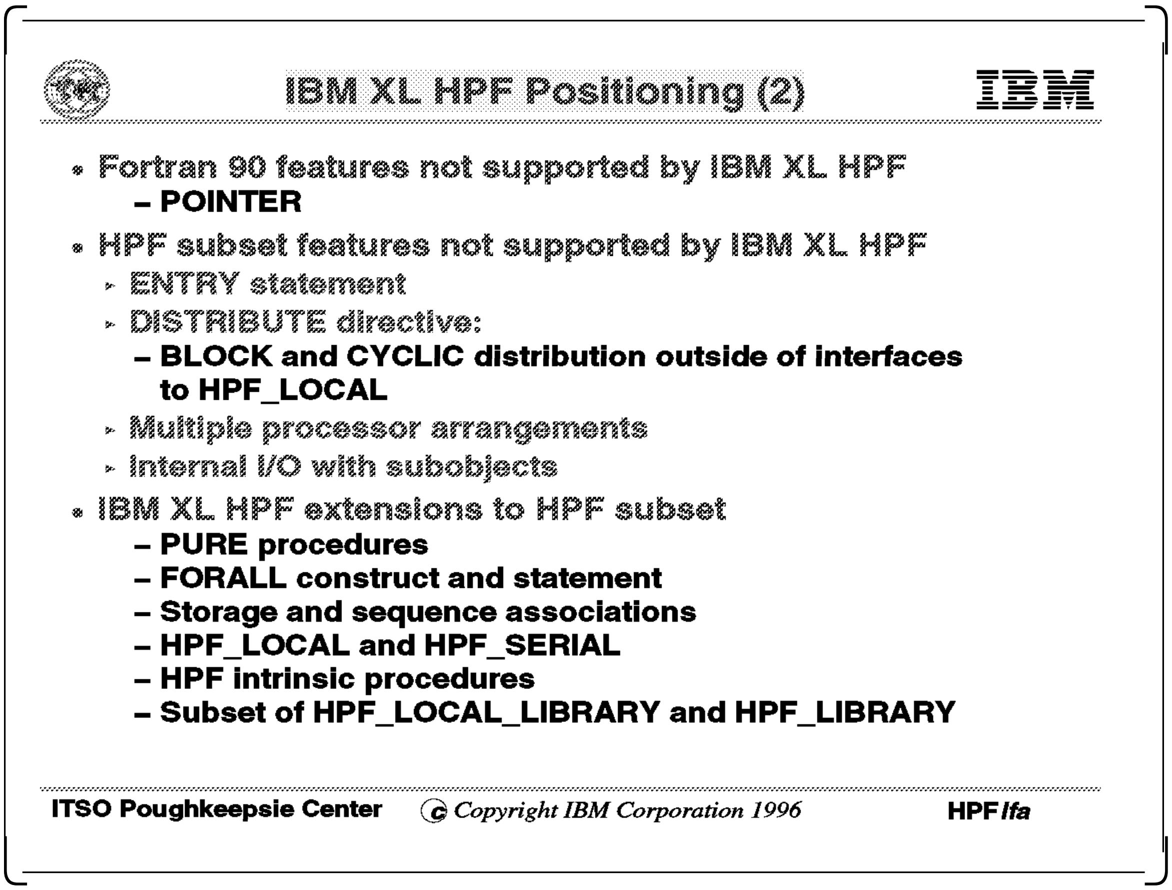
  IBM XL Fortran includes several intrinsic procedures to the standard Fortran 90 list, such as erf, erfc, gamma, lgamma, and so on.

- Service and Utility Procedures

  IBM XL Fortran provides utility services, such as alarm_, date_, exit_, flush_, getarg, getuid_, qsort_, and many others.

HPF Subset Not Supported by IBM XL HPF

IBM XL HPF supports the entire Subset HPF definition, with the exception of the features listed in Section 5.2.3, “HPF Restrictions” on page 127.



IBM XL HPF Positioning (2)

- Fortran 90 features not supported by IBM XL HPF
  - POINTER
- HPF subset features not supported by IBM XL HPF
  - ENTRY statement
  - DISTRIBUTE directive:
    - BLOCK and CYCLIC distribution outside of interfaces to HPF_LOCAL
  - Multiple processor arrangements
  - Internal I/O with subobjects
- IBM XL HPF extensions to HPF subset
  - PURE procedures
  - FORALL construct and statement
  - Storage and sequence associations
  - HPF_LOCAL and HPF_SERIAL
  - HPF intrinsic procedures
  - Subset of HPF_LOCAL_LIBRARY and HPF_LIBRARY

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*lfa*

IBM Extensions to HPF Subset

IBM XL HPF includes several HPF features not part of the Subset HPF:

- PURE procedure
- FORALL construct

  Only the FORALL statement is part of the Subset HPF.
- SEQUENCE and NOSEQUENCE directives
- HPF_LOCAL and HPF_SERIAL extrinsic kinds
- Procedures from HPF_LOCAL_LIBRARY and HPF_LIBRARY

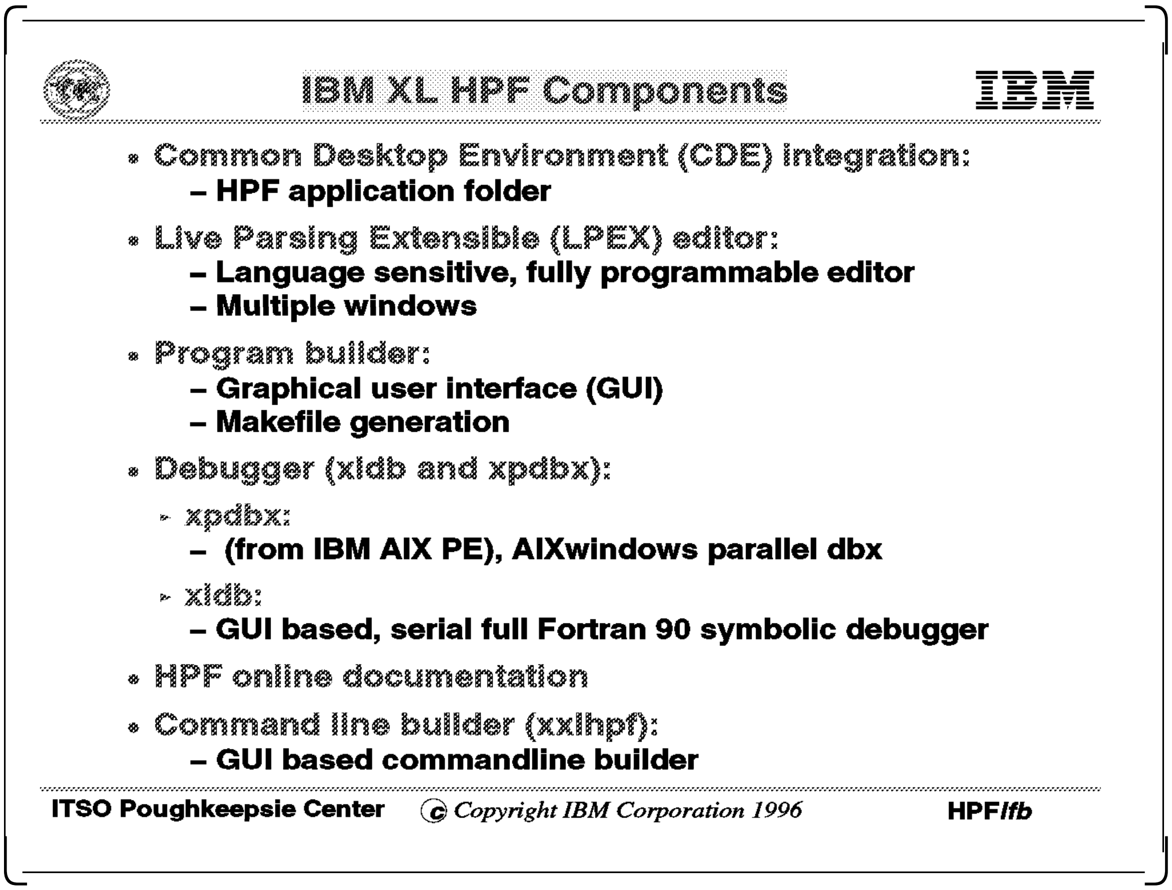
Full HPF Standard

The full HPF standard complements the Subset HPF with the following features that are not supported by IBM XL HPF yet:

- REALIGN directive
- REDISTRIBUTE directive
- DYNAMIC directive
- INHERIT directive

Fortran 90 Not Supported by IBM XL HPF

IBM XL HPF supports neither Fortran 90 pointers nor integer pointers.



IBM XL HPF Components

- Common Desktop Environment (CDE) integration:
  - HPF application folder
- Live Parsing Extensible (LPEX) editor:
  - Language sensitive, fully programmable editor
  - Multiple windows
- Program builder:
  - Graphical user interface (GUI)
  - Makefile generation
- Debugger (xldb and xpdbx):
  - xpdbx:
    - (from IBM AIX PE), AIXwindows parallel dbx
  - xldb:
    - GUI based, serial full Fortran 90 symbolic debugger
- HPF online documentation
- Command line builder (xxlhpf):
  - GUI based commandline builder

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1996* HPF*lfb*

IBM XL HPF provides programmers a set of AIX-based tools integrated with the AIX common desktop environment (CDE), for the development of parallel applications running on AIX under RS/6000 SP systems and clusters of RISC System/6000 systems.

CDE

IBM XL HPF uses a new GUI based on the CDE in AIX Version 4.1.3, or later. CDE integration consists of an HPF application folder that is integrated within the CDE Application Manager. The XL HPF application folder contains icons representing the HPF tools and applications.

CDE integration of the HPF tools allows programmers to invoke the tools in a simple and consistent manner. The CDE desktop recognizes different types of files using a data type database. A data type identifies the files of a particular format and associates them with the appropriate applications. These associations mean that programmers don′t have to remember command-line invocations of tools. In most cases when a programmer double-clicks on a file, the CDE desktop will automatically launch the correct application that interprets that file′s data.

The HPF application folder contains:

- Live Parsing Extensible (LPEX) editor
- Program Builder
- Debugger(xldb and pdbx)
- HPF online documentation
- Command-line builder (xxlhpf)

LPEX Editor

The LPEX Editor is a language-sensitive, fully programmable editor that supports full F90 and some HPF functions. The LPEX Editor can be used to create and edit many types of text files, including program source and documentation. Using LPEX, developers can:

- Use multiple windows to display several documents or to display more than one view of the same document
- Dynamically configure LPEX to be a multiple-window or single-window tool
- Select a block of text and move or copy it between documents
- Cut and paste to a shell or another application
- Undo previous changes to a document

Developers can customize and extend virtually every aspect of this programmable editor. LPEX is extended through dynamic link libraries. There is no proprietary extension language to learn. With the LPEX API, developers can write powerful extensions to the editor. Also, LPEX provides a rich command language that developers can use to create or modify editor functions. Developers can:

- Define their own fonts and colors
- Modify the editor action key layout
- Add menus to perform frequently used commands (menu definitions can be applied on a filename extension basis)
- Write their own editor commands

Program Builder

The Program Builder, a makefile generator that interprets XL HPF, manages the repetitive tasks of compiling, linking, and correcting errors in program source code. The Program Builder:

- Provides a GUI to simplify the process of setting and saving compile and linker options.
- Supports error browsing from a list display. Selecting a compile error in the list will position the programmer at the error in the source code in the LPEX Editor.
- Creates a makefile that is used by the AIX make command to construct and maintain programs and libraries. The Program Builder also determines build dependencies by scanning the source code files for dependency information.

Debugger (xldb and pdbx)

The xldb is a GUI-based, serial full F90 symbolic debugger. The intuitive GUI allows programmers to control the execution of the program, examine and modify data (variables, storage, and registers), and perform many other useful functions.

The xldb debugger provides machine-level and source-level debugging. It is built around a set of core functions that let developers quickly and efficiently control execution and analyze data. With these core functions, developers can:

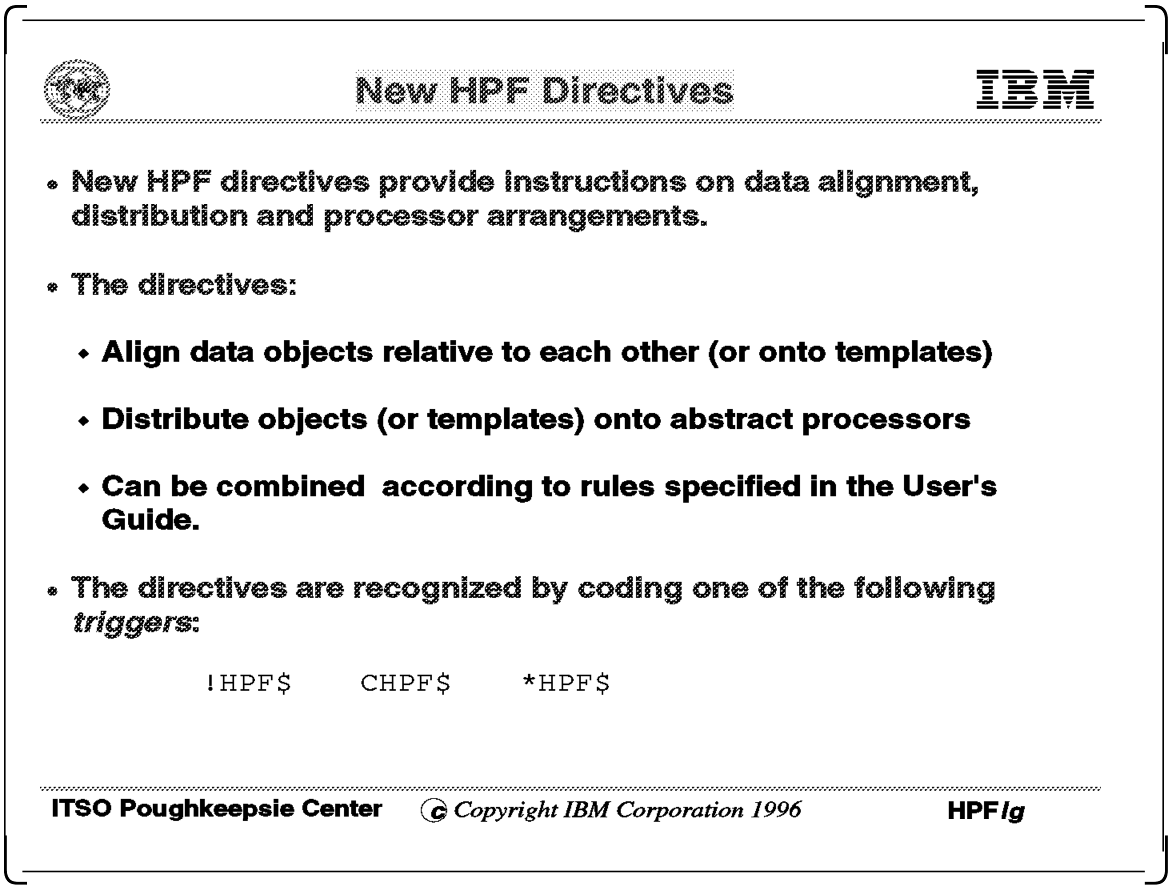
- Display and change variables
- Display and change storage

- Display and change the processor registers
- Display the call stack
- Add and delete simple and complex breakpoints
- Control the execution of multiple threads
- View source code as listing, disassembly, or mixed

The pdbx is a parallel dbx debugger provided by Parallel Environment for AIX Version 2 and can be invoked through the HPF application folder under CDE. The pdbx debugger provides debugging support for Parallel Environment jobs.

Command-Line Builder (xxlhpf)
The xxlhpf is a GUI-based command-line builder that interprets all options available for XL HPF. It simplifies the process of selecting compiler options and helps you to understand what each option does.

5.2.5 New HPF Directives



New HPF Directives

- **New HPF directives provide instructions on data alignment, distribution and processor arrangements.**
- **The directives:**
  - **Align data objects relative to each other (or onto templates)**
  - **Distribute objects (or templates) onto abstract processors**
  - **Can be combined according to rules specified in the User's Guide.**
- **The directives are recognized by coding one of the following *triggers*:**

```
!HPF$    CHPF$    *HPF$
```

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1996* **HPF*lg***

The new HPF directives provide data mapping facilities. The process of mapping data to processors is made up of two steps:

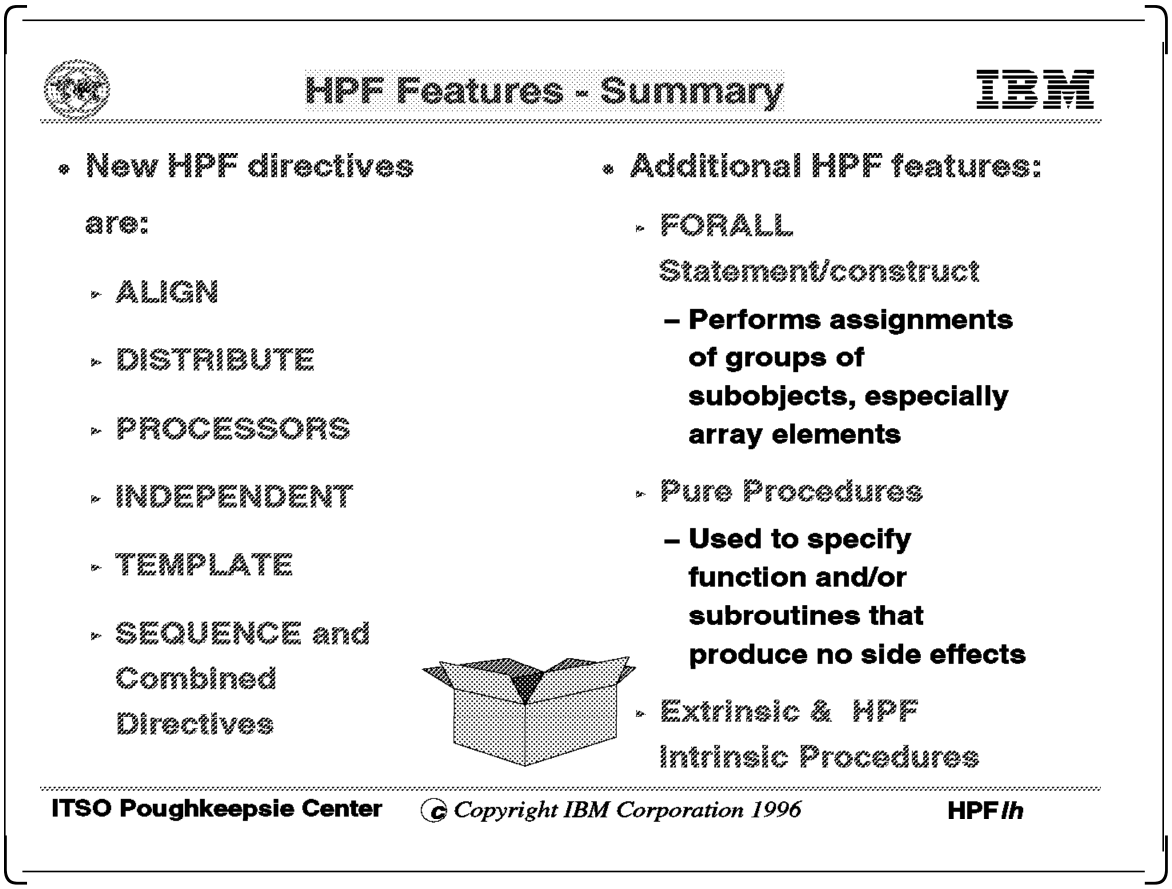
1. Aligning data objects relative to each other or onto templates — this establishes relationships between data
2. Distributing objects (or templates) onto abstract processors — this determines how data is mapped to the machine′s physical resources.

The mapping process is specified in directives. In the Fortran world, directives contain information or instructions destined to the compiler, generally as comments starting with a specific string of character set up at compile time either with the `-qdirective` flag or with the `DIR` keyword in the *@PROCESS* directive. If the `-qdirective` flag is not set up, the default option is `-qnodirective`, and the comments are ignored by the compiler.

For instance, such directives were used for vectorizing FORTRAN 77 programs on IBM mainframes equipped with the vector facility (VF). Though other triggers could be inhibited by the -qnodirective, the *HPF$* trigger for HPF directives is uninhibitable. A typical HPF source code will include directives, such as:

```
|HPF$ TEMPLATE TMPLT(100,100)
```

5.2.6 HPF Features - SUMMARY



HPF Features - Summary

- New HPF directives are:
  - ALIGN
  - DISTRIBUTE
  - PROCESSORS
  - INDEPENDENT
  - TEMPLATE
  - SEQUENCE and Combined Directives
- Additional HPF features:
  - FORALL Statement/construct
    - **Performs assignments of groups of subobjects, especially array elements**
  - Pure Procedures
    - **Used to specify function and/or subroutines that produce no side effects**
  - Extrinsic & HPF Intrinsic Procedures

ITSO Poughkeepsie Center ©Copyright IBM Corporation 1996 HPF*lh*

The new HPF directives that are being made available in this product are:

- ALIGN, which is used to establish a mapping between data objects.
- DISTRIBUTE, which specifies a mapping of data objects to abstract processors in a processor arrangement.
- PROCESSORS, which declares one or more rectilinear processor arrangements, specifying for each name, its rank (number of dimensions), and the extent in each dimension.
- INDEPENDENT, which a programmer uses to assert that no iteration in a DO loop can affect any other iteration, either directly or indirectly.
- TEMPLATE, which defines an abstract space of indexed positions — it can be considered an "array of nothings," as compared to an array of integers, say. Templates can be used as *align-targets*, which can then be distributed.
- SEQUENCE, which specifies that a set of data objects is to be treated as sequential.
- Combined directives may be used to specify multiple HPF attributes for an entity.

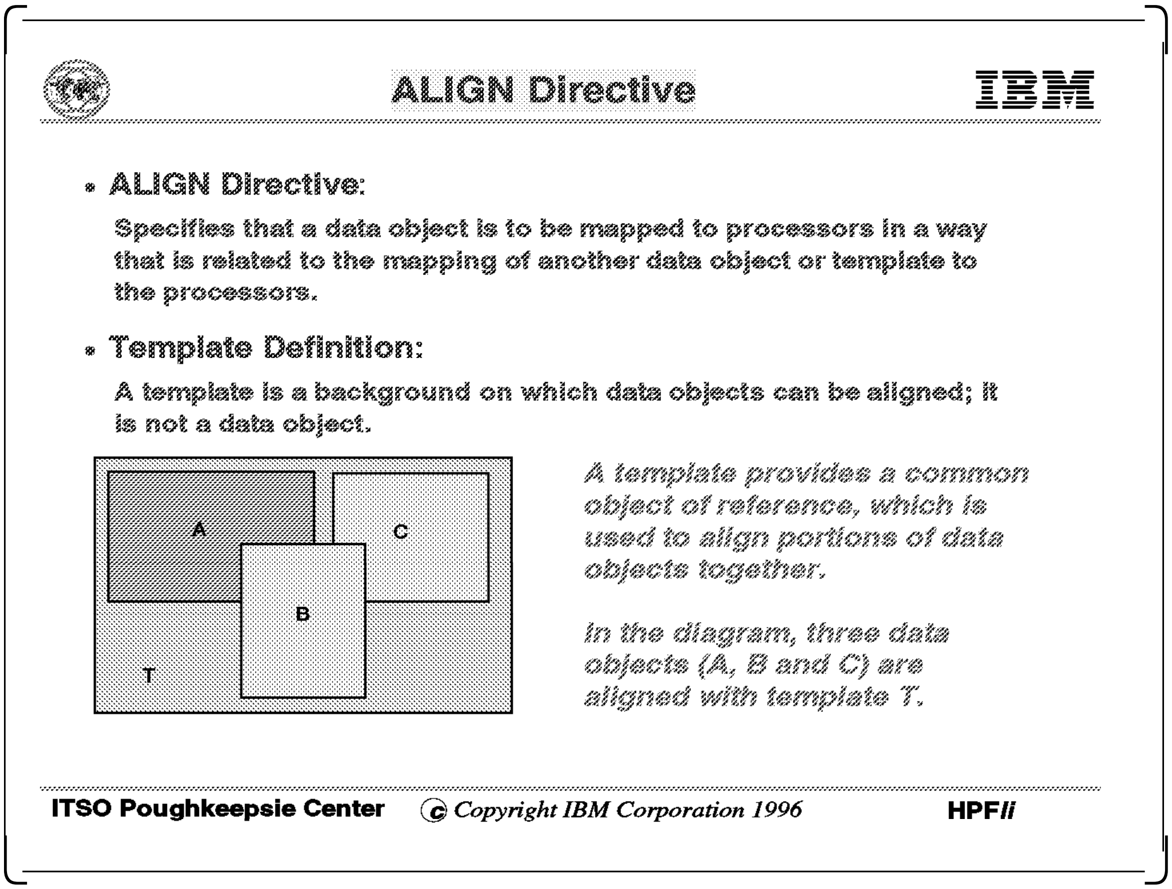
In addition, the following features are also available.

- FORALL statements, which are used to specify an array assignment in terms of array elements or groups of array sections, possibly masked with a scalar logical expression.
- PURE procedures, which are used to specify functions and/or subroutines that produce no side effects. For example, the only effect of a pure function

reference on the state of a program is to return a result — it does not modify the values, pointer associations, or data mapping of any of its arguments or global data, and performs no I/O. A pure subroutine is one that produces no side effects except for modifying the values and/or pointer associations of INTENT(OUT) and INTENT(INOUT) arguments.

- XL HPF includes the Subset HPF Intrinsic procedures, and allows existing non-HPF serial procedures to be called through EXTRINSIC(HPF_LOCAL) or EXTRINSIC(HPF_SERIAL) calls. Also, existing SPMD procedures can be converted to HPF_LOCAL. The HPF_LOCAL can be used to tune the hottest spots.

5.2.7 ALIGN Directive



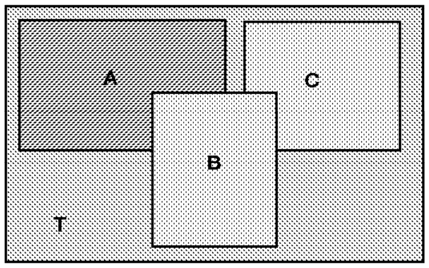
ALIGN Directive

- **ALIGN Directive:**

  Specifies that a data object is to be mapped to processors in a way that is related to the mapping of another data object or template to the processors.

- **Template Definition:**

  A template is a background on which data objects can be aligned; it is not a data object.



A template provides a common object of reference, which is used to align portions of data objects together.

In the diagram, three data objects (A, B and C) are aligned with template T.

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*ii*

The ALIGN directive is used to specify that certain data objects are to be mapped in the same way as certain other data objects. Operations between aligned data objects are likely to be more efficient than operations between data objects that are not known to be aligned (because two objects that are aligned are intended to be mapped to the same abstract processor). The ALIGN directive is designed to make it particularly easy to specify explicit mappings for all the elements of an array at once. While objects can be aligned in some cases through careful use of matching DISTRIBUTE directives, ALIGN is more general and frequently more convenient.

A template is simply an abstract space of indexed positions; it can be considered as an “array of nothings” (as compared to an “array of integers,” say). A template may be used as an abstract align-target that may then be distributed.

The diagram shows three overlapping data objects (A, B and C) that are aligned with the template T.

More Detail on Data Alignment and Distribution

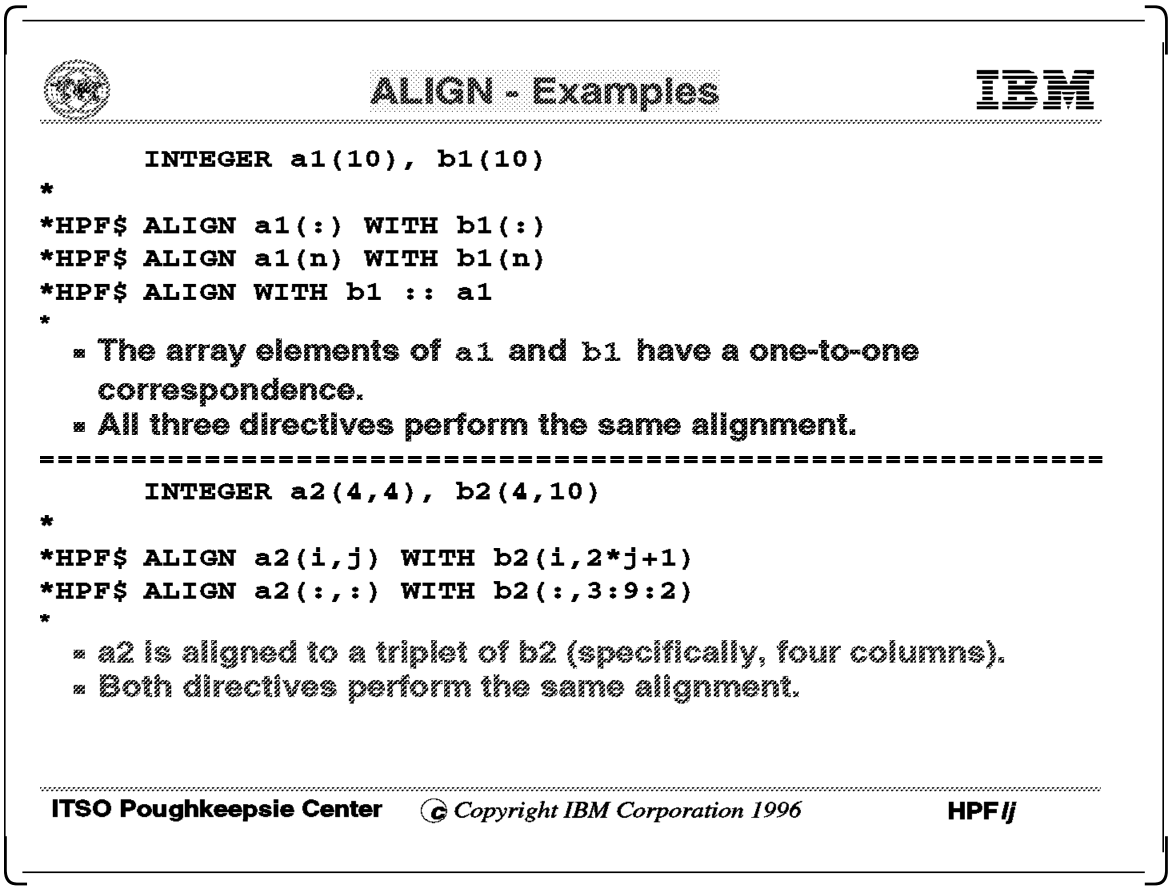
HPF adds directives to FORTRAN 90 to allow the user to advise the compiler on the allocation of data objects to processor memories. So, there is a two-level mapping of data objects to memory regions, referred to as *abstract processors*:

- Data objects (typically array elements) are first *aligned* relative to one another;
- This group of arrays is then *distributed* onto a rectilinear arrangement of abstract processors.

The implementation then uses the same number, or perhaps some smaller number, of physical processors to implement these abstract processors. This mapping of abstract processors to physical processors is implementation-dependent.

The underlying assumptions are that an operation on two or more data objects is likely to be carried out much faster if they all reside in the same processor, and that it may be possible to carry out many such operations concurrently if they can be performed on different processors.

The basic concept is that every array (indeed, every object) is created with *some* alignment to an entity, which in turn has *some* distribution onto *some* arrangement of abstract processors. If the specification statements contain explicit specification directives specifying the alignment of an array A with respect to another array B, then the distribution of A will be dictated by the distribution of B; otherwise, the distribution of A itself may be specified explicitly. In either case, any such explicit declarative information is used when the array is created.



ALIGN - Examples

```
      INTEGER a1(10), b1(10)
*
*HPF$ ALIGN a1(:) WITH b1(:)
*HPF$ ALIGN a1(n) WITH b1(n)
*HPF$ ALIGN WITH b1 :: a1
*
```

- The array elements of `a1` and `b1` have a one-to-one correspondence.
- All three directives perform the same alignment.

```
==========================================================
      INTEGER a2(4,4), b2(4,10)
*
*HPF$ ALIGN a2(i,j) WITH b2(i,2*j+1)
*HPF$ ALIGN a2(:,:) WITH b2(:,3:9:2)
*
```

- a2 is aligned to a triplet of b2 (specifically, four columns).
- Both directives perform the same alignment.

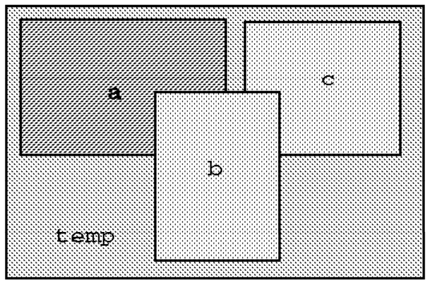
ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*lj*

The first example shows three ALIGN directives, which all perform the same one-to-one alignment between arrays a1 and b1.

In the second example, both ALIGN directives again perform the same alignment; however, the first statement uses dummy arguments, while the second uses fixed values. Here, the four columns of array a1 are aligned with the third, fifth, seventh, and ninth columns of array b1, respectively.

TEMPLATE - Example

```
      INTEGER a(10,20), b(12,8), c(10,10)
*
*HPF$ TEMPLATE temp(20,35)
*HPF$ ALIGN a(i,j) WITH temp(i+1,j+1)
*HPF$ ALIGN b(i,j) WITH temp(i+5,j+18)
*HPF$ ALIGN c(i,j) WITH temp(i+1,j+24)
*
==========================================================
```

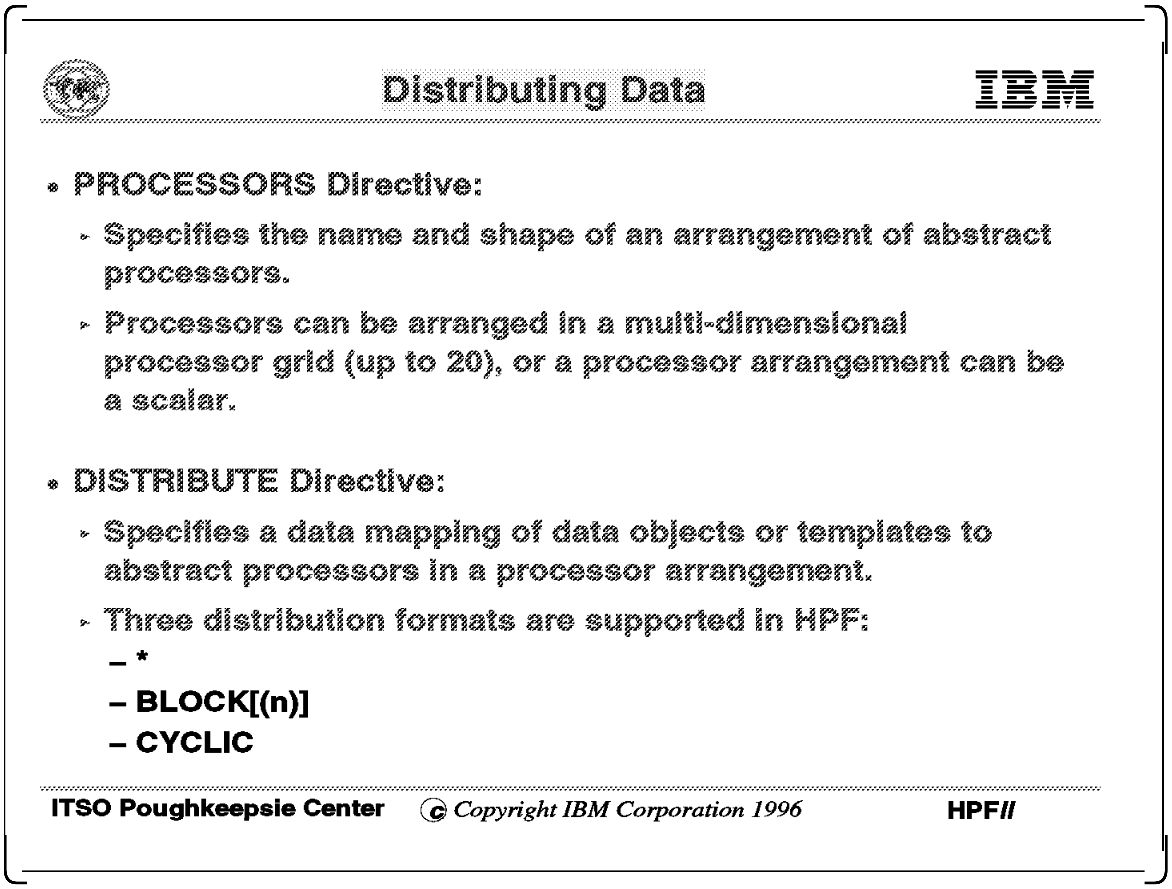


- The TEMPLATE directive declares templates and index spaces associated with objects.
- In the example, regardless of how template temp is distributed, the arrays a, b and c are always properly aligned.

This example shows three arrays all aligned at various positions on a template defined to be of size 20x35. The relationships between the arrays could have been established by two ALIGN statements that aligned arrays a and b, and arrays b and c, but in this case the ALIGN statements would have been more complex.

The template can be considered to be an array whose elements have no content, and therefore occupy no storage; it is merely an abstract index space that can be distributed, and with which arrays may be aligned.

5.2.8 Distributing Data



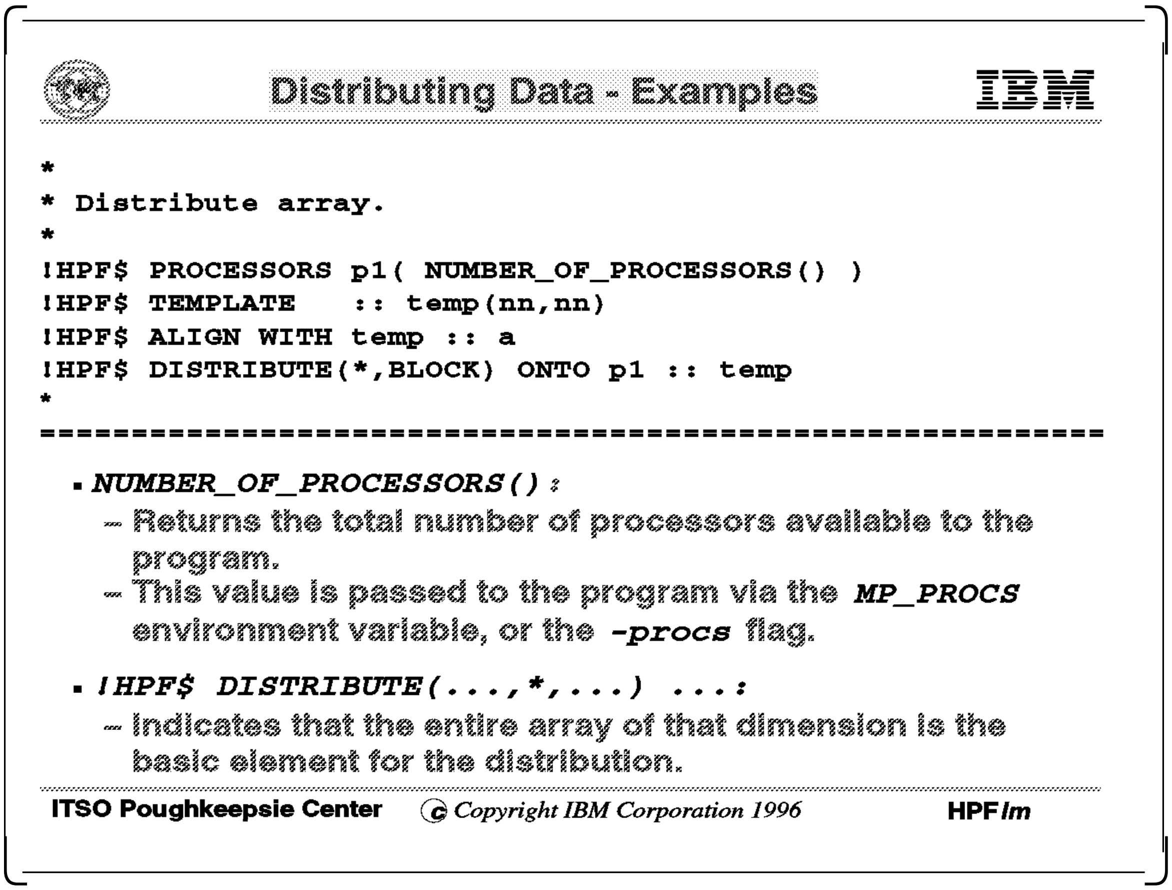
Distributing Data

IBM

- PROCESSORS Directive:
  - Specifies the name and shape of an arrangement of abstract processors.
  - Processors can be arranged in a multi-dimensional processor grid (up to 20), or a processor arrangement can be a scalar.
- DISTRIBUTE Directive:
  - Specifies a data mapping of data objects or templates to abstract processors in a processor arrangement.
  - Three distribution formats are supported in HPF:
    - *
    - **BLOCK[(n)]**
    - **CYCLIC**

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1996* **HPF*//***

The PROCESSORS and DISTRIBUTE directives are used to define a processor arrangement, and then distribute data onto the defined arrangement.



Distributing Data - Examples

```
*
* Distribute array.
*
!HPF$ PROCESSORS p1( NUMBER_OF_PROCESSORS() )
!HPF$ TEMPLATE   :: temp(nn,nn)
!HPF$ ALIGN WITH temp :: a
!HPF$ DISTRIBUTE(*,BLOCK) ONTO p1 :: temp
*
==========================================================
```

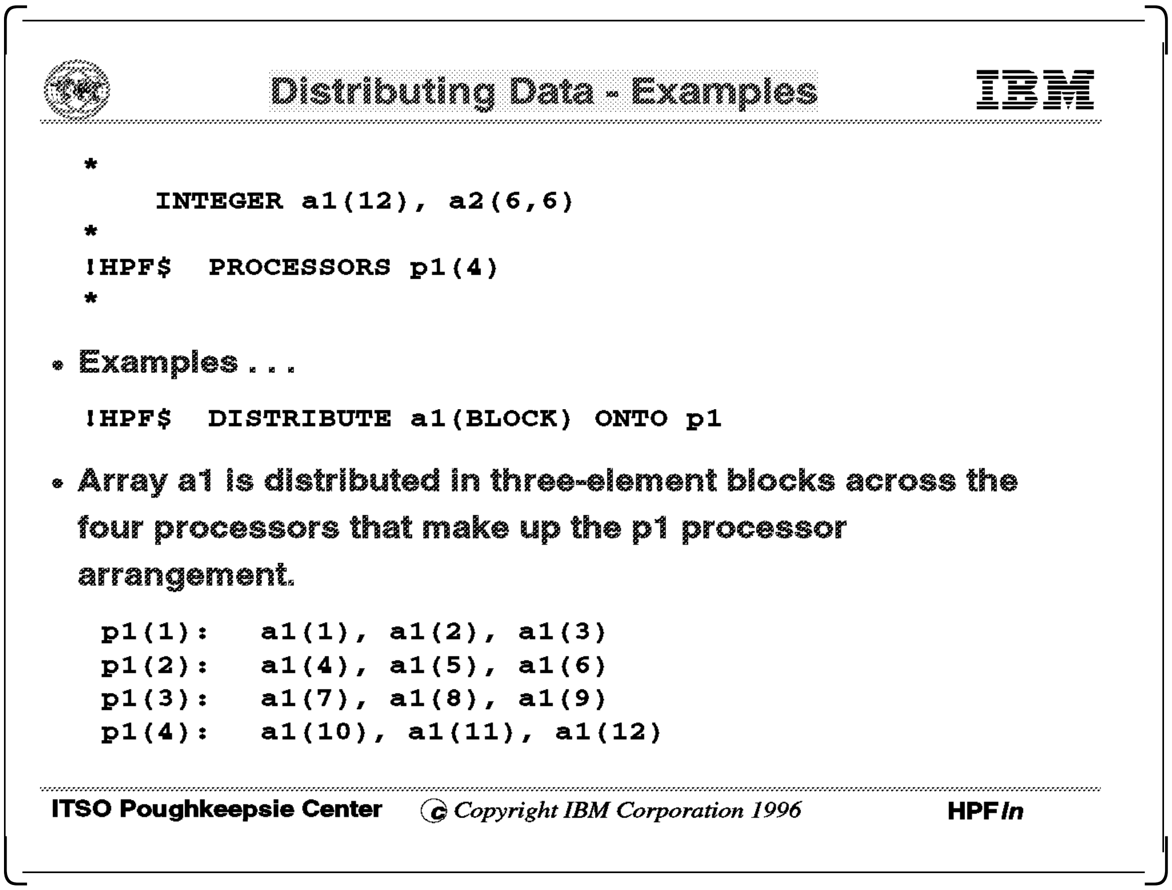
- ***NUMBER_OF_PROCESSORS() :***
  - Returns the total number of processors available to the program.
  - This value is passed to the program via the ***MP_PROCS*** environment variable, or the ***-procs*** flag.
- ***!HPF$ DISTRIBUTE(...,*,...) ...:***
  - Indicates that the entire array of that dimension is the basic element for the distribution.

ITSO Poughkeepsie Center © *Copyright IBM Corporation 1996* HPF/*m*

This example shows:

- The use of the NUMBER_OF_PROCESSORS intrinsic function to determine the number of available processors; this value is used to define a linear array of processors p1.
- A two-dimensional template is defined, and an array a is aligned with it.
- The data in the array is then distributed onto the processor arrangement in a column-BLOCK format; that is all array elements in a given column are mapped to the same processor. For example, if the number of available processors is four (p1(4) has been defined in the PROCESSORS directive), and array a1 is of size 5x10, then,

  Columns [1, 2, 3] are mapped to p1(1),
  Columns [4, 5, 6] are mapped to p1(2),
  Columns [7, 8, 9] are mapped to p1(3),
  Column [10] is mapped to p1(4),



Distributing Data - Examples

```
*
      INTEGER a1(12), a2(6,6)
*
!HPF$  PROCESSORS p1(4)
*
```

- Examples . . .

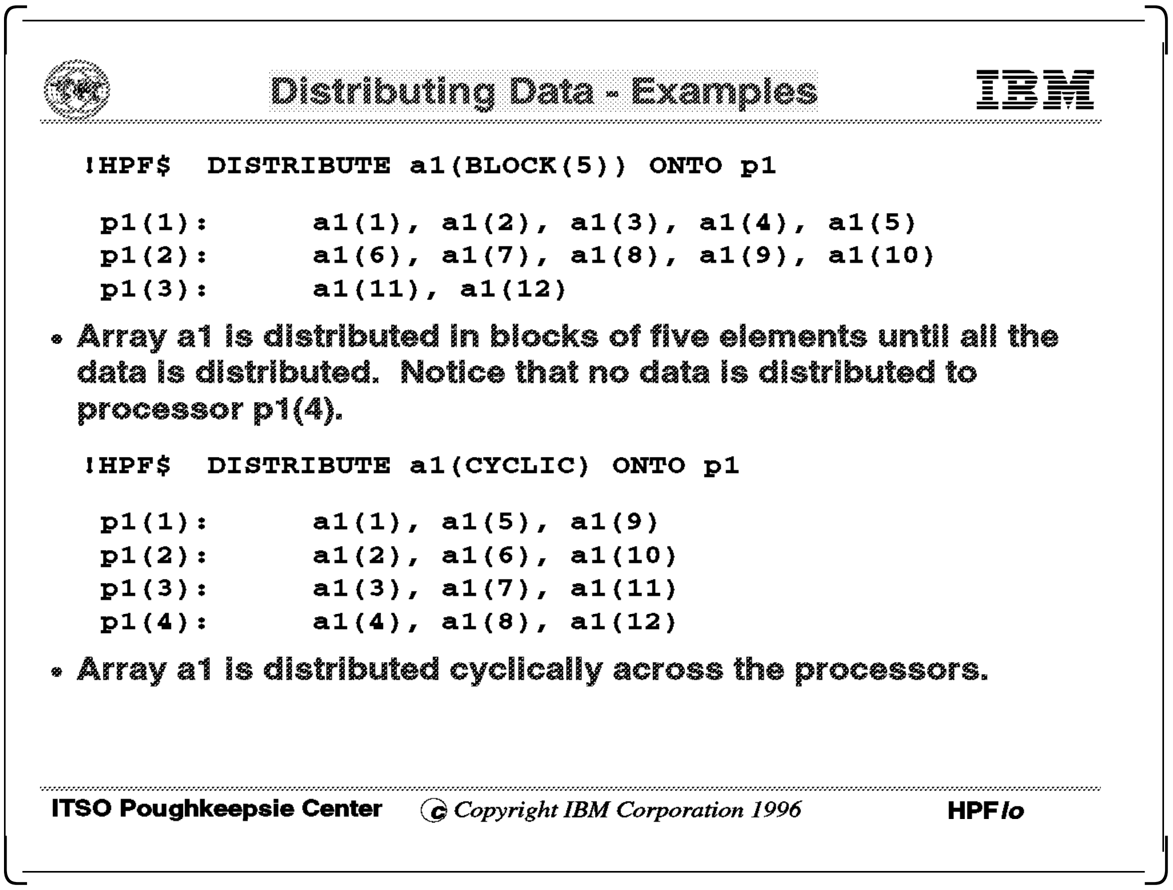
```
!HPF$  DISTRIBUTE a1(BLOCK) ONTO p1
```

- Array a1 is distributed in three-element blocks across the four processors that make up the p1 processor arrangement.

```
 p1(1):   a1(1), a1(2), a1(3)
 p1(2):   a1(4), a1(5), a1(6)
 p1(3):   a1(7), a1(8), a1(9)
 p1(4):   a1(10), a1(11), a1(12)
```

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*In*

This example shows a simple block distribution of the one-dimensional array a1 on the 4-processor arrangement p1. A BLOCK distribution of data implies that each processor is assigned at most one block of data — no "wrap-around" data distribution is allowed. In some cases, one or more processors may not be distributed any data due to the defined blocking format.



Distributing Data - Examples

```
!HPF$  DISTRIBUTE a1(BLOCK(5)) ONTO p1

 p1(1):     a1(1), a1(2), a1(3), a1(4), a1(5)
 p1(2):     a1(6), a1(7), a1(8), a1(9), a1(10)
 p1(3):     a1(11), a1(12)
```

- **Array a1 is distributed in blocks of five elements until all the data is distributed. Notice that no data is distributed to processor p1(4).**

```
!HPF$  DISTRIBUTE a1(CYCLIC) ONTO p1

 p1(1):     a1(1), a1(5), a1(9)
 p1(2):     a1(2), a1(6), a1(10)
 p1(3):     a1(3), a1(7), a1(11)
 p1(4):     a1(4), a1(8), a1(12)
```

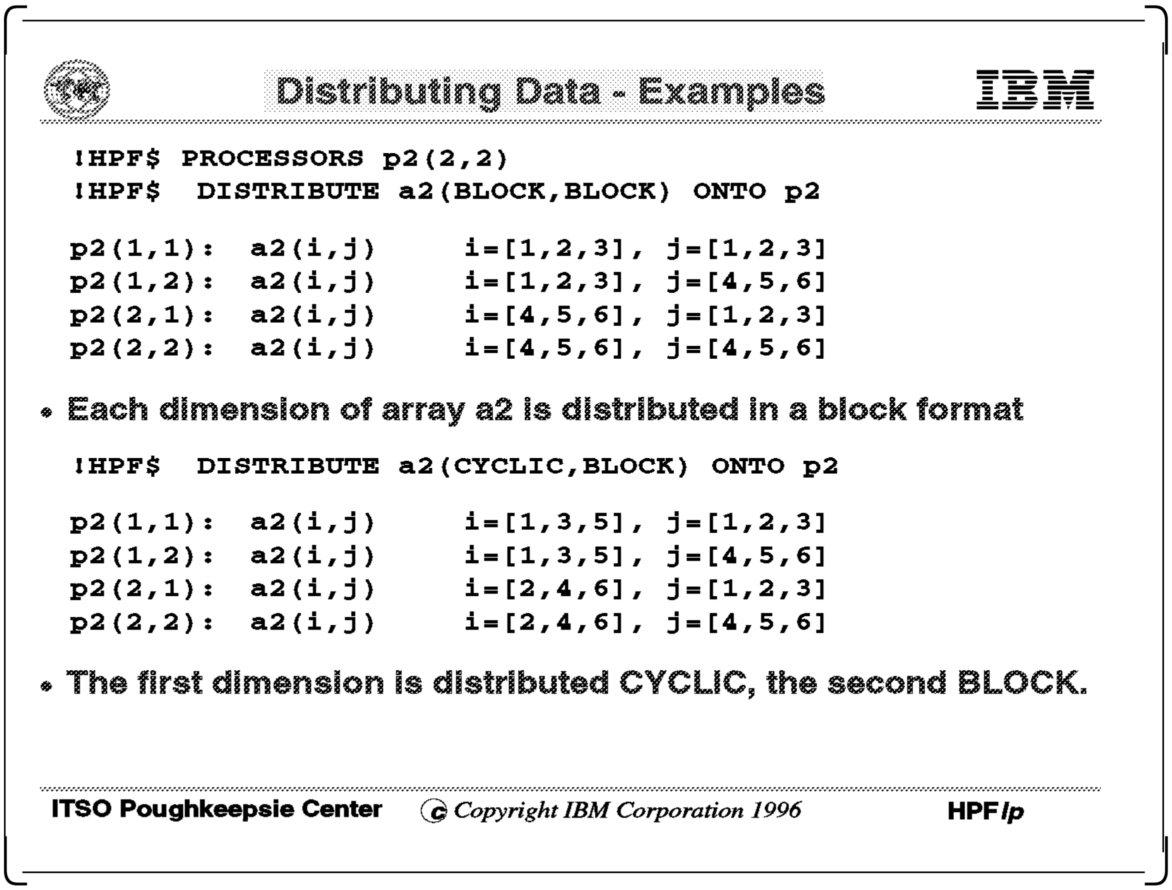
- **Array a1 is distributed cyclically across the processors.**

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1996* **HPF*Io***

The first example shows the elements of array a1 distributed in blocks of five elements until all the data is distributed. Notice that no data is distributed to processor p1(4). All the data must be distributed, but not wrapped around the processor arrangement; therefore, a distribution format of BLOCK(2) would not be valid, since only the first eight elements would be distributed.

The second example illustrates the CYCLIC method of data distribution. The period of the data cycle is one, so the elements of the array are distributed in an even round-robin fashion. If the distribution format was CYCLIC(2), then successive pairs of the elements of a1 would be distributed onto the processor arrangement, with p1(1) and p1(2) each having an additional pair.

Note: In the current Beta release of the XL HPF Compiler, CYCLIC(n) distribution with n > 1 is only supported in HPF_LOCAL interface bodies.



Distributing Data - Examples

```
!HPF$ PROCESSORS p2(2,2)
!HPF$  DISTRIBUTE a2(BLOCK,BLOCK) ONTO p2

p2(1,1):  a2(i,j)     i=[1,2,3], j=[1,2,3]
p2(1,2):  a2(i,j)     i=[1,2,3], j=[4,5,6]
p2(2,1):  a2(i,j)     i=[4,5,6], j=[1,2,3]
p2(2,2):  a2(i,j)     i=[4,5,6], j=[4,5,6]
```

- **Each dimension of array a2 is distributed in a block format**

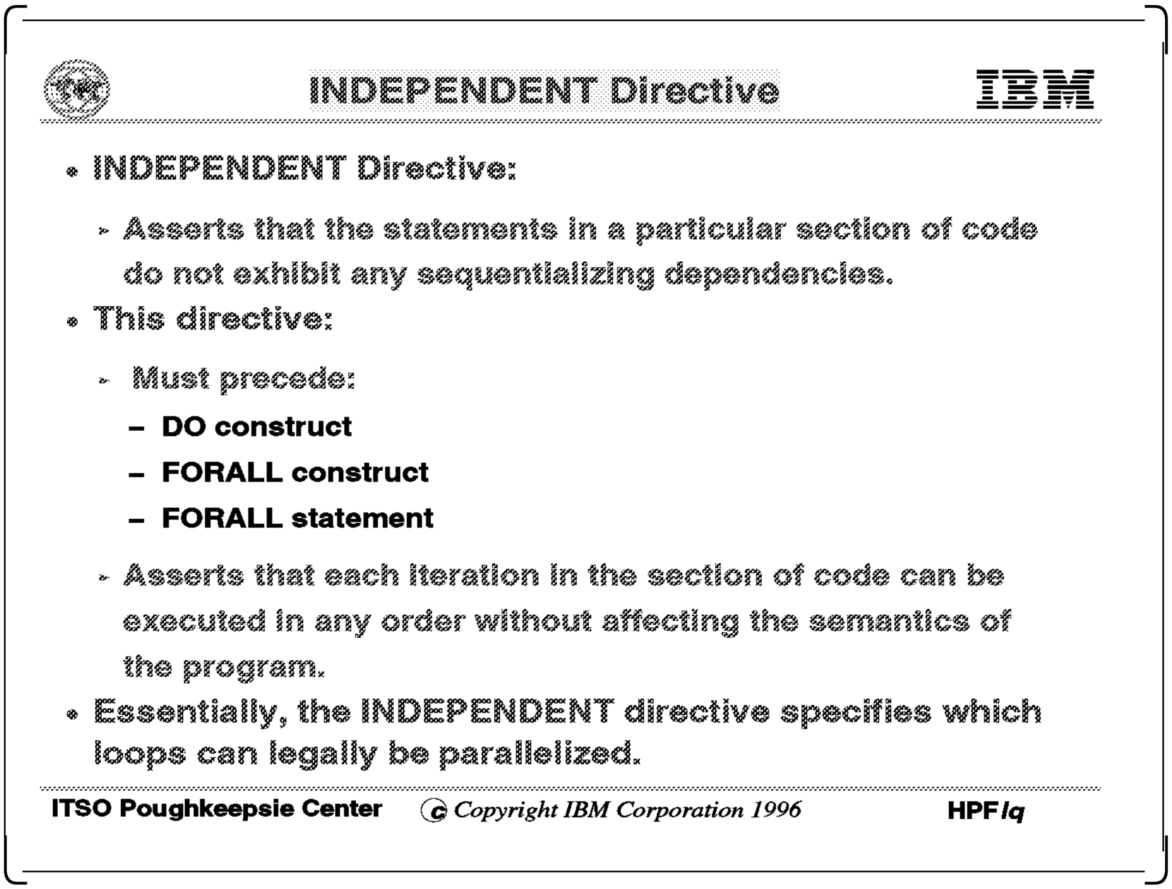
```
!HPF$  DISTRIBUTE a2(CYCLIC,BLOCK) ONTO p2

p2(1,1):  a2(i,j)     i=[1,3,5], j=[1,2,3]
p2(1,2):  a2(i,j)     i=[1,3,5], j=[4,5,6]
p2(2,1):  a2(i,j)     i=[2,4,6], j=[1,2,3]
p2(2,2):  a2(i,j)     i=[2,4,6], j=[4,5,6]
```

- **The first dimension is distributed CYCLIC, the second BLOCK.**

The first example shows the BLOCK distribution of data across both dimensions of a processor array. In essence, each processor in arrangement p2 receives a 3x3 submatrix of array a2.

The second example illustrates the use of both data distribution methods in distributing an array onto a processor arrangement. The rows are distributed cyclically, and the columns are block distributed — the first three columns of a2 are distributed cyclically on processors p2(1,1) and p2(2,1), while the next three columns of a2 are distributed cyclically on processors p2(1,2) and p2(2,2).

5.2.9 INDEPENDENT Directive



INDEPENDENT Directive

- INDEPENDENT Directive:
  - Asserts that the statements in a particular section of code do not exhibit any sequentializing dependencies.
- This directive:
  - Must precede:
    - DO construct
    - FORALL construct
    - FORALL statement
  - Asserts that each iteration in the section of code can be executed in any order without affecting the semantics of the program.
- Essentially, the INDEPENDENT directive specifies which loops can legally be parallelized.

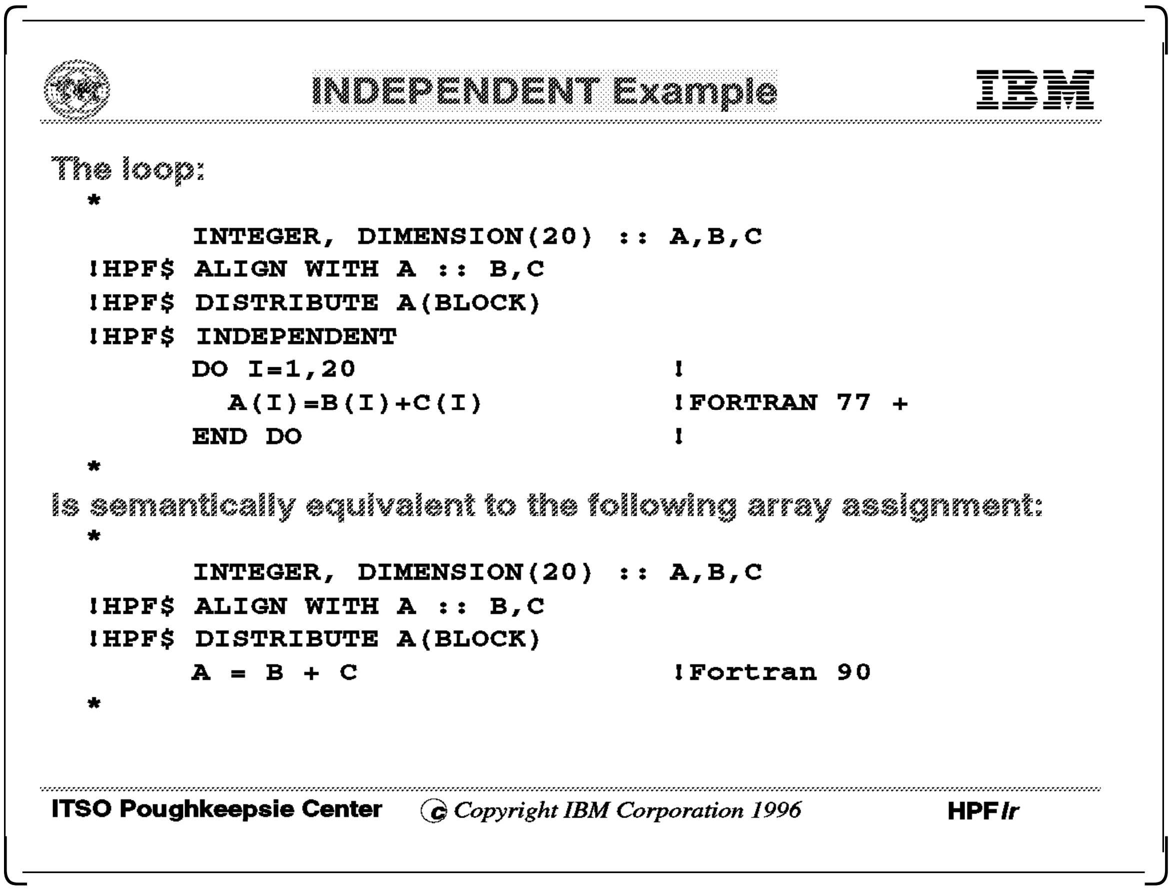
ITSO Poughkeepsie Center © *Copyright IBM Corporation 1996* HPF*lq*

When parallelizing a program, one must understand that some algorithms are not eligible for parallelization, such as recurrent algorithms or routines with a side effect. In such cases, the result depends on the order of operation execution, which is incompatible with the simultaneous execution of these operations in parallel and independent processes.

To help the compiler determine whether DO constructs, and FORALL construct/statements are eligible or not, the programmer must put an INDEPENDENT directive before them.

The INDEPENDENT directive must precede a DO construct, FORALL construct, or FORALL statement, and it specifies that each operation in these statements or constructs can be executed in any order without affecting the semantics of the program.

The INDEPENDENT directive provides an assertion that it is legal to parallelize a specific loop, in those cases where the HPF compiler cannot determine whether it is legal or not.



INDEPENDENT Example

The loop:

```
  *
        INTEGER, DIMENSION(20) :: A,B,C
  !HPF$ ALIGN WITH A :: B,C
  !HPF$ DISTRIBUTE A(BLOCK)
  !HPF$ INDEPENDENT
        DO I=1,20                   !
          A(I)=B(I)+C(I)            !FORTRAN 77 +
        END DO                      !
  *
```

is semantically equivalent to the following array assignment:

```
  *
        INTEGER, DIMENSION(20) :: A,B,C
  !HPF$ ALIGN WITH A :: B,C
  !HPF$ DISTRIBUTE A(BLOCK)
        A = B + C                   !Fortran 90
  *
```

ITSO Poughkeepsie Center *© Copyright IBM Corporation 1996* HPF*Ir*

The example shows two methods of adding the arrays B and C, with the results stored in array A. Each iteration of the DO loop can be executed in any order without affecting the semantics of this piece of code.

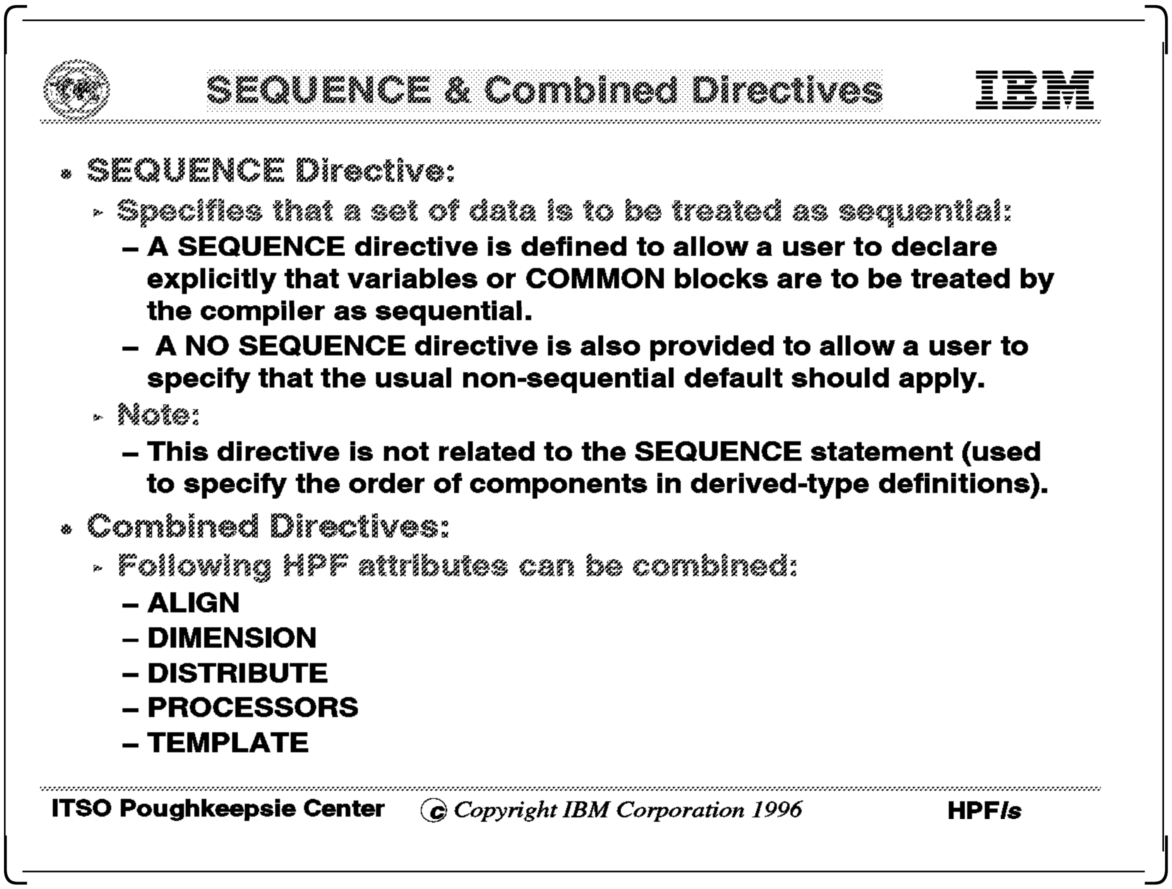
The second example, semantically equivalent to the first, does not need an INDEPENDENT directive because there is no ambiguity in the Fortran 90 statement.

But the classic recurrent DO loop could not be considered as independent:

```
DO i=2,n
  a(i) = i + a(i-1)
ENDDO
```

In cases such as this, where there are dependencies between the left and the right sides of an assignment statement, an independent directive must not be coded since the assertion of independent iterations is false.

5.2.10 SEQUENCE and Combined Directives



SEQUENCE & Combined Directives

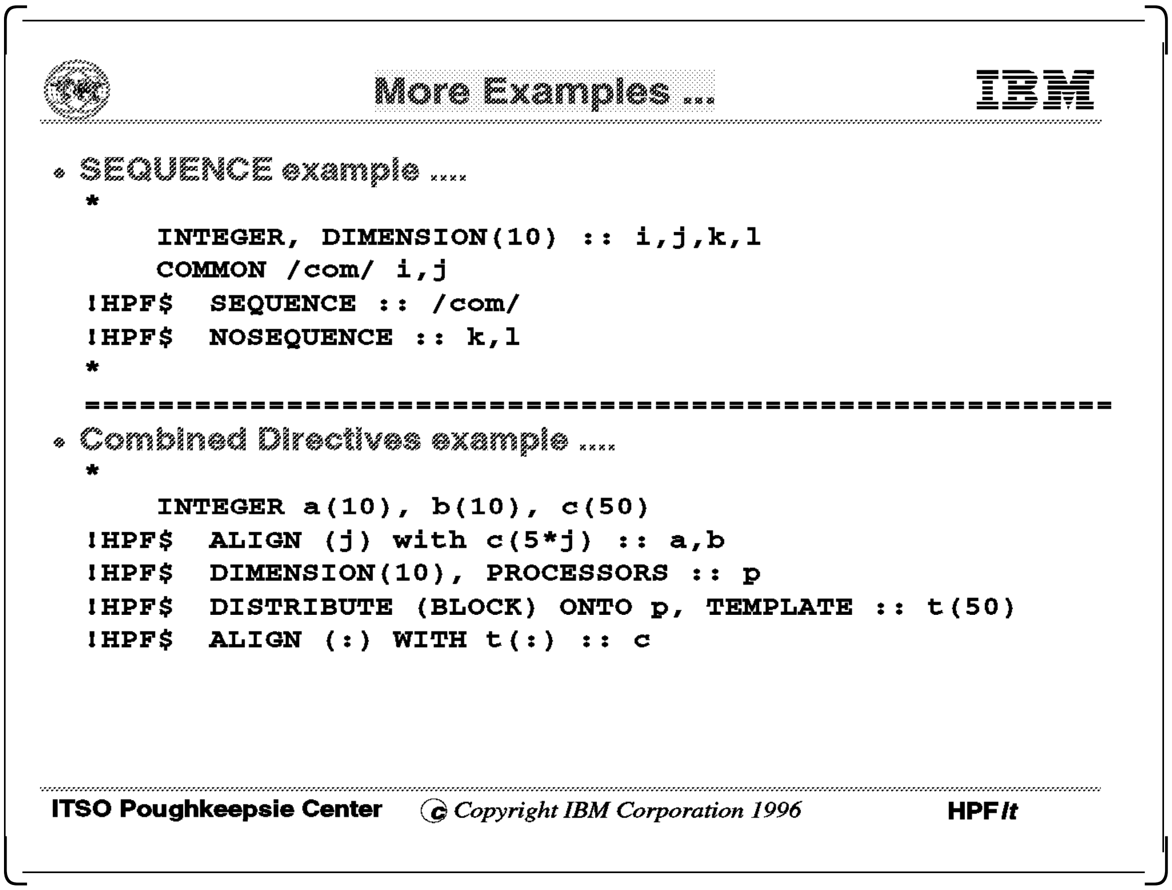
- **SEQUENCE Directive:**
  - **Specifies that a set of data is to be treated as sequential:**
    - A SEQUENCE directive is defined to allow a user to declare explicitly that variables or COMMON blocks are to be treated by the compiler as sequential.
    - A NO SEQUENCE directive is also provided to allow a user to specify that the usual non-sequential default should apply.
  - **Note:**
    - This directive is not related to the SEQUENCE statement (used to specify the order of components in derived-type definitions).
- **Combined Directives:**
  - **Following HPF attributes can be combined:**
    - ALIGN
    - DIMENSION
    - DISTRIBUTE
    - PROCESSORS
    - TEMPLATE

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1996* HPF*ls*

The SEQUENCE directive can be used to assert that full sequence and storage association for affected variables must be maintained.

Note: A goal of HPF was to maintain compatibility with Fortran 90. Full support of Fortran sequence and storage association, however, is not compatible with the goal of high performance through distribution of data in HPF. Some forms of associating subprogram dummy arguments with actual values make assumptions about the sequence of values in physical memory, which may be incompatible with data distribution. Certain forms of EQUIVALENCE statements are recognized as requiring a modified storage association paradigm. In both cases, the SEQUENCE directive directs the compiler to treat the declared variables as sequential.

Combined directives can specify multiple HPF attributes for an entity, in the same way that the Fortran 90 type declaration statement can specify multiple attributes for an entity.



More Examples ...

- SEQUENCE example ....

```
*
     INTEGER, DIMENSION(10) :: i,j,k,l
     COMMON /com/ i,j
!HPF$  SEQUENCE :: /com/
!HPF$  NOSEQUENCE :: k,l
*
==============================================================
```

- Combined Directives example ....

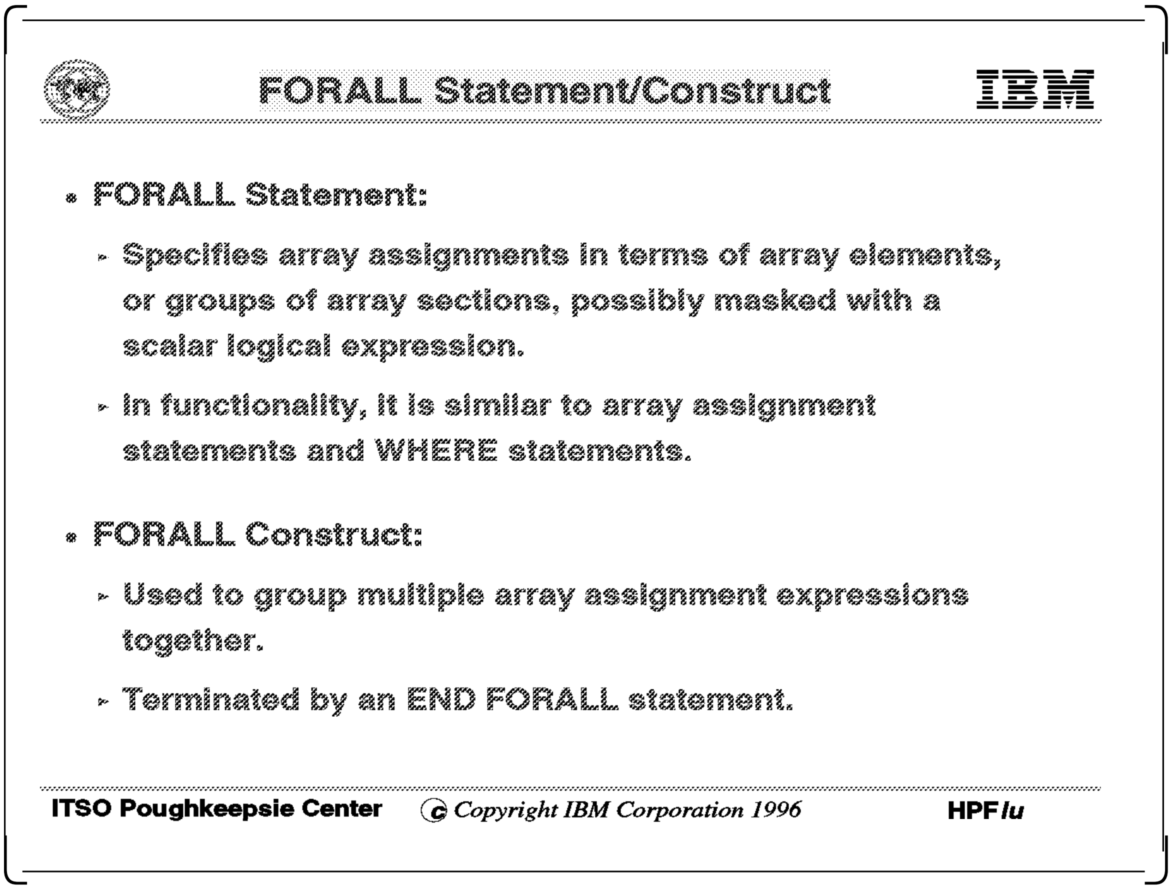
```
*
     INTEGER a(10), b(10), c(50)
!HPF$  ALIGN (j) with c(5*j) :: a,b
!HPF$  DIMENSION(10), PROCESSORS :: p
!HPF$  DISTRIBUTE (BLOCK) ONTO p, TEMPLATE :: t(50)
!HPF$  ALIGN (:) WITH t(:) :: c
```

ITSO Poughkeepsie Center   © Copyright IBM Corporation 1996   HPF*It*

The SEQUENCE directive example shows variables i and j. Members of the com COMMON area are to be treated as sequential, while variables k and l are treated normally.

The Combined directives example shows the DIMENSION and PROCESSORS directives combined in the same statement.

5.2.11 FORALL Statement

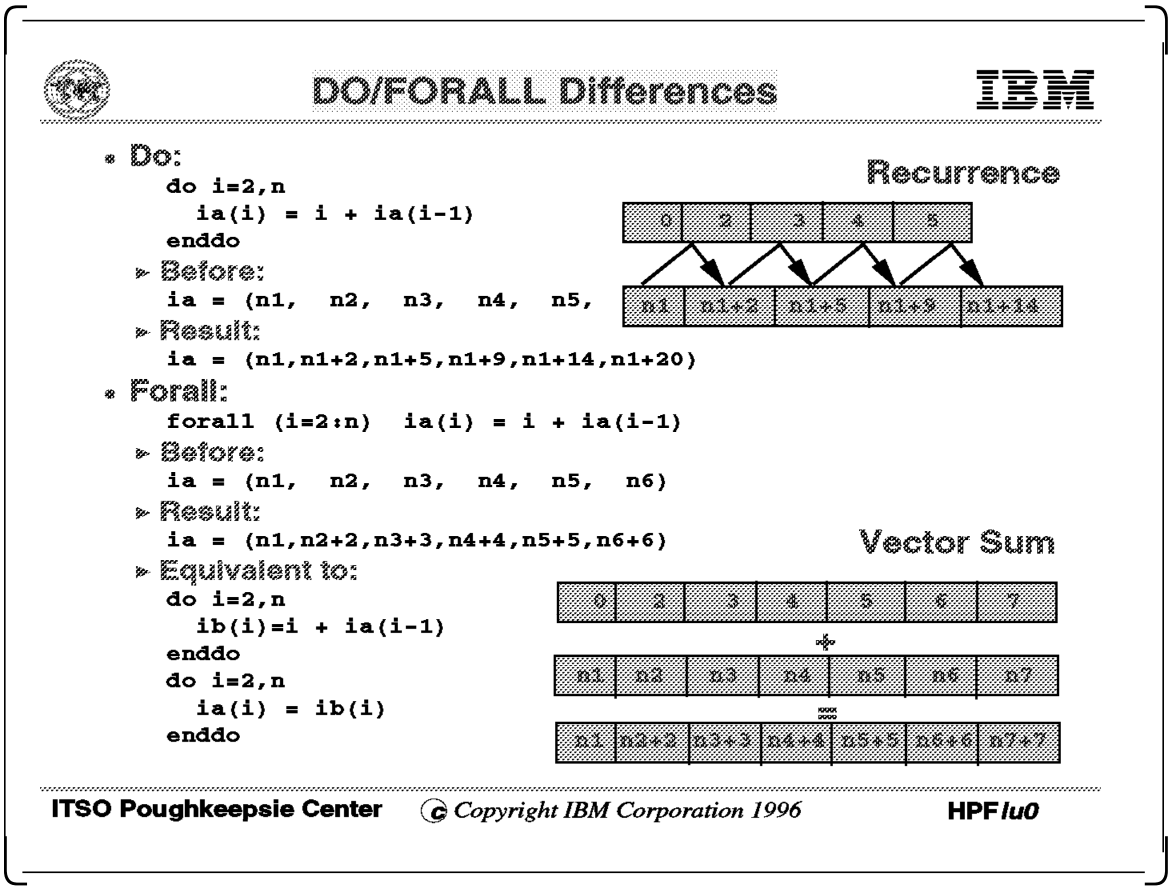


FORALL Statement/Construct

- FORALL Statement:
  - Specifies array assignments in terms of array elements, or groups of array sections, possibly masked with a scalar logical expression.
  - In functionality, it is similar to array assignment statements and WHERE statements.
- FORALL Construct:
  - Used to group multiple array assignment expressions together.
  - Terminated by an END FORALL statement.

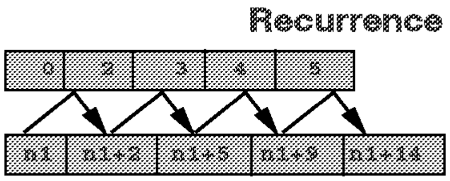
ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*lu*

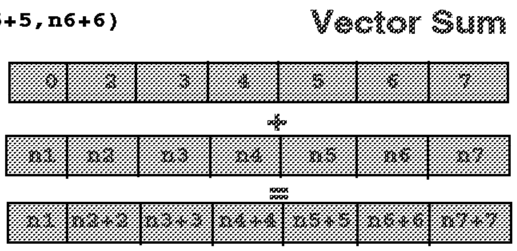
The FORALL statement is used to specify an array assignment in terms of array elements or groups of array sections, possibly masked with a scalar logical expression. In functionality, it is similar to array assignment statements and WHERE statements. The statement overcomes the several restrictions that Fortran 90 places on array assignments. In particular, Fortran 90 requires that operands of the right side expressions be conformable with the left hand side array. Also, FORALL permits you to select more irregular blocks for assignment, such as the main diagonal of an array.



DO/FORALL Differences

- **Do:**
```
do i=2,n
  ia(i) = i + ia(i-1)
enddo
```
  - Before:
```
ia = (n1,  n2,  n3,  n4,  n5,
```
  - Result:
```
ia = (n1,n1+2,n1+5,n1+9,n1+14,n1+20)
```
- **Forall:**
```
forall (i=2:n)  ia(i) = i + ia(i-1)
```
  - Before:
```
ia = (n1,  n2,  n3,  n4,  n5,  n6)
```
  - Result:
```
ia = (n1,n2+2,n3+3,n4+4,n5+5,n6+6)
```
  - Equivalent to:
```
do i=2,n
  ib(i)=i + ia(i-1)
enddo
do i=2,n
  ia(i) = ib(i)
enddo
```

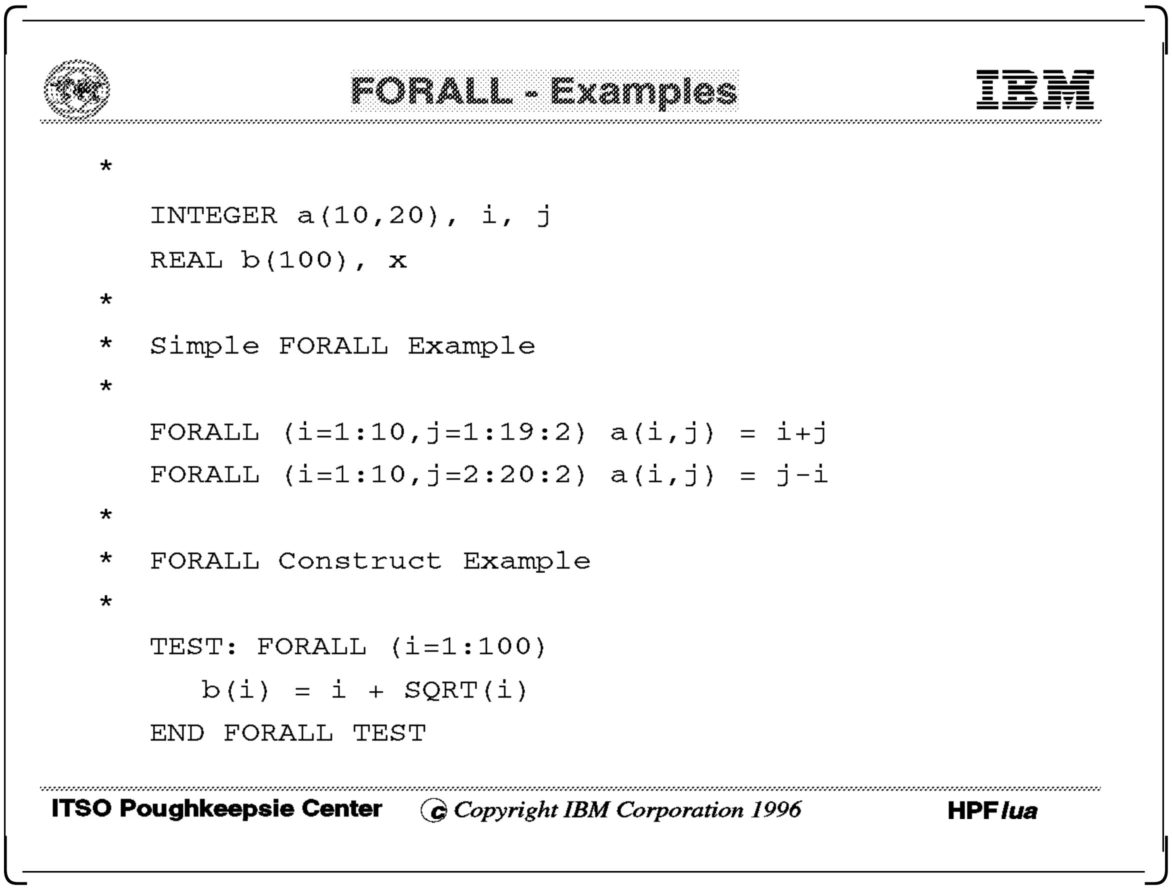




ITSO Poughkeepsie Center © *Copyright IBM Corporation 1996* **HPF*lu0***

The DO loop given in this foil introduces a side effect because the Fortran design specifies that the statements in the loop are serially executed. So, as it is, this DO loop could not be parallelized.

On the contrary, the FORALL design specifies that the instruction execution is independent from the range in the loop. In fact, the ia vector coordinates from 2 to n are added to an implicit vector (2,3,...,n) into a work area. Then, the work area is copied onto ia starting at ia(2).



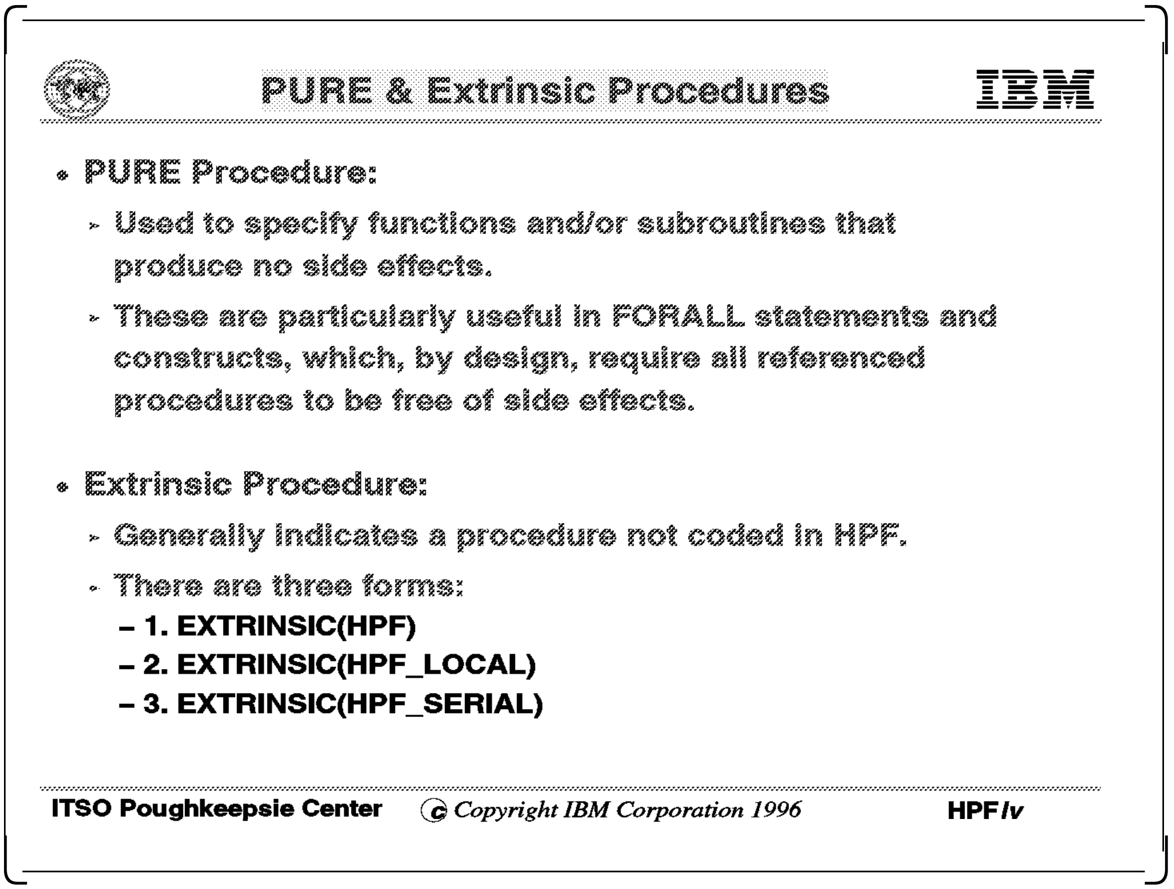
FORALL - Examples

```
*
   INTEGER a(10,20), i, j
   REAL b(100), x
*
*  Simple FORALL Example
*
   FORALL (i=1:10,j=1:19:2) a(i,j) = i+j
   FORALL (i=1:10,j=2:20:2) a(i,j) = j-i
*
*  FORALL Construct Example
*
   TEST: FORALL (i=1:100)
      b(i) = i + SQRT(i)
   END FORALL TEST
```

ITSO Poughkeepsie Center   © *Copyright IBM Corporation 1996*   **HPF*lua***

This example code illustrates both the FORALL statement and FORALL construct. In the construct example, the construct name is TEST, and the construct is terminated by the END FORALL statement.

5.2.12 PURE and Extrinsic Procedures



PURE & Extrinsic Procedures

IBM

- PURE Procedure:
  - Used to specify functions and/or subroutines that produce no side effects.
  - These are particularly useful in FORALL statements and constructs, which, by design, require all referenced procedures to be free of side effects.
- Extrinsic Procedure:
  - Generally indicates a procedure not coded in HPF.
  - There are three forms:
    - 1. EXTRINSIC(HPF)
    - 2. EXTRINSIC(HPF_LOCAL)
    - 3. EXTRINSIC(HPF_SERIAL)

ITSO Poughkeepsie Center © Copyright IBM Corporation 1996 HPF*lv*

PURE procedures are used to specify functions and/or subroutines that produce no side effects. For example, the only effect of a pure function reference on the state of a program is to return a result — it does not modify the values, pointer associations, or data mapping of any of its arguments or global data, and performs no external I/O (READ from and WRITE to internal files is allowed).

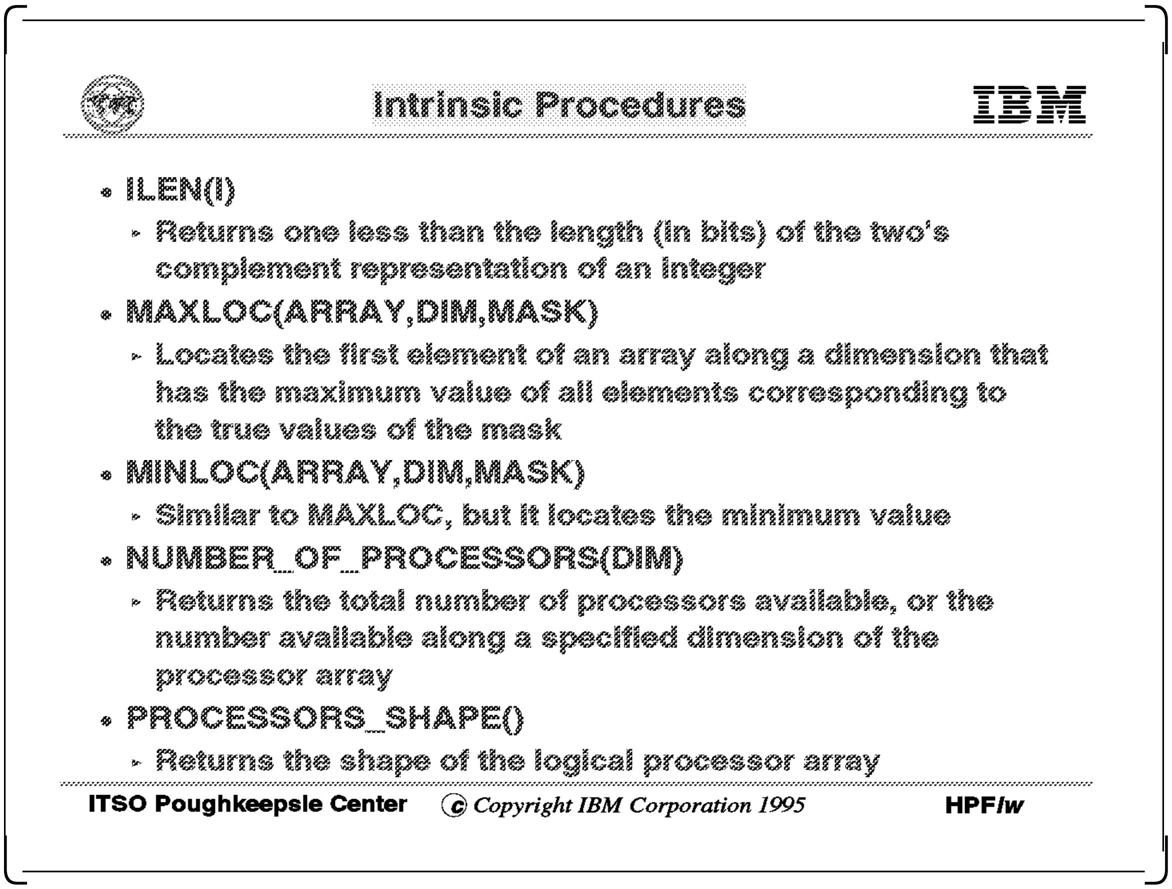
A pure subroutine is one that produces no side effects except for modifying the values and/or pointer associations of INTENT(OUT) and INTENT(INOUT) arguments.

Existing non-HPF serial procedures can be called through EXTRINSIC(HPF_LOCAL) or EXTRINSIC(HPF_SERIAL) calls and including SPMD procedure.

- EXTRINSIC(HPF_LOCAL) indicates a local procedure that is targeted to a single processor, with many copies executing on different processors. This programming style is referred to as SPMD (single program, multiple data).
- EXTRINSIC(HPF_SERIAL) indicates a serial procedure that is targeted to a single processor, with only one instance of the procedure executing on only one processor. Serial procedures are useful for code written in other languages that is not required or desirable to recode.

EXTRINSIC(HPF) procedures are HPF-conforming and are known as HPF procedures. If a procedure does not specify the EXTRINSIC attribute, and is compiled with the *-qhpf* option, then it is considered to be an HPF procedure; this is the default.

5.2.13 Intrinsic Procedures



Intrinsic Procedures

- ILEN(I)
  - Returns one less than the length (in bits) of the two's complement representation of an integer
- MAXLOC(ARRAY,DIM,MASK)
  - Locates the first element of an array along a dimension that has the maximum value of all elements corresponding to the true values of the mask
- MINLOC(ARRAY,DIM,MASK)
  - Similar to MAXLOC, but it locates the minimum value
- NUMBER_OF_PROCESSORS(DIM)
  - Returns the total number of processors available, or the number available along a specified dimension of the processor array
- PROCESSORS_SHAPE()
  - Returns the shape of the logical processor array

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 HPF/w

The Subset HPF Intrinsic procedures are also included with XL HPF; these include: ILEN(I), NUMBER_OF_PROCESSORS() and PROCESSOR_SHAPE(). In addition, the Fortran 90 intrinsic procedures MAXLOC and MINLOC have been extended to provide a DIM (dimension) argument.

MAXLOC(array,dim,mask)
This Fortran 90 intrinsic procedure has been improved to select the dimension in a n-dimension array. For instance, if you want to know the maximum value per column in a matrix, you can use:

```
res = MAXLOC(mat,DIM=2)
```

MINLOC(array,dim,mask)
As MAXLOC, the HPF MINLOC intrinsic procedure includes the DIM argument.

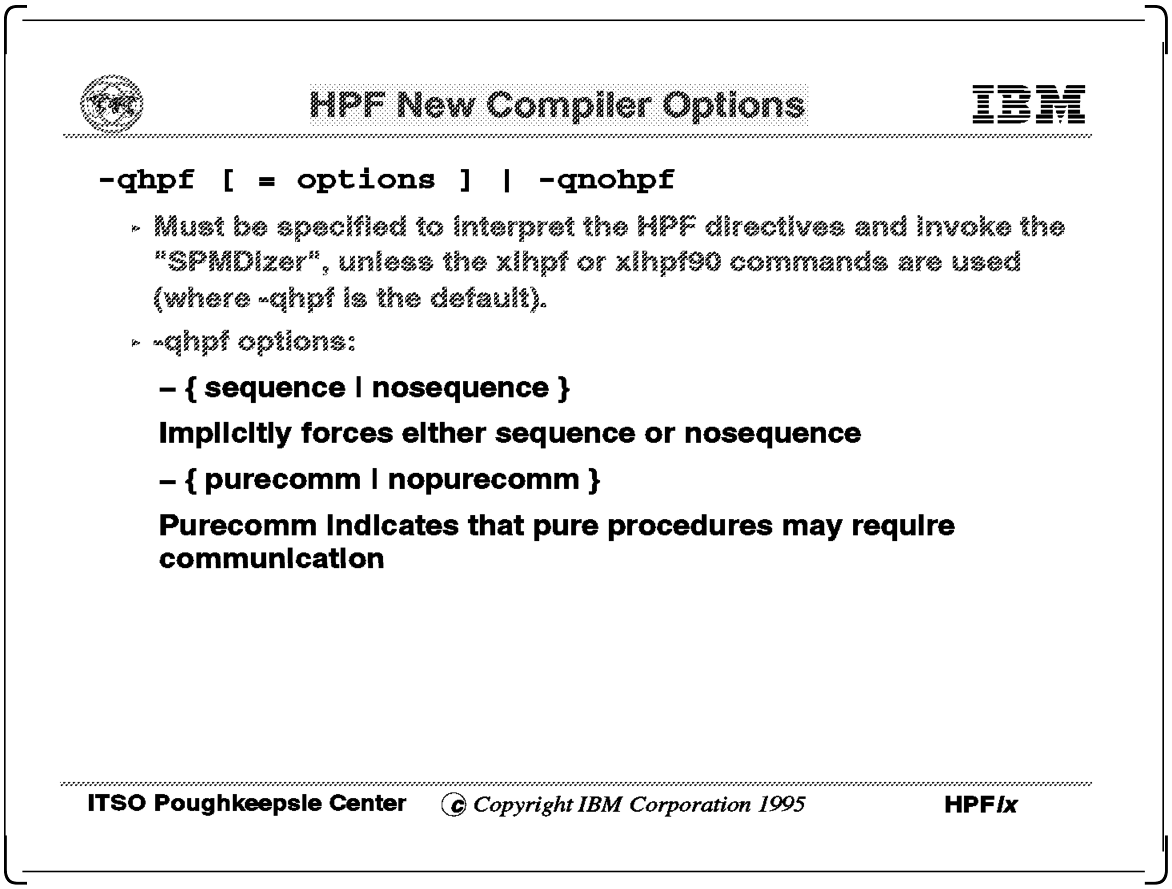
NUMBER_OF_PROCESSORS(dim)
This intrinsic procedure returns the execution time number of processors set up through the -procs flag of the poe command. If dim is present it must equal 1.

PROCESSORS_SHAPE()
This intrinsic function returns an integer that is the rank of the processor array.

ILEN(i)

If i is greater than or equal to 0, $ILEN(i) = CEILING(\log_2(I + 1))$.
If i is negative, $ILEN(i) = CEILING(\log_2(-i))$.

5.2.14 HPF Compiler Options



HPF New Compiler Options

-qhpf [= options] | -qnohpf

- Must be specified to interpret the HPF directives and invoke the "SPMDizer", unless the xlhpf or xlhpf90 commands are used (where -qhpf is the default).
- -qhpf options:
  - **– { sequence | nosequence }**
    Implicitly forces either sequence or nosequence
  - **– { purecomm | nopurecomm }**
    Purecomm indicates that pure procedures may require communication

ITSO Poughkeepsie Center ⓒ *Copyright IBM Corporation 1995* HPF/x

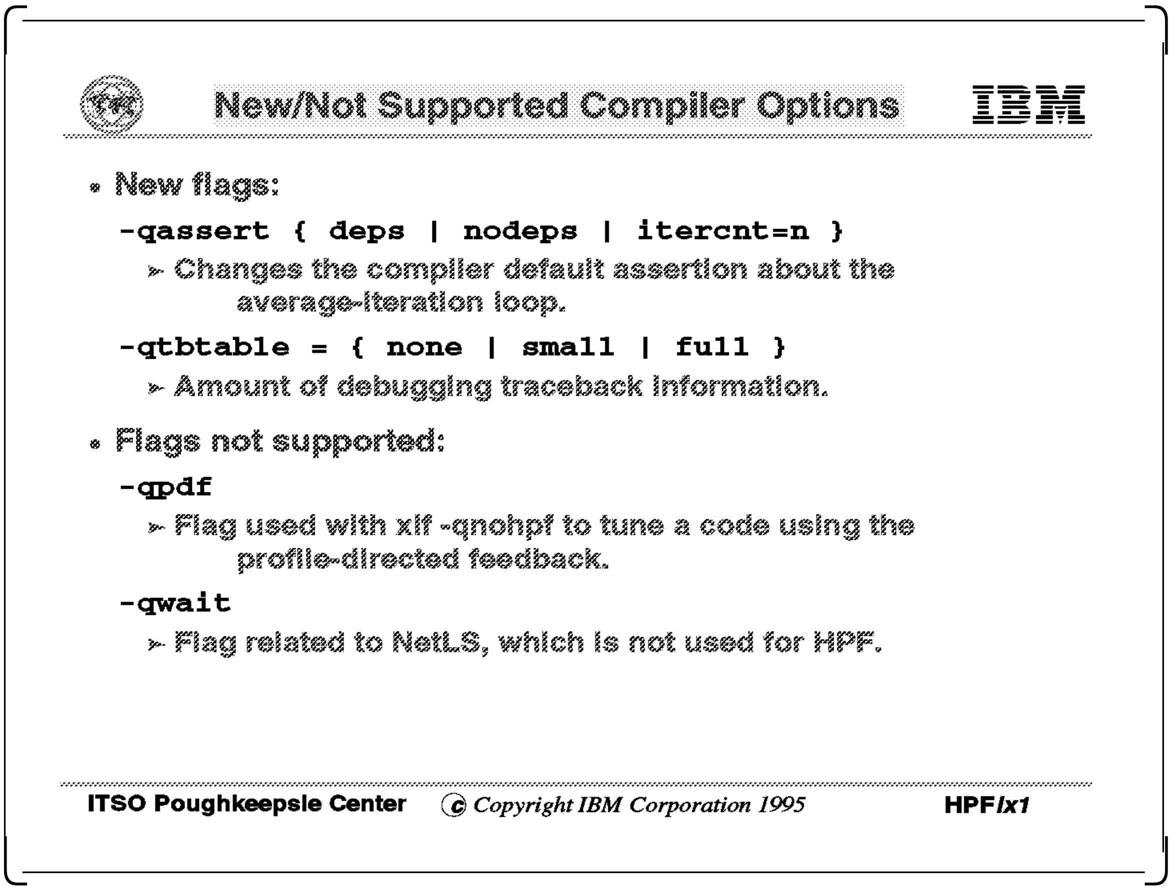
The new HPF compiler options are set up through the following flags:

-qhpf | -qnohpf
With -qnohpf, the compiler works like xlf.

You can add some of the following suboptions:

{ sequence | nosequence } Forces either SEQUENCE or NOSEQUENCE as default directive in every applicable scope. The default option is NOSEQUENCE.

{ purecomm | nopurecomm } The purecomm suboption indicates that pure procedures may require communication. The opposite option, nopurecomm, specifies that all pure procedures in the program do not require communication to access their data, which is already locally available.



New/Not Supported Compiler Options

- New flags:

  -qassert { deps | nodeps | itercnt=n }

  - Changes the compiler default assertion about the average-iteration loop.

  -qtbtable = { none | small | full }

  - Amount of debugging traceback information.

- Flags not supported:

  -qpdf

  - Flag used with xlf -qnohpf to tune a code using the profile-directed feedback.

  -qwait

  - Flag related to NetLS, which is not used for HPF.

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 HPF*lx1*

IBM XL HPF has the following new flags:

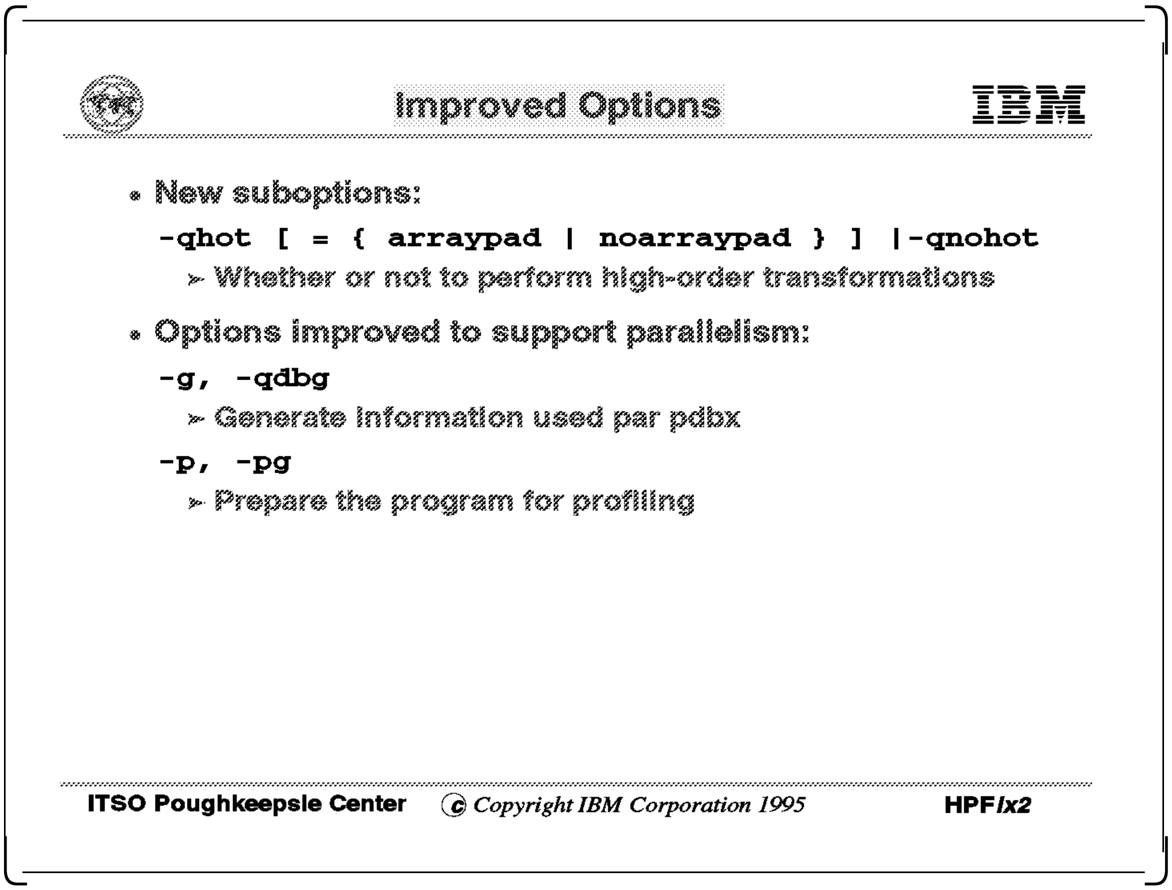
-qassert={deps|nodeps|itercnt=n}
This suboption indicates the kind of information you need from the compiler. The compiler is able to provide information about possible dependences. You can also set up itercnt with the value to be used to evaluate the way DO loops must be improved.

-qtbtable={none|small|full}
This flag sets up the amount of debugging traceback information in the object file.

The following XL Fortran flags are not supported by XL HPF:

-qwait
This flag is related to NetLS/iFORLS, which is not used by the XL HPF compiler.

-qpdf
This suboption, not supported with -qhpf, indicates to the compiler that it can use profiling information to try to optimize the program.



Improved Options

- **New suboptions:**

  -qhot [= { arraypad | noarraypad }] |-qnohot

  - Whether or not to perform high-order transformations
- **Options improved to support parallelism:**

  -g, -qdbg

  - Generate information used par pdbx

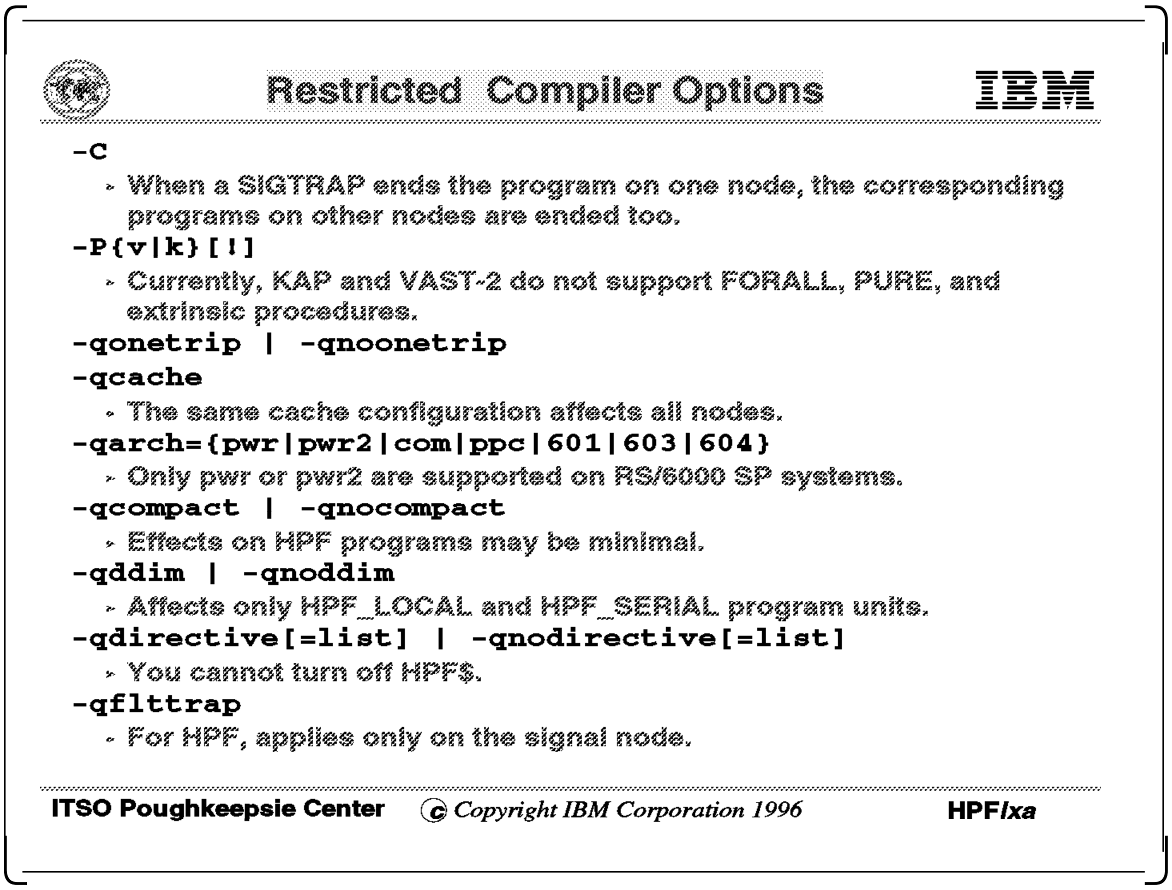
  -p, -pg

  - Prepare the program for profiling

ITSO Poughkeepsie Center © Copyright IBM Corporation 1995 HPF/x2

In XL HPF, several features are improved to effectively support parallel programs and provide information to the tools delivered with IBM Parallel Environment, such as the parallel dbx (pdbx or xpdbx), and the parallel profiling. The flag -qhot may be set up with the new suboptions { arraypad | noarraypad }, which indicates to the compiler whether you want to use this ultimate optimization.

As a matter of fact, It was observed that it is better not to have arrays with a dimension of 2^n. With arraypad, when the compiler finds such an array, it automatically increases its extent. See *IBM XL High Performance Fortran for AIX User's Guide* for more information.



Restricted Compiler Options

-C

- When a SIGTRAP ends the program on one node, the corresponding programs on other nodes are ended too.

-P{v|k}[!]

- Currently, KAP and VAST-2 do not support FORALL, PURE, and extrinsic procedures.

-qonetrip | -qnoonetrip

-qcache

- The same cache configuration affects all nodes.

-qarch={pwr|pwr2|com|ppc|601|603|604}

- Only pwr or pwr2 are supported on RS/6000 SP systems.

-qcompact | -qnocompact

- Effects on HPF programs may be minimal.

-qddim | -qnoddim

- Affects only HPF_LOCAL and HPF_SERIAL program units.

-qdirective[=list] | -qnodirective[=list]

- You cannot turn off HPF$.

-qflttrap

- For HPF, applies only on the signal node.

Some restrictions apply when you want to use these flags with -qhpf.

See *IBM XL High Performance Fortran for AIX User's Guide* for more information.

Appendix A. Special Notices

This publication is intended to help customer specialists and IBM personnel involved in the education process for scientific and technical computing environments offered by IBM on RS/6000 Scalable POWERparallel systems running AIX Version 4.1.3 and PSSP Version 2 Release 1.

The information in this publication is not intended as the specification of any programming interfaces that are provided by products described hereafter, such as IBM Parallel Environment Version 2, IBM PVMe Version 2, IBM Parallel ESSL for AIX Version 4, IBM Parallel OSL for AIX, and IBM XL HPF for AIX.

See the PUBLICATIONS section of the IBM Programming Announcement for these licensed program products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

You can reproduce a page in this document as a transparency, if that page has the copyright notice on it. The copyright notice must appear on each page being reproduced.

Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How To Get ITSO Redbooks" on page 165.

- *RS/6000 Scalable POWERparallel Systems: PSSP V2 Technical Presentation*, SG24-4542

B.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	**Subscription Number**	**Collection Kit Number**
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection (available soon)	SBOF-7250	SK2T-8042

B.3 Other Publications

These publications are also relevant as further information sources:

- IBM AIX Parallel Environment:
  - *IBM Parallel Environment for AIX: General Information*, GC23-3906
  - *IBM Parallel Environment for AIX: Installation, Administration, and Diagnosis*, GC23-3892
  - *IBM Parallel Environment for AIX: Operation and Use*, GC23-3891
  - *IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference*, GC23-3893
  - *IBM Parallel Environment for AIX: MPI Programming and Subroutine Reference*, GC23-3894
  - *IBM Parallel Environment for AIX: Hitchiker's Guide*, GC23-3895
- IBM PVMe for AIX:
  - *IBM PVMe for AIX: User's Guide and Reference*, GC23-3884
- IBM Parallel ESSL for AIX Version 4
  - *Parallel ESSL Guide and Reference*, GC23-3836
  - *ESSL Guide and Reference*, SC23-0526
- IBM Parallel OSL for AIX
  - *Parallel OSL User's Guide*, SC23-3824

- *OSL Guide and Reference*, SC23-0519

- IBM XL High Performance Fortran for AIX
  - *IBM XL High Performance Fortran User's Guide*, SC09-2228
  - *IBM XL High Performance Fortran Language Reference*, SC09-2226

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER —** to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

- Invoice to customer number ____________________

- Credit card number ____________________

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

AIX	advanced interactive executive (IBM′s flavor of UNIX)
API	application program interface
BLAS	basic linear algebra sub-programs
C	UNIX system-programming language
CD-ROM	(optically read) compact disk - read only memory
CDE	Common Desktop Environment (form X/Open)
CPU	central processing unit
CSS	communication subsystem (included in PSSP to support the high performance switch)
ESSL	Engineering and Scientific Subroutine Library
FORTRAN	formula translation (programming language): for FORTRAN 77 and before
Fortran	formula translation (programming language): Fortran 90 (case sensitive)
GUI	graphical user interface
HPF	high performance FORTRAN
HPS	high performance switch
I/O	input/output
IBM	International Business Machines Corporation
IP	internet protocol (ISO)
ITSO	International Technical Support Organization
LAN	local area network
LP	linear programming
LPEX	live parsing editor
LU	lower and upper triangulation
MIMD	multiple instruction stream, multiple data stream
MIP	mixed integer programming
MPI	Message Passing Interface
MPL	Message Passing Library
NIC	numerically intensive computing
OSL	optimization subroutine library (high-performance math programming routines)
PC	Personal Computer (IBM)
PE	parallel environment
PESSL	Parallel Engineering and Scientific Subroutine Library
POWER	performance optimization with enhanced RISC (architecture)
PSSP	AIX Parallel System Support Programs (IBM program product for SP1 and SP2)
PTF	program temporary fix
PVM	parallel virtual machine (developed by Oak Ridge National Laboratory, USA)
QP	quadratic programming
RISC	reduced instruction set computer/cycles
RM	resource manager
SP	IBM RS/6000 Scalable POWERparallel Systems (RS/6000 SP)
SPMD	simple program multiple data
TCP	transmission control protocol (USA, DoD)
TCP/IP	Transmission Control Protocol/Internet Protocol (USA, DoD, ARPANET)
TLB	translation lookaside buffer
UDP	user datagram protocol (TCPIP)
UNIX	an operating system developed at Bell Laboratories (trademark of UNIX System Laboratories, licensed exclusively by X/Open Company, Ltd.)
US	user space (optimized communication protocol for NIC parallel programs using the high performance switch on RS/6000 SP systems)
VF	vector facility
VT	visualization tool
X	X Window System (trademark of MIT)
X/MOTIF	Window System and Motif bindled toghether (IBM)

Index

Special Characters

-euilib 15
-exec (PVMe) 78
-ip (PVMe) 78
-qassert 157
-qdirective 134
-qhpf 127, 156, 159
-qnohpf 127, 156
-qpdf 157
-qtbtable 157
-qwait 157
-share (PVMe) 78
-trace (PVMe) 78
:MPI_Start 23
.PVMe Environment Variables 79
  PVMEPATH 79
  PVMHFN 79

A

abbreviations 169
acronyms 169
ALIGN 135, 137, 139
Amdahl′s law 2, 3, 4, 7

B

basic linear algebra communication subprograms (BLACS) 103
basic linear algebra communication subroutines (BLACS) 84
bibliography 163
BLACS 84, 85, 86, 103, 107, 109
BLACS examples 109
BLACS subroutines 105
  BLACS_EXIT 110
  BLACS_GET 105, 108
  BLACS_GRIDEXIT 110
  BLACS_GRIDINFO 108
  BLACS_GRIDINIT 105, 108
  BLACS_GRIDMAP 105
  BLACS_PINFO 105, 107
BLACS_EXIT 110
BLACS_GET 105, 108
BLACS_GRIDEXIT 110
BLACS_GRIDINFO 108
BLACS_GRIDINIT 105, 108
BLACS_GRIDMAP 105
BLACS_PINFO 105, 107
BLAS 91
BLOCK 142, 143, 144, 145
blocking buffered communication 19
blocking communication 16, 18
blocksize for ESSL 90

C

cartesian topology 17, 43
CDE 131
collective communication 17, 37
collective communication routine 64
collective nonblocking functions 47
command-line builder (xxlhpf) 131
common desktop environment (CDE) 131
communicator 17, 27
convolution 94
correlation 94
CYCLIC 127, 144

D

datatype 30
derived datatype 17, 31
DISTRIBUTE 135, 141

E

Eigensystem/Singular Value Routines 100
  PDGEBRD 100
  PDGEHRD 100
  PDSYEVX 100
  PDSYTRD 100
eigenvalues 94
eigenvectors 94
EKKBSLV 114
EKKGTMD 114, 119
EKKIGET 114
EKKISET 114
EKKMSLV 114, 119
EKKPTMD 114
engineering and scientific subroutine library (ESSL) 83
environmental management 45
ESSL/370 83
ESSL/6000 83
EXTRINSIC 136, 153

F

fast Fourier transform (FFT) 94
Fast Fourier Transform routines 101
FFT 94
FORALL 135, 146, 150, 151
FORALL construct 130

G
graph topology 17, 40

H
High Performance Fortran (HPF) 121
HPF 121, 123
HPF Compiler Options 1
HPF constructs 121
HPF directives 121, 123, 134, 135
  ALIGN 135
  DISTRIBUTE 135
  INDEPENDENT 135
  PROCESSORS 135
  SEQUENCE 135
  TEMPLATE 135
HPF extensions 124
HPF flags 127
  -qdirective 134
  -qhpf 127
  -qnohpf 127
HPF operating environment 126
HPF statements 121
HPF_LIBRARY 130
HPF_LOCAL 127, 130, 136, 144, 153
HPF_LOCAL_LIBRARY 130
HPF_SERIAL 130, 136, 153

I
IBM Parallel Environment 9, 11
ILEN 154
INDEPENDENT 135, 146, 147
INTENT 136, 153
intercommunicator 29
intracommunicator 29
intrinsic procedure 154
Inumcpu 119
IPESSL 102
Iwhichcpu 115, 119

L
LAPACK 91, 103
level 2 parallel BLAS routines 94
level 3 parallel BLAS routines 94
linear programming (LP) 111, 112
live parsing extensible (LPEX) editor 131
LoadLeveler 53
LPEX 131

M
MAXLOC 154
Message Passing Interface 9, 10
message passing library 10
MIMD 91
MINLOC 154
MIP 111, 112, 119
mixed integer programming (MIP) 111, 112
MP_EUILIB 15
MPCI 11, 15, 56
MPE 10
MPI 11, 16, 85, 86, 126
MPI_Bsend 19
MPI_Buffer_attach 19
MPI_Buffer_detach 19
MPI_Comm_compare 27
MPI_Comm_create 27
MPI_Comm_dup 27
MPI_Comm_free 27
MPI_Comm_group 25, 27
MPI_Comm_rank 27
MPI_Comm_split 29
MPI_Errhandler_create 45
MPI_Group_compare 26
MPI_Group_difference 26
MPI_Group_excl 26
MPI_Group_free 26
MPI_Group_incl 25
MPI_Group_intersection 26
MPI_Group_range_excl 26
MPI_Group_range_incl 26
MPI_Group_union 26
MPI_Ibsend 21
MPI_Irsend 21
MPI_Isend 21
MPI_Issend 21
MPI_Pack 33
MPI_Recv 18
MPI_Request_free 24
MPI_Rsend 18
MPI_Send 18
MPI_Send_init 23, 24
MPI_Sendrecv 18
MPI_Ssend 18
MPI_Startall 23
MPI_Test 20
MPI_Testall 21
MPI_Testany 21
MPI_Testsome 21
MPI_Type_commit 32
MPI_Type_contiguous 31
MPI_Type_extent 32
MPI_Type_free 32
MPI_Type_hindexed 31
MPI_Type_hvector 31
MPI_Type_indexed 31
MPI_Type_lb 32
MPI_Type_size 32
MPI_Type_struct 31
MPI_Type_ub 32
MPI_Type_vector 32
MPI_Unpack 33

MPI_Wait 20, 23
MPI_Waitall 20, 23
MPI_Waitany 20
MPI_Waitsome 20
MPL 86
MPMD 55
multiple program multiple data (MPMD) 55

N

nonblocking buffered 22
nonblocking communication 11, 16, 20
NOSEQUENCE 130
NUMBER_OF_PROCESSORS 142, 154
NUMROC 102

O

optimization subroutine library (OSL) 83
OSL 83
OSL migration to OSLp 116
OSL/6000 111
OSLp 111
OSLp - New Routines 114
  EKKBSLV 114
  EKKGTMD 114, 119
  EKKIGET 114
  EKKISET 114
  EKKMSLV 114, 119
  EKKPTMD 114
  Inumcpu 119
  Iwhichcpu 115, 119
OSLp Operating Environment 113

P

parallel environment 54
Parallel ESSL 83, 84, 85, 86, 92, 104, 105, 107, 126
Parallel ESSL general routines 102
  IPESSL 102
  NUMROC 102
  PDURNG 102
Parallel ESSL subroutines 95
  PDGEMM 97
  PDGEMV 95
  PDGER 95
  PDSYMM 97
  PDSYMV 95
  PDSYR 95
  PDSYR2 95
  PDSYR2K 98
  PDTRAN 98
  PDTRMM 97
  PDTRMV 95
  PDTRSM 98
  PDTRSV 96
Parallel Linear Algebraic Routines 99
  PDGETRF 99
  PDGETRS 99
Parallel Linear Algebraic Routines *(continued)*
  PDGOTRF 99
  PDGOTRS 99
parallel operating environment 9, 15
parallel optimization subroutine library (OSLp) 84, 111
Parallel OSL 111
Parallel OSL (OSLp) 83
parallel virtual machine (PVM) 49
partition manager 9
PBLAS 85, 90, 91, 94
pdbx 9, 131
PDCFT2 101
PDCFT3 101
PDCRFT2 101
PDCRFT3 101
PDGEBRD 100
PDGEHRD 100
PDGEMM 97
PDGEMV 95
PDGER 95
PDGETRF 99
PDGETRS 99
PDGOTRF 99
PDGOTRS 99
PDRCFT2 101
PDRCFT3 101
PDSYEVX 100
PDSYMM 97
PDSYMV 95
PDSYR 95
PDSYR2 95
PDSYR2K 98
PDSYTRD 100
PDTRAN 98
PDTRMM 97
PDTRMV 95
PDTRSM 98
PDTRSV 96
PDURNG 102
persistent communication 16, 23
PESSL Operating Environments 92
point-to-point communication 16
POWER 87
POWER2 87, 90
predefined datatype 30
process grid 89, 105
process group 25
process group management 17, 25
process identifier 25
PROCESSOR_SHAPE 154
PROCESSORS 135, 141, 142, 149
program builder 131
PURE 135, 153
PURE procedure 130
pvm_addhosts 68
pvm_bcast 72

pvm_catchout 68
pvm_gather 72
pvm_hostsync 68
pvm_initsend 70
pvm_mkbuf 70
pvm_precv 71
pvm_psend 71
pvm_reduce 72
pvm_scatter 72
pvm_trecv 71
pvmd3e 58
PVMe 15, 49
PVMe collective communication 72
PVMe console 74
PVMe daemon 58
PVMe daemon options 78
  -exec 78
  -ip 78
  -share 78
  -trace 78
PVMe examples 75
  supplying a hostlist file 76
  Using PVMe 75
  Using the IP protocol 77
PVMe group 72
PVMe host management 68
PVMe message exchange 71
PVMe message packing 70
PVMe message unpacking 70
PVMe nodelist generic request 81
PVMe nodelist specific request 81
PVMe session on mixed hostlist 80
PVMe signal exchange 71
PVMe task management 68
pvme_bcast 72

Q
quadratic problem (QMIP) 112

R
random number 94
reduction function 39
reduction operation 39
resource manager 53, 81
Running a PVMe Program 74

S
ScaLAPACK 85, 91, 103
SEQUENCE 130, 135, 148, 149, 156
shape of the process grid 89
single program multiple data 109
single program multiple data (SPMD) 55
SPMD 55, 85, 109
Subset HPF 124

T
TEMPLATE 135, 140
TLB 87
topology 40
translation lookaside buffer (TLB) 87

U
user controlled buffering 33
user defined reduction operation 39
user space protocol (US) 62, 64

V
virtual machine (PVM) 57
visualization tool 9

X
xldb 131
xpdbx 9
xpvm 49, 65, 78
xxlhpf 131

Printed in U.S.A.

SG24-4541-01