

Parallel Language Processing System for High-Performance Computing

●Eiji Yamanaka ●Tatsuya Shindo

(Manuscript received April 22, 1997)

Fujitsu has developed a common parallel language processing system for the VPP and AP distributed memory parallel computers. The parallel language processing system includes a parallelizing compiler, libraries, and parallelizing support tools.

The systems were developed to satisfy the following requirements :

- 1) To provide a language processing system that is common to the VPP and AP,
- 2) to provide multi-paradigms for parallel programming, and
- 3) to realize functions to achieve high performance.

The following have also been developed :

A Fortran parallelizing compiler that processes VPP Fortran. The compiler provides parallelizing notations for manual tuning and makes programming easy. The MPI and PVM message passing libraries, which can be custom tuned to suit a machine's architecture.

The SSL II library of popular numerical calculation algorithms, which are parallelized to achieve high performance.

A GUI programming support tool called Workbench that provides users with several options for parallel programming.

1. Introduction

The performance of single, central processing units has reached the limit of improvement, and it is expected that the same is true for shared memory parallel processing computers. Therefore, distributed memory parallel computers in which multiple processor elements (PEs: a PE is an operation unit consisting of a CPU and memory) are connected through a high-speed network are being investigated.

Therefore, Fujitsu has developed the VPP distributed-memory vector parallel supercomputer and the AP scalar parallel server.

To make the best use of a distributed memory parallel computer, it is necessary to distribute the load among PEs evenly and reduce the overheads for communication and synchronization between PEs. Therefore, it is important to develop programming techniques that can be used to meet these requirements.

The parallel language processing system sup-

ports parallel programming techniques with features that have not been implemented in conventional programming.^{1,2)}

2. Purposes of the Parallel Language Processing System

The system was developed to satisfy the following requirements:

- 1) To provide a language environment that is common to the VPP and AP systems,
- 2) to support multiple parallel processing methods, and
- 3) to achieve a higher processing rate.

2.1 Language environment common to the VPP and AP systems

The VPP system is a vector parallel system in which vector computers are used as nodes. The AP system is a scalar parallel system in which scalar computers are used as nodes. The VPP and AP systems have their own optimal type of pro-

gram, therefore, users should choose one of these systems, or link them, according to the programs to be executed.

The most important objective was to enable programs to be ported between the two systems easily.

For the parallel language processing system, the parallel programming language, message passing library, and numerical calculation library were designed to be used on both the VPP and AP systems. Also, user views in Workbench (the programming environment) are unified to satisfy the objective.

2.2 Support of multiple parallel processing methods

There are two major methods of creating a parallel program; one method uses parallel programming languages and the other uses message passing.

Parallel programming languages accept conventional programs created by sequential processing languages and are easy to write; however, they also restrict parallel processing models due to their syntax and they make other models hard to write.

Message passing can describe various parallel processing methods because communication between processors (i.e., low-level implementation) is directly described by message passing. However, message passing also makes programming difficult.

Each method has its own advantages and disadvantages. Therefore, the VPP and AP systems support both methods so that one or the other can be applied according to the purpose.

For message passing, a number of specifications have been suggested and standardized. MPI and PVM, which are in wide use, are supported in both the VPP and AP systems as platforms. The VPP system supports PARMACS, which is popular in Europe, and the AP system supports APLib, which was developed for the AP1000. Users, therefore, have a wide selection.

2.3 In search of higher processing rates

The primary goal of high-performance computing (HPC) is higher processing rates. The parallel language processing system focuses on providing specifications that speed up execution of a parallel processing program and developing installation forms that exploit the hardware capabilities.

The parallel processing compiler is a specially developed parallel programming language called VPP Fortran. VPP Fortran supports split allocation of arrays to multiple processors, highly abstract parallel descriptions such as parallel execution of DO loops, and a notation that enables flexible control of system operations, including block data transfer and synchronization.

To implement message passing, a lower layer called MPLib was placed in the VPP and AP systems and all message passing libraries are located on MPLib. The interface of MPLib is designed so that overheads due to the layer are minimized. Programs inside MPLib are tuned so that MPLib is optimized for each machine.

The components of the parallel language processing system are described in later sections.

3. Parallel Programming Language

The VPP and AP systems support the VPP Fortran parallel programming language. This section looks at VPP Fortran.

3.1 Purposes of VPP Fortran

VPP Fortran was designed originally for the VPP500 system. When VPP Fortran was designed, the situation regarding languages for distributed memory parallel computers was as follows:

- There were no practical standard parallel languages for distributed memory parallel computers.
- Techniques for implementing automatic parallelization in distributed memory parallel computers were not fully developed, and it was expected that sufficient performance could not be achieved by automatic paral-

lelization using only compilers.

- Message passing differed greatly from message passing in conventional sequential processing programming, thus programmers who created sequential processing programs could not easily handle message passing.

Because of the above, VPP Fortran was developed so that general programmers could write parallel processing programs for distributed memory parallel computers.

3.1.1 Easy programming

Generally, programs for parallel computers with distributed memory are hard to create, and only users very familiar with parallel processing can use such computers. Parallel computer systems for practical use must have sufficient performance and easy programming features for general programmers.

3.1.2 Application of existing resources

Parallel computer systems for practical use must be able to handle existing user programs.

3.1.3 Performance on the hardware

The performance of programs running on distributed memory parallel computers tends to be much lower than the peak performance offered by the hardware. To avoid this waste, new parallel processing languages must fully exploit the hardware's performance.

3.2 Overview of VPP Fortran

VPP Fortran is a kind of Fortran 90 with parallel processing functions for distributed memory parallel computers.

3.2.1 Logical configuration

Figure 1 shows the logical configuration of VPP Fortran. A parallel computer with a layered memory has a global space shared by PEs and local spaces specific to each PE.

The global space is a shared virtual memory space consisting of the operating system, execution libraries, and memory in each PE.

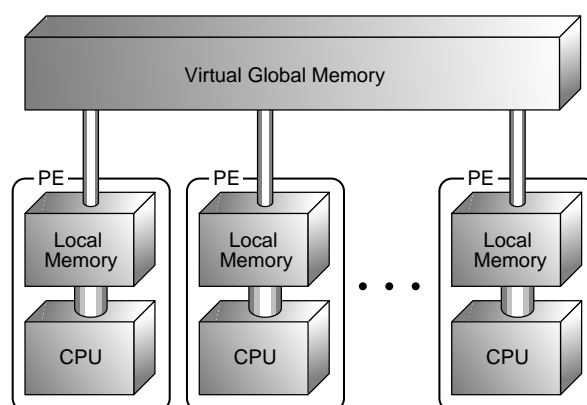


Fig. 1— Logical configuration of the system.

3.2.2 Features of VPP Fortran

1) Global array

One of the problems in creating programs for distributed memory parallel computers is how to locate a logical unit of data stored in physically separated memory areas.

To solve this problem, VPP Fortran introduces the global array using the global space provided by the system. In practice, the global array is split and assigned by a PE.

Use of the global array enables a unit of data to be handled logically. Therefore, there is generally no need to change a conventional programming style. Distributed memory parallel computers supporting the global array simplify porting of programming styles and existing programs as compared with conventional distributed memory parallel computers.

2) Directive method

Parallel processing functions in VPP Fortran are supported by directives and service subroutines. Most parallel processing functions are described by directives only. By regarding directives as comments, programs with parallel processing functions described by directives are guaranteed to run in the same manner even if they are sequentially translated and sequentially executed. This provides compatibility with programs in other systems, and is useful for verifying parallel processing programs.

3) Compatibility with subprograms in standard Fortran

Subprograms in standard Fortran can be applied to VPP Fortran without any modification to parallel processing. VPP Fortran enables effective use of existing program libraries.

4) Gradual parallel processing

To obtain high performance on distributed memory parallel computers it is always necessary to tune programs. For VPP Fortran, parallel processing functions are equipped so that programs can be tuned gradually. The users can gradually tune programs to achieve the maximum parallel performance. Thus, the performance that users can obtain depends on the tuning.

5) Explicitly splitting function

Data transfer between PEs generally degrades the performance of parallel processing programs that run on distributed memory parallel computers. If an area for a calculation assigned to a PE does not correspond to the data area on the PE, data must be transferred from another PE to continue the calculation. Therefore, VPP Fortran has specifications that ensure consistent data partitioning and procedure partitioning. The specifications minimize the data transfer frequency to prevent a reduction in the performance of parallel processing programs.

6) Explicit transfer

Although VPP Fortran is designed so that the data transfer frequency is reduced, some programs have structures that require data transfer between PEs. To maintain high performance, VPP Fortran has specifications that enable users to control patterns and data transfer timing. The specifications guarantee that programs can transfer data efficiently.

Table 1 shows a list of VPP Fortran parallel processing functions.

3.2.3 Execution method of VPP Fortran

A spread-barrier method is adopted for VPP Fortran. The spread-barrier method basically consists of two cyclic operations: spread and barrier.

Spread is a split execution in each PE. Barrier is a simultaneous synchronization of all PEs.

Figure 2 shows an example of how a program is executed. The program is handled by eight PEs. parallel region indicates the beginning of the part to be parallel processed, and end parallel indicates the end of the part to be parallel processed. The beginning and end of the part to be parallel processed are indicated once in a program. spread region indicates that parts C and D are allocated to different PEs and parallel processed. spread do indicates that iterations of the DO loop are divided and parallel processed. end spread indicates the end of the split execution. Barrier synchronizes spread region/end spread and spread do/end spread. Programs are executed with cycles of spread and barrier.

Separating split execution from parallel processing allows overheads due to parallel processing to be reduced.

3.3 Performance and applicability of VPP Fortran

3.3.1 Performance of VPP Fortran

Figure 3 shows the ratio of the estimated peak performance to the actual performance of the VPP500 hardware using the LINPACK program. The data in the figure can be found in reference ³⁾. The LINPACK program written with VPP Fortran uses the block LU decomposition that is suitable for parallel processing. LINPACK is within the scope of the VPP Fortran specifications; there are no specific procedures such as program coding using an assembler. Figure 3 shows that a VPP Fortran program can be very efficient.

3.3.2 Applicability of VPP Fortran

Table 2 lists the number of parallel directives added when application programs were improved for parallel processing using VPP Fortran in the field. VPP's applicability can be measured by the number of parallel directives required to change a program for parallel processing. (Other important factors are the ease of rewriting and the ability to represent the characteristics of parallel pro-

Table 1. List of parallel processing functions in VPP Fortran

	Syntax/Function	Description
Directive	processor, proc alias subprocessor	Declares processor group forms or aliases used in the program.
	index partition	Declares the partition type or scope.
	local	Declares the local variables.
	global	Declares the global variables.
	parallel region / end parallel	Specifies the scope of the program to be parallel processed.
	spread region / end spread	Specifies split execution for a part of the program.
	spread do / end spread	Specifies split execution for the DO loop.
	spread move / end spread	Specifies data transfer between PEs.
	overlapfix	Specifies transfer of the boundaries.
	movewait	Specifies to wait for the end of asynchronous data transmission.
	broadcast	Specifies broadcast data transmission.
	unify	Specifies data transfer between reduplicated local arrays.
	barrier	Specifies barrier synchronization.
	lockon / endlock	Specifies the critical section.
Service procedure	novproc, norproc	Inquires about the number of PEs used for execution.
	idvproc, idrproc	Inquires about the identification numbers of PEs used for execution.
	postevt, waitvt	Post / wait type synchronization.
Promotion of Fortran acceptance	common	Common connection between global variables.
	equivalence	Memory sharing for local variables and global variables.
	Procedure interface	Virtual / actual connection between arguments for global variables and split arrays.
	Input / Output	Introduces parallel I / O.

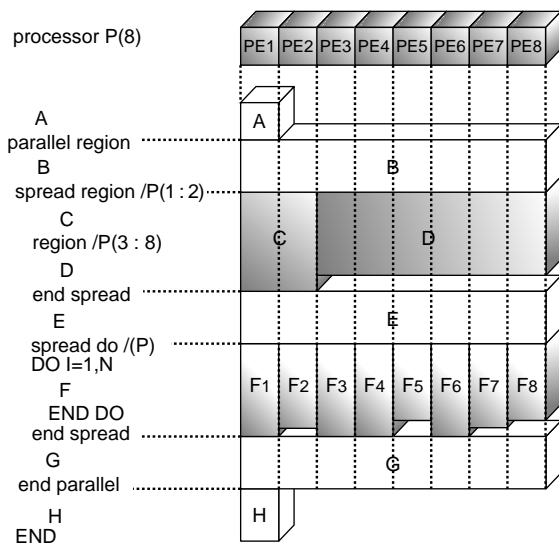


Fig. 2— Spread-barrier execution.

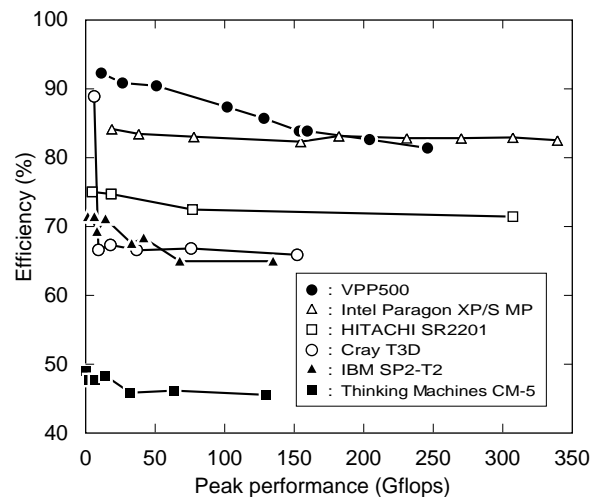


Fig. 3— Efficiency of LINPACK program.

Table 2. Parallelization of application program

Field	Total number of steps	Number of directives	Content ratio
Computational hydrodynamics	13 765	83	0.6%
	4 019	303	7.5%
	5 711	777	13.6%
	3 645	356	9.8%
	5 302	760	14.3%
Meteorology	11 779	381	3.2%
Petroleum	904	48	5.3%
Particle simulation	479	14	2.9%
	6 623	66	1.0%
Quantum chromodynamics	1 209	94	7.8%

cessing.)

For VPP Fortran, the percentage of parallel directives is 10% or less in most programs in each field. This fact indicates the wide applicability of VPP Fortran.

4. Message Passing

Message passing techniques are based on interprocess communication. The message passing techniques were developed in the early 1980s, mainly in the USA. Currently, MPI, PVM, and PARMACS are the most popular.

4.1 Approach to message passing

Fujitsu started using message passing in the HPC field in 1993. Table 3 shows the message passing libraries for the VPP and AP systems currently provided by Fujitsu.

4.2 Message passing programming

Parallel processing programs using message passing are created by adding library calls for communication between PEs to sequential processing programs. Functions to create processes, transfer and receive messages, and exclusively control synchronization are essential for message passing.

Figure 4 shows a message passing program with a simple MPI used as an example.

The features of message passing parallel programs can be clearly identified by comparing mes-

Table 3. Major message passing libraries

Type	Developer	Features
MPI	MPI Forum	Supports group communication functions and communication space.
PVM	Oak Ridge National Research (USA)	Supports hereto-environments using daemons.
PARMACS	PALLAS Inc.(FRG)	Supports process topologies.

Table 4. Features of message passing

Function	Message passing	Data paralleled (VPP Fortran)
Data space	Local space only	Global variables and local space are used in parallel.
Data allocation	No split allocation is supported.	Split allocation is supported.
Data transfer	Flexible	Frequently used data is put in stationary patterns.
Direct quotation from another PE's data	Impossible (Explicit transfer is required.)	Possible (Transfer is automatic.)
Parallel processing method	Flexible	Mainly handles DO loops.

```

main (argc, argv)
{
  MPI_Init (&argc, &argv);      /*Initialization      */
  if (rank == A) {
    strcpy (msg, "How are you?"); /*Posts the message "How are you?" */
    MPI_Send (msg, ..., B, ...); /*from A to B.      */
  }
  eles {
    MPI_Recv (msg, ..., A, ...); /*B receives the message */
  }
  /*from A.      */
  MPI_Finalize ();             /*End      */
  return 0;
}

```

Fig. 4— MPI program.

sage passing parallel programs with data-parallelized programs such as VPP Fortran programs. Table 4 gives a basic comparison of the features.

4.3 Features for installing the message passing function

The VPP and AP systems do not install the message passing function directly on the hardware or operating system. Instead, they install the message passing function on a core library called MPLib, which is a common layer.

MPLib is fully capable of handling the high-speed networks in the VPP and AP systems to

make the best use of the hardware.

This mounting method enables the above message passing function to be standardized for the VPP and AP systems.

Also, by setting MPlib as a common layer and using execution information obtained by MPlib, all programs can be executed using a common operation method regardless of the types of the above message passing function.

4.4 Performance of the message passing function

The VPP and AP systems are parallel computers with distributed memory. Their performance depends on their network communication capabilities.

The following two values were measured as indexes of data transmission capability using a program (PingPong program) for transferring messages between two processes:

- Latency: Data transmission time for data with length 0 (Time required to start data transfer)
- Bandwidth: Data transmission rate for a specified amount of data

Table 5 shows the performance of the major message passing libraries, including those in other vendors' distributed memory parallel computers. The rates in "Bandwidth" indicate that the VPP and AP systems make the best use of their high-speed networks.

Figure 5 shows the performance efficiency for application software using PARMACS on the VPP500.

In general, the greater the transmission overhead due to process addition, the less the efficiency of parallel processing. However, in the VPP system, the rate at which the efficiency is decreased is low. This effect is due to the cross-bar network, which enables fast and highly efficient operations in the VPP system.

Table 5. Performance of message passing

System	Type	Latency (μ s)	Bandwidth (megabytes/second)
VPP system	MPI	26	429
	PVM	15	323
	PARMACS ^{Note)}	17	470
AP system	MPI	68	60
T3D (CRAY)	PARMACS ^{Note)}	60	25
SP2 (IBM)	PARMACS ^{Note)}	56	23
CM5E (Thinking Machines)	PARMACS ^{Note)}	18	25

Note) The values for PARMACS were measured by PALLAS Inc.(FRG).

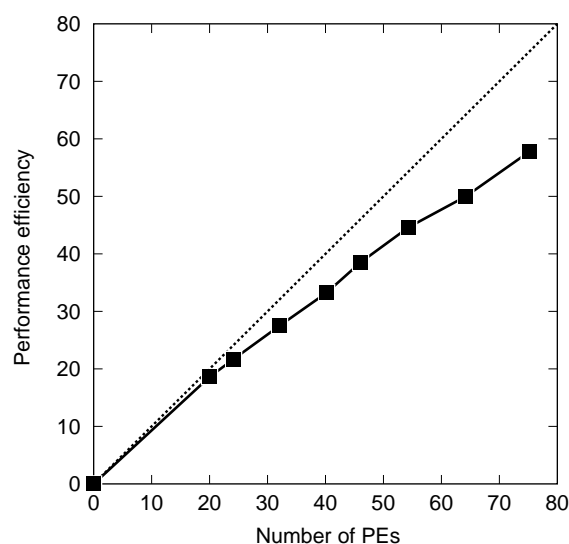


Fig. 5— Performance efficiency.

5. Technological Computation Libraries for Parallel Processing

Fujitsu has developed parallel numeric computation libraries SSL II/VPP and SSL II/AP for the VPP and AP systems.

5.1 Overview of SSL II/VPP and SSL II/AP

SSL II/VPP and SSL II/AP are new, parallel numeric computation libraries that are highly tuned to bring out the hardware capabilities of the VPP and AP distributed memory parallel computers. SSL II/VPP and SSL II/AP feature high capabilities and scalability.

The parallel numeric computation algorithms are provided in the form of easy-to-use VPP For-

tran subroutines.

Data for calculations is stored in global arrays that are separated and located in PEs. To store sparse matrixes, the Ellpack format and diagonal format are supported.

These parallel numeric computation algorithmic procedures were developed jointly by Fujitsu and a group studying numeric computation at the Australian National University. Members in the group have been leading users of parallel computers, and they are an authority on numeric computation.

SSL II/VPP and SSL II/AP use the same parallel numeric computation algorithm and interface to maintain compatibility between programs on the VPP and AP systems.

5.2 Functions in SSL II/VPP and SSL II/AP

SSL II/VPP and SSL II/AP provide functions that are used frequently or applied to sizable calculations. The following functions are provided:

- Linear equation solver for dense matrices (real matrix, positive definite symmetric matrix, complex matrix)
- Linear equation solver for banded matrices (real matrix, positive definite symmetric matrix)
- Linear equation solver for sparse matrices (real matrix, positive definite symmetric matrix)
- Matrix multiplication of real matrices, real sparse matrix vector multiplication
- Inverse of real matrices
- Fourier transforms (uni-dimensional to three-dimensional complex transforms, uni-dimensional to three-dimensional real transforms)
- Eigenvalue problem (real symmetrical matrix, tri-diagonal matrix, generalized eigenvalue problem)
- Singular value decomposition
- Least square solution
- Uniform random numbers

5.3 Overview of the algorithm

We developed the latest numeric computation algorithm to implement the functions listed above on distributed memory parallel computers. For large-scale calculations, the algorithm provides high-speed processing almost proportional to the number of PEs.

The algorithm has the following features:

- For the linear equation solver, a blocked direct computational method is used. In this method, the optimal load balance is maintained by dynamically redistributing the data among PEs, and data transfer and computation are overlapped.
- For the sparse matrix solver, the preconditioned conjugate gradient method (CG method) and the solid, modified generalized conjugate residuals method (MGCR method) are used.
- For Fourier transform kernels, a recursive 5-step algorithm suitable for vector computers is used.
- Double-precision uniform random numbers have a long period of 10^{52} or more, and an algorithm with good statistical characteristics is used.
- For eigenvalue problems and singular value problems, the one-sided Jacobi method is used due to its good scalability.

5.4 Performance of SSL II/VPP

This part describes some examples of the performance of SSL II/VPP on the VPP700 (1 PE: 2.2 Gflops).

- SSL II/VPP can solve a real coefficient linear equation with 50,000 elements in 1,746 seconds using 27 PEs. (A restriction on memory prevents operation of just a single PE; however, if a single PE could be used to solve the equation, it would take about 10 hours.) With 27 PEs, the performance is 47.72 Gflops, which is about 80% of the peak hardware performance.

Figure 6 shows the relationship between the

number of PEs and the performance.

- SSL II/VPP can compute a matrix multiplication with 10,000 elements in about 93 seconds using 10 PEs, yielding a performance of about 98% of the peak hardware performance.

Figure 7 shows the relationship between the number of PEs and the performance.

- SSL II/VPP can process a uni-dimensional/three-dimensional large-sized complex FFT at a high rate.

Problems that are too large to store in a PE can also be handled. For example, using 25 PEs, a uni-dimensional FFT with 2^{28} elements and a

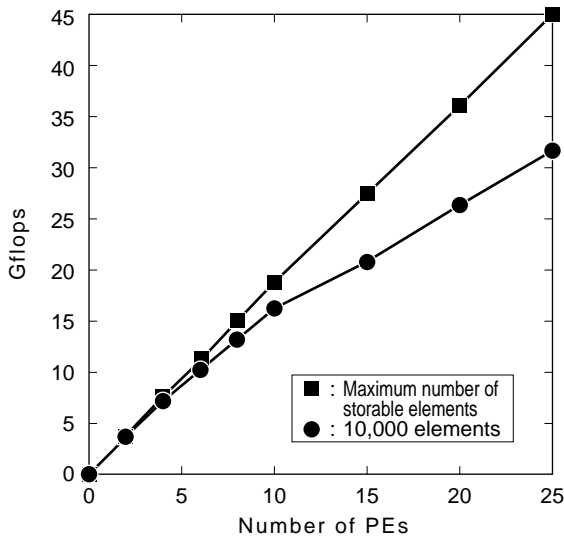


Fig. 6— Linear equation solver.

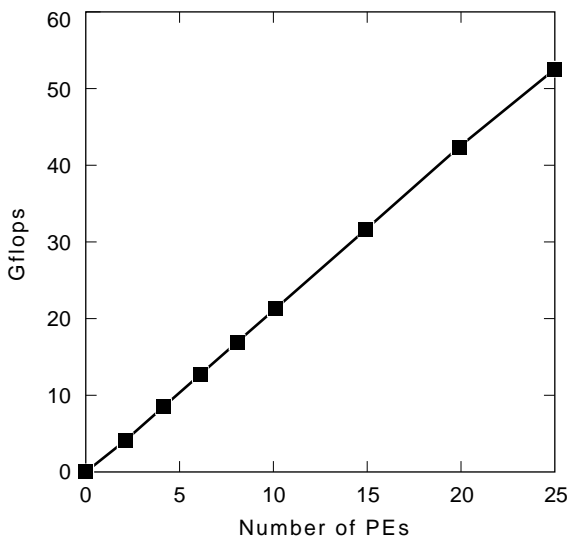


Fig. 7— Matrix multiplication(10,000 elements).

three-dimensional FFT with $1,024 \times 1,024 \times 1,024$ elements can be calculated in 1,536 seconds (24.46 Gflops, 44.4% of peak hardware performance) and 10.35 seconds (15.56 Gflops, 28.2% of peak hardware performance), respectively. These performance figures are some of the highest in the field of large-scale FFTs.

Figure 8 shows the relationship between the number of PEs and the performance.

- SSL II/VPP can generate 250 M double-precision uniform random numbers per second per PE.

Figure 9 shows the relationship between the number of PEs and the performance.

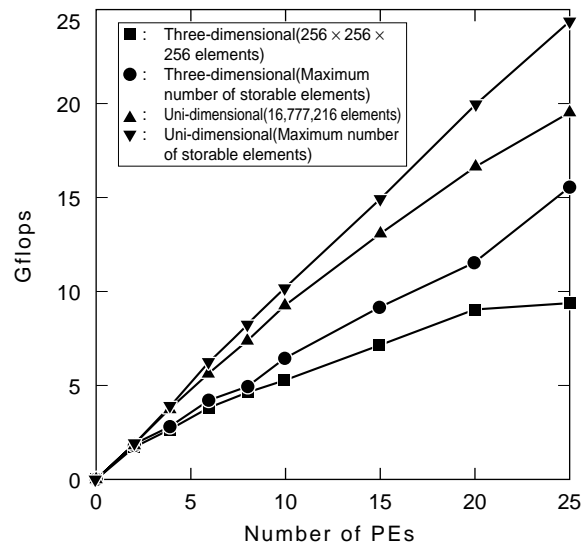


Fig. 8— Complex Fourier transform.

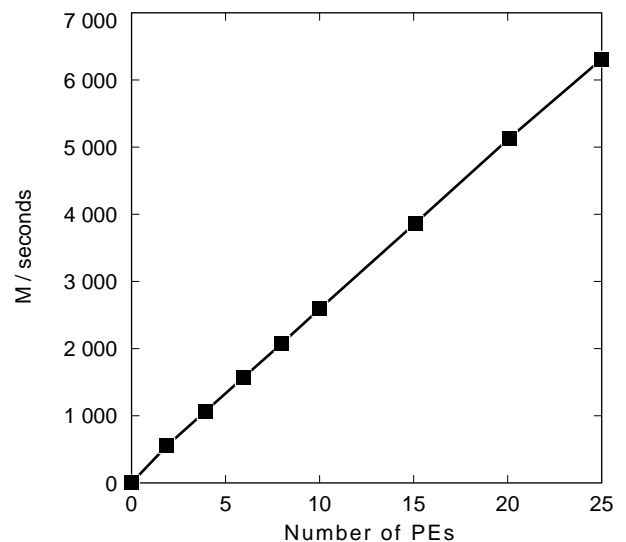


Fig. 9— Uniform random number.

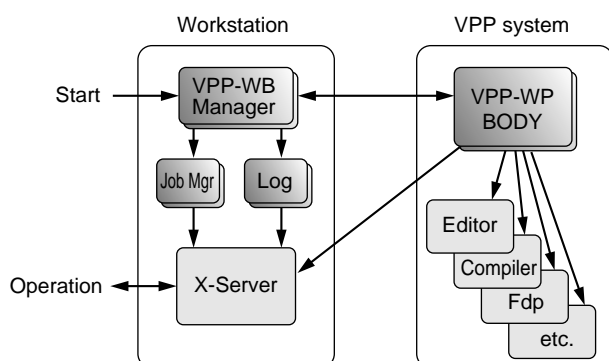


Fig. 11— Working environment.

7. System-specific functions

In addition to a parallel processing system that is common to the VPP and AP systems, system-specific functions are provided to exploit various features and improve performance.

7.1 Functions specific to the VPP system

7.1.1 Automatic vectorization/LIW optimization functions

The VPP system has a vector parallel architecture. In this architecture, each PE corresponds to a conventional supercomputer. Thus, conventional programs can be used without performance deterioration even if the programs are not modified for parallel processing.

If the programs run satisfactorily on the VPP system, there is no need to modify the program for parallel processing. To obtain a higher performance, the programs must be modified for parallel processing.

For more effective use of the PE's vector architecture, the VPP system supports automatic vectorization and optimization functions that have proved reliable on the VP series. The main functions of automatic vectorization and optimization are as follows:

- Vectorization of nested DO loops
- Vectorization of DO loops containing control statements such as IF statements
- Vectorization of DO loops containing intrinsic functions

- Partial vectorization of DO loops
- Vectorization of total sums, inner product operations, maximum value/minimum value retrievals, and collection/diffusion operations
- Vector pipeline schedule
- Optimization for LIW

7.1.2 Analyzer

When creating new programs or porting programs developed for other systems, performance tuning is always necessary. The VPP system supports an analyzer sampler and PEPA/MPA for performance tuning.

1) Sampler

The sampler is a sampling performance analysis tool. In sampling, an executed program is checked at various points by periodic CPU interrupts to analyze the performance.

Sampling does not require retranslation of programs and enables performance analysis using programs in the standard executable format.

In vector program analyses, the distribution of cost (percentage of CPU interrupts for each procedure, loop, and array; and average physical vector length) is displayed by a routine and loop.

Using the information displayed, high-cost parts can be identified and programs can be effectively tuned.

For parallel processing programs, the parallel processing, parallel processing rate, parallel processing acceleration rate, load balance between PEs waiting for synchronization, and asynchronous transfer wait rate are displayed for the entire program or a specific procedure.

By using the above information, program performance can be improved by increasing the parallel processing rate and reducing the wait-time for synchronization and asynchronous transfer.

2) PEPA/MPA

PEPA/MPA checks how the hardware is used while the sampler analyzes the performance checking software's characteristics (procedures and loops).

The PEPA (PE performance analyzer) collects events related to PEs. It collects the VU busy rate

(fraction of time the vector pipeline is operating), All-PE average performance (ratio of operations using a floating point for scalar instructions to operations using a floating point for vector instructions), number of operations (total number of operations using a floating point for scalar instructions or vector instructions), measurement time (time needed to collect data).

MPA (mover performance analyzer) collects events related to the DTU (data transfer unit hardware), which handles data transfers between PEs.

The PEPA and MPA are provided as subroutines. The PEPA/MPA can be used by adding calls for these subroutines to the beginning and end of parts to be analyzed. It is also possible to analyze the entire program by specifying environment variables without modifying the source program.

7.2 Functions specific to the AP system

7.2.1 APLib (AP1000 compatible library)

The AP3000 supports APLib, which was originally designed for the AP1000 communication library.

APlib inherits software resources developed on the AP1000; it permits multi-thread programming methods to be used on each node.

Tasks created using APLib are mapped onto threads on the AP3000. A node in the SMP configuration permits the tasks to be processed in different CPUs.

7.2.2 Parallel profiler

The AP system is provided with a parallel profiler and performance analysis tool for analyzing the performance of programs created in VPP Fortran.

After execution of a parallel processing program, the parallel profiler indicates the total time and calculation, communication, and synchronization times for each subroutine, functional procedure, or DO loop.

The performance analysis tool accumulates

information about all events. Therefore, to avoid a data space shortage, the performance analysis tool should be used only for a specific portion of a parallel processing program.

When tuning a parallel processing program, it is recommended to first apply the parallel profiler to the entire program to locate the portions that take much time to process. Then, the performance analysis tool can be used for individual sections to check the details.

8. Conclusion

This paper outlined the parallel programming language, message passing libraries, numeric computation libraries, and programming tools of the parallel language processing system developed for the VPP and AP systems. These parallel language processing system tools enable general users to use distributed memory parallel computers for practical purposes.

Current indications are that there is a limit to the performance of shared memory parallel computers. Therefore, distributed memory parallel computers will become more important in HPC.

In response, we intend to enhance the functions, implement an automatic parallelization function for distributed memory parallel computers (which still remains to be achieved), and promote standard languages for parallel computers.

References

- 1) M. Nakanishi, H. Ina, and K. Miura: A High Performance Linear Equation Solver on the VPP500 Parallel Supercomputer. Proceedings of Supercomputing'94, pp.803-810 (1994).
- 2) S. Kamiya, P. Lagier, W. Krotz-Vogel, and N. Asai: Two Programming Paradigms on the VPP System. Proceedings of the International Symposium on PDSC'95, pp.35-44 (1995).
- 3) J.J.Dongarra: Performance of Various Computers Using Standard Linear Equations Software. August 12, 1996.



Eiji Yamanaka received the Master degree in Control Engineering from the Tokyo Institute of Technology, Tokyo, Japan in 1988. He joined Fujitsu Ltd., Numazu, Japan in 1988, where he is currently engaged in research and development of parallel compilers for Fujitsu's VP and VPP computer series.

E-mail : yamanaka@lp.nm.fujitsu.co.jp



Tatsuya Shindo received the B.S. degree in Electrical Engineering from Waseda University, Japan in 1983. He joined Fujitsu Laboratories Ltd. in 1983, where he was engaged in research of parallel processing. From 1990 until 1992 he was a visiting researcher at Stanford University on sabbatical leave. He is currently a manager of parallel software development at Fujitsu Ltd.