H/V-버스 병렬컴퓨터의 설계 및 성능 분석

Design and Performance Analysis of the H/V-bus Parallel Computer

김 종 현*
Jong Hyun Kim

Abstract

The architecture of a MIMD-type parallel computer system is specified; a simulator is developed to support design and evaluation of systems based on the architecture; and experiments are conducted with the simulator to evaluate system performance. The horizontal/vertical-bus(H/V-bus) system architecture provides an NxN array of processing elements which communicate with each other through a network of N horizontal buses and N vertical buses. The simulator, written in SLAM II and FORTRAN, is designed to provide high-resolution in simulating the IPC mechanism. Parameters provide the user with independent control of system size, PE speed and IPC mechanism speed. Results generated by the simulator include execution times, PE utilizations, queue lengths, and other data. The simulator is used to study system performance when a partial differential equation is solved by a parallel Gauss-Seidel method. For comparisons, the benchmark is also executed on a single-bus system simulator that is derived from the H/V-bus system simulator. The benchmark is also solved on a single PE to obtain data for computing speedups. An extensive analysis of results is presented.

1. Introduction

In developing parallel computer systems, the implementation of a system with a large number of processing elements is expensive and time consuming. It is difficult to evaluate the performance of such a system in detail until implementation is complete. To avoid such difficulties, simulation is used as a standard tool for evaluating a design prior to its actual implementation[1,4]. Computer systems are usually modeled as discrete systems, characterized by discrete states and by state transitions(events) occuring at discrete time instants. To serve as a tool for investigating the behavior of a parallel computer system under a variety of conditions, the simulator model has to be responsive to all changes in

* 연세대학교 원주캠퍼스 전산학과

configuration, not only in terms of the number of processing elements but also in terms of the structure of the interprocessor communication(IPC) mechanism. Moreover, the model has to serve as a test vehicle for new or modified hardware structures, such as those for data transfers between processing elements and for processor synchronization. Basic decisions to be made in formulating the simulation model of a parallel computer system are the degree of detail, language and workload model[4,10,11,17].

Simulations vary in detail of implementation from those that are relatively gross to those that represent operations at the micro-instruction or gate level. The choice of the degree of detail in a simulation model is based largely on the objectives of the study. A more detailed simulation is likely to be more accurate and to have a broader field of application than a less detailed one, but is more expensive.

Katz[13], who was the first to use simulation in the design of a parallel processor system by building a macro-level simulation model to evaluate the effect of selected hardware parameters, software parameters and environmental parameters on the performance of a multiprocessor operating system, suggested that a macro-level simulation model (i.e., high degree of abstraction of the real system) was reasonable for the evaluation of total system performance over any length of time, particularly when using a computer much slower than the simulated system. Navaux[18] developed a Processor Interconnection Simulator to determine the best interconnection for the processor-memory communication of the SMM multiprocessor system. The simulation was at the system level (i.e., it took into account processors, memories, switches and peripherals, communication and operation aspects). A microscopic-level simulation was performed to compare the performance of the IBM 370/168 and the AMDAHL 470/V6 computers[1].

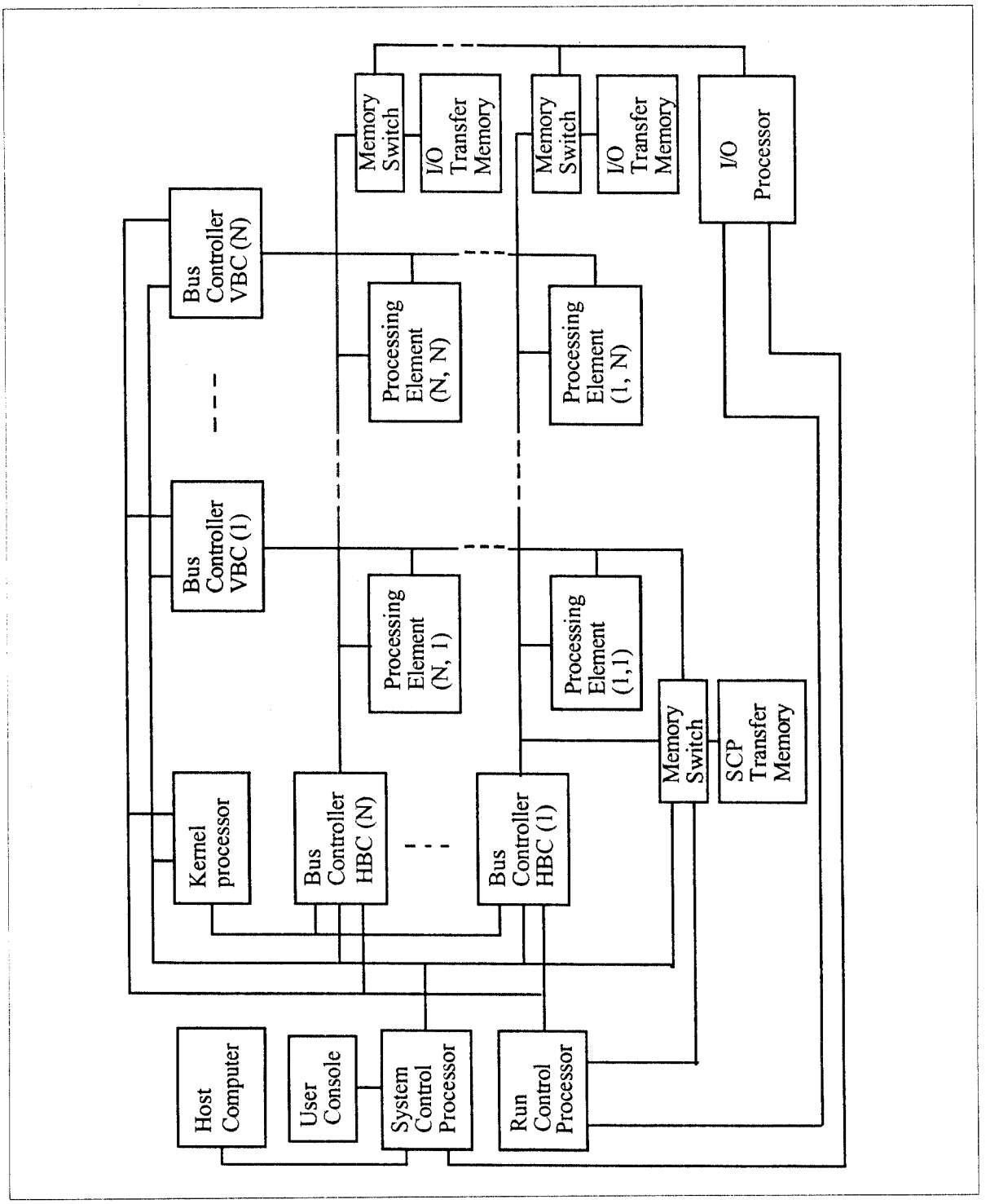
The simulation model has a general-purpose design allowing both the differences between two machines to be parameterized and the system configuration to be easily modified. Kumar and So[16] performed trace-driven simulation for studying the MIMD parallel computer with a multi-stage shuffle-exchange network. The simulation focused on investigating the memory access patterns of applications. Trace was obtained by executing an application program on a single processor system. The sequence of memory references and the position of barriers were identified and recorded on the trace.

A detailed simulation study has been carried out to evaluate the bandwidth of interleaved memories for high-performance vector processors[22]. Analysis like this study can not be performed without using simulation because a wide range of memory access patterns must be considered. Performance of hierarchical ring-based shared memory multiprocessors has been investigated by using simulation[11]. The simulator developed for this study was written using the smpl simulation library and reflected the behavior of the packets on a cycle-by-cycle basis.

In this study we design the architecture of the H/V-bus parallel computer system, develop the software simulator of the designed architecture, and measure its performance by using the simulator. The simulation model includes functional and temporal behaviors of major hardware components such that the effect of their speeds on system performance can be investigated. This simulator is driven by an executable workload, a set of assembly-language program codes which solves an application problem. Program codes are based on the instruction set of a general (not specific) processor. This study suggests a useful method to evaluate a parallel computer architecture before the detailed design and actual implementation are complete.

2. Simulation Model

〈Figure 1〉 illustrates the NxN parallel computer system incorporating the H/V-bus scheme, which is



〈Figure 1〉 Organization of NxN H/V-bus parallel processor system

used as a simulation model in this study. The configuration includes an NxN array of processing elements(PEs), 2N bus controllers(BCs), a Kernel, a system control processor(SCP), a run control processor(RCP), an input/output processor(IOP), SCP transfer memory(SCPTM), N IOP transfer memory (IOPTM), and a host computer. The host computer is a multiprogrammed general-purpose computer which provides program development and utility functions such as editing, file management, and language translation. The SCP is a dedicated computer which controls set-up, execution, and termination of a problem solving. It supports user interaction with the system through the user console. The RCP is a dedicated computer which coordinates the actions of the bus controllers.

The PEs, SCPTM, and IOPTMs are referred to collectively as bus elements(BEs). The SCP and RCP have buses to communicate with BCs, the Kernel, the IOP, and the SCPTM. There are N horizontal buses and N vertical buses interconnecting the BEs. Each bus has its own controller - a HBC or VBC - which handles bus arbitration and address translation, and controls operations on its bus by issuing commands to the BEs connected to the bus. Independent bus operations can occur concurrently on all horizontal buses and vertical buses. Each PE consists of a processor, a transfer memory and bus interfaces.

The H/V-bus system provides three mechanisms which enable a PE to communicate with other BEs: a broadcast operation which writes a data value from the transfer memory of a source PE to the transfer memory of destination BEs which need the data; a global read operation which reads data from the transfer memory of the BE holding the data to the processor in the PE performing the read operation; and processor synchronization and mutual exclusion. All data transfers among BEs are initiated by a PE and controlled by the mechanisms for the broadcast operation and the global read operation.

The Kernel processor provides mechanisms for processor synchronization and mutual exclusion among processors in PEs. The Kernel consists of a kernel processor, an arbiter and a kernel queue. PEs communicate with the Kernel through HBCs. The Kernel performs synchronization and mutual exclusion operations by responding to specific synchronization instructions executed by the PEs: P(sema) instruction, V(sema) instruction, and signal(event) instruction.

3. Simulator Development

The simulation for this study is based on the event-driven approach. The language used to implement the simulator influences its cost because it affects programming and debugging time, execution time, maintainability, readability, and transportability. Various languages have been used in implementing parallel computer simulator. SIMSCRIPT has been extensively used to simulate time-sharing, multiprogramming and multiprocessing systems. Katz[13] has simulated a multiprocessing system with SIMSCRIPT. The FORTRAN-based GASP IV simulation language was used by Askins and Pooch[1] to implement the microscopic-level simulation because of its ability to simulate both continuous and discrete events, and compatibility with most FORTRAN IV compilers. Corrigan and Johnson[8] used a GPSS model to simulate and evaluate a dual-processor 8085A multiprocessor system that employed a time-sharing method of accessing the system bus and memory. The simulation was based on the activity-scanning-approach feature of GPSS. Some of simulators are written in C language and run under the UNIX operating system[12,21]. In such simulators a process simulates a fuction of a node in parallel processor system. Use of general-purpose language like C, FORTRAN and PASCAL has an advantage in general availability, but has disadvantages in greater programming difficulty and requiring a programmer to write subprograms for general simulation utilities

which are already provided in simulation languages.

Based on the backgrounds, we develop the software simulator of the H/V-bus parallel computer system using SLAM II and FORTRAN 77 (referred to subsequently as SLAM and FORTRAN respectively). SLAM is a simulation-oriented extension of FORTRAN[19]. SLAM is excellent in ease of learning, ease of conceptualizing a problem, stastics-gathering capability, and available documentation[2]. Because discrete events in a SLAM simulation are coded as FORTRAN subroutines, the user has almost complete freedom to program logical constructs in FORTRAN.

The simulator does not include all of the detailed descriptions which would be required if a fine level of simulation was used for the entire system configuration. Instead of simulating the download process which loads object codes into the local memories of PEs, the simulator reads the object codes from an external data file and saves them into data arrays at the beginning of a simulation. The functions of the SCP, RCP and IOP are simulated, but their timings are not included. The SCPTM and IOPTM are not simulated.

As output, the simulator produces snapshots of the state of the computer system at different time periods and profiles of each job that goes through the system. It records statistics on such factors as the execution time of each processor, queue behavior, processor idle time, interprocessor bus utilization, and so on. From the outputs, validity, reliability (producing the same result on different runs in case of a deterministic model) and efficiency (a high ratio of simulated real-time to execution time of the simulator) can be determined. In the following sections event routines, timing characteristics and workload modeling are briefly described.

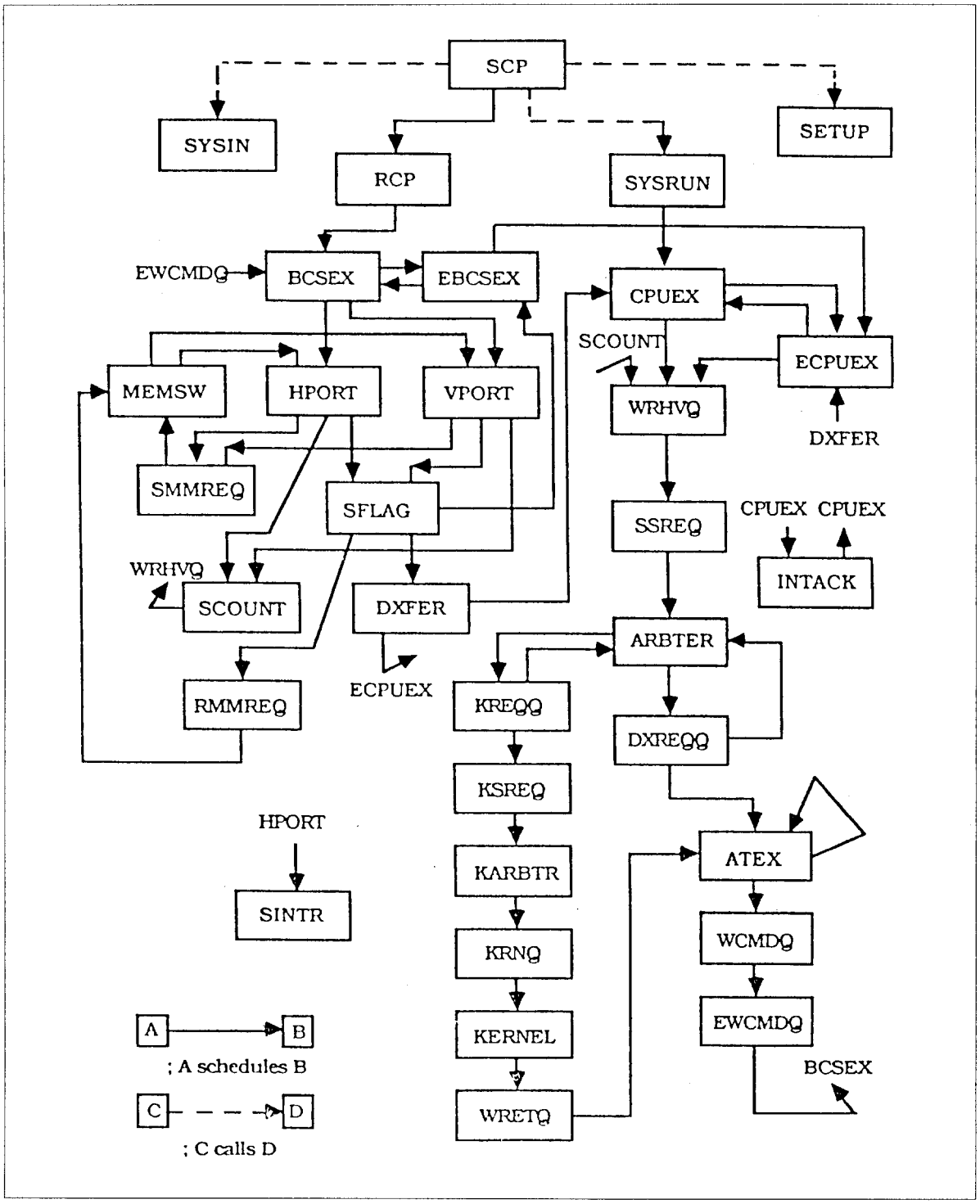
3.1. Event routines

The simulator for the H/V-bus parallel computer system consists of a main program, 29 event routines, 4 subroutines, and SLAM control statements which provide the SLAM processor with information about a simulation. Each event routine represents a system component, an activity in a PE, or an activity in an interprocessor communication (IPC) mechanism. Scheduling and calling relationships which exist among event routines and subroutines are depicted in 〈Figure 2〉. When a simulation study begins, the INTLC subroutine schedlues an initial event, the SCP event which calls the SETUP subroutine and the SYSIN subroutine to define the system state at time 0. Then the SCP event schedules the RCP event and calls the SYSRUN subroutine to schedule the CPUEX events, which simulate operations of processors in PEs, for all PEs involved in solving an application problem. The CPUEX event, which is scheduled repeatedly to execute a sequence of instructions assigned to each PE until problem solving is completed, initiates an interprocessor communication operation, when called for, by scheduling the WRHVQ event which simulates a write operation of an entry into a message queue. After that, events which are required for an interprocessor communication operation are scheduled in sequence. Details of events in 〈Figure 2〉 are described in [15].

3.2. Timing characteristics

The resolution used in simulating functions and operations of components involved in interprocessor communication is that associated with the propagation delay time of logic gates. The resolution of other operations, such as instruction fetch and execution operations by processors and local memory accesses, is that of the execution cycle time of assembly-language level instructions.

In the simulator the processors of all PEs are assumed to have the same instruction sets including floating-point arithmetic operations and special instruc-



〈Figure 2〉 Scheduling and calling relationships among event routines and subprograms.

tions for synchronization. Parameters incorporated into the simulator enable a user to change PE speed or IPC speed to study their effects on system performance. To provide a basis for comparing systems with different characteristics, a baseline system is defined. Assumptions for instruction-execution times of the baseline system are based on the execution times of assembly-language instructions of the microprocessor operating with 20MHz clock frequency. Execution times are kept in an array in the simulator. Instuction-execution times can be adjusted by using a system variable TCYCL which controls the clock period of the processor.

Assumptions for timings of components in the IPC mechanism are based on a preliminary design of hardware components and taken from databooks for CMOS memory chips and Schottky TTL ICs. In simulating the kernel processor, execution times of P and V operations are assumed to be 5 and 6 clock periods respectively; signal operations of all types are assumed to have a fixed execution time of 10 clock periods.

3.3. Workload development

As mentioned earlier, this simulator is driven by an executable workload, a set of assembly-language program codes which solves an application problem, partial differential equation. Given an application problem, a parallel algorithm is developed for solving it. The algorithm includes a separate process for each processor in the system. Each process is programmed in the assembly language of the processor, assembled into a sequence of machine-language instructions for the processor, and converted into a sequence of integers representing the instructions. A set of program codes for PEs are saved in an external data file which is read by the SETUP subroutine of the simulator at the beginning of a simulation. Initial variables and local variables are stored into arrays in the SETUP subroutine.

3.4. System variables for different configurations

Because the intended use of the simulator is to investigate the behavior of the H/V-bus parallel computer system under a variety of conditions, the simulator includes the following system variables which permit design parameters of the simulated system to be easily modified.

1) TCYCL represents the clock period of the PE in microseconds. This value is used to adjust instruction execution times in investigations of the effects on system performance of varying PE speed.

2) TFACTR is a ratio factor used to adjust the delay times of components in the IPC mechanism. Normally it is set to 1. To simulate components having delays that are half the original delay times, TFACTR is set to 0.5. This value is used in investigations of the effects on system performance of varying the speed of the IPC mechanism.

3) NUMPE is used to adjust the number of PEs in the system. This variable allows the system size to easily change.

4) MPRINT determines the mode of message printout. When it is set to 1, the simulator output contains only the most important messages. When it is 2, more detailed messages are produced. When it is 3, the simulator prints all messages which show system status each time an event occurs.

4. Experiments

Partial differential equation(PDE) is solved on the simulator as a benchmark to verify the simulation and prove its usefulness. The experiments are repeated for a single-bus system not only to show the high adaptability of the simulator but also to compare the system behavior of two systems.

The single-bus system employs a broadcast scheme and has a bus controller connected to its single bus. The single bus is similar to a horizontal bus in the H/V-bus system, but components not needed in the single-bus system are removed and the number of PEs that can be connected to the bus is limited because it might not be practical for a single-bus system to contain a large number of PEs due to excessive bus contention. Such a system can be simulated to prove a basis for comparing the performances of two systems in which the IPC mechanisms are implemented with components of the same speed. The software simulator of the single-bus system is easily derived from that of the H/V-bus system through the following modifications: remove the vertical-bus controllers (VBCs); remove all but one horizontal-bus controller (HBC); and remove components for operations on vertical buses in all PEs.

4.1. Partial differential equation

Deminet[9] and Raskin[20] have solved Laplace's equation with Dirichlet boundary conditions by the method of finite difference method on the Cm* multiprocessor[22,23]. The PDE

$$\frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} \qquad (4.1)$$

is solved on a rectanglular grid of size M x N, where only the values at the outer edges of the grid are given. The finite difference method[6] transforms the problem into a set of linear equations:

$$A\underline{x} = \underline{b} \qquad (4.2)$$

where $\underline{x}$ is an MN vector of all the points in the grid, A is an MN x MN sparse matrix and $\underline{b}$ is an MN vector derived from the boundary conditions. Given an initial estimate of the vector x, an iterative method can be used to solve equation (4.2). In each iteration a new estimate of the solution at each grid point is computed by averaging the solution values at the four nearest neighbors of each point. In the following experiments, the grid size is chosen to be 6x8 points (i.e., a linear system of 48 elements). The boundary conditions are chosen to be zero at all boundary points and the initial estimate of the solution at all grid point is 1.0. Because the boundary conditions are zero the solution is known to be 0 at all grid points. This results in a simple termination test since, in this case, the error vector is identical to the vector of new estimates computed during the current iteration. The error bound is chosen to be 0.1.

In this experiment, the Gauss-Seidel method [6] is considered. This method is an iterative algorithm for computing the solution of a set of algebraic equations. The following difference equation can be derived to solve equation (4.2) where all diagonal elements of the matrix A have the value 4:

$$x(i+1) = (I - \frac{1}{A})\, x(i) + \frac{1}{4} a = B\, x(i) + b \qquad (4.4)$$

where i is the iteration index, the matrix B is the Jacobi matrix, and vector b is zero because the boundary condition is zero. This method consists of repeatedly cycling through the process which computes a new estimate, using the most up-to-date value available, until a suitable convergence criterion is satisfied. During a computation period a PE broadcasts the new estimate of each variable immediately after it is computed. Therefore there is no distinct data exchange period when this method is used.

4.2. Simulation and analysis

The PDE using the Gauss-Seidel method is solved on the simulators for H/V-bus systems and single-bus systems containing 4 PEs (2x2 system), 9 PEs (3x3 system), 16 PEs (4x4 system), and for single-processor system. In solving this problem on a parallel

computer system the problem should be partitioned so that all PEs have approximately equal computation loads. Otherwise, some PEs will complete their tasks and become idle while waiting for PEs with heavier computation loads to finish their tasks, reducing speedup and efficiency. Problem partition for the systems with 4 PEs, 9 PEs and 16 PEs are illustrated in 〈Figure 3〉.

For the 4-PE system, all PEs have equal computation loads. For the 9-PE system, PE(1,1), PE(1,3), PE(3,1) and PE(3,3) have the heaviest computation loads and PE(2,2) has the lightest load. For the 16-PE system, computation loads are divided into two groups. One consists of computations for a grid point at a corner and for two grid points on edges; the second consists of computations for a grid point on an edge and for two grid points inside the grid plane. The second load requires more arithmetic operations than the first. Assembly-language programs for computation loads for individual PEs are converted into object codes and stored into data files which are read by the simulator at the beginning of a simulation.

To verify that the problem is solved correctly, the results generated by the simulators are compared with those generated by a sequential program written in FORTRAN and executed on the workstation on which the simulator runs.

In the experiments parameters of the parallel computer systems were varied for the following two cases:

Case (1): Clock frequency of the PE was varied.

Case (2): Speed of the IPC mechanism was varied.

In simulation, system variable TCYCL was used to implement case (1), and system variable TFACTR was used to implement case (2).

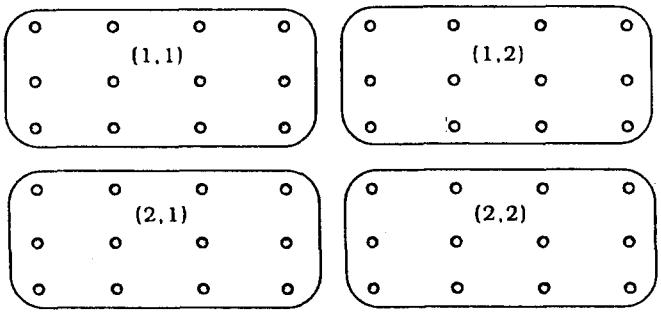
Case (1): Performance Versus PE Speed

This experiment demonstrates the effect of PE speed on system performance. 〈Figure 4〉 shows simulated execution times of H/V-bus systems with various number of PEs and the single-processor system for different clock frequencies of the PE. As PE speed increases (clock frequency becomes higher), execution times decrease for all systems. Resulting speedups are presented in 〈Figure 5〉. As PE speed increases speedup decreases because the entire single-processor solution benefits from higher PE speedup while only part of parallel solution (viz., computation time) benefits and overhead for parallel processing (viz., interprocessor communication time) are not reduced. As system size increases the decreasing rate become higher as shown. For the 4-PE system speedup of 3.75 is achieved. The speedup is less than the ideal value of 4 because of the imbalance in the number of convergence tests and the time required for processor synchronization at the end of each iteration. For the 9-PE system and 16-PE system speedups are worse due to factors such as the workload imbalance, the increased number of iterations required, and the higher traffics on interprocessor buses.

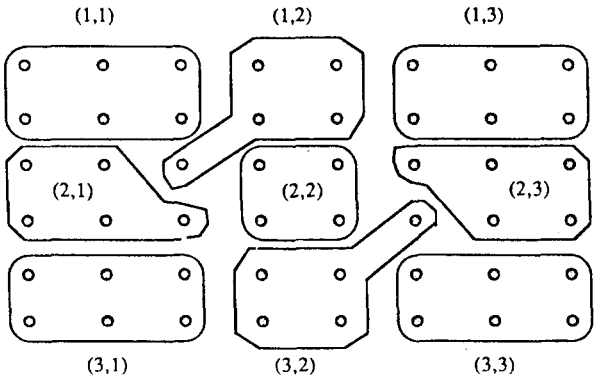
〈Figure 6〉 presents bus utilizations of the interprocessor buses during the total solution for 16-PE systems. The value of the H-bus (or V-bus) is that of the H-bus (or V-bus) with the largest number of bus operations. Bus utilization of the single-bus system is much higher than those of buses of the H/V-bus system. It is because all of interprocessor operations are performed on a bus in the single-bus system. For all systems, as PE speed increases bus utilizations become higher because the total execution time decreases while the time required for interprocessor bus operations is fixed.

Case (2): Performance Versus IPC Speed

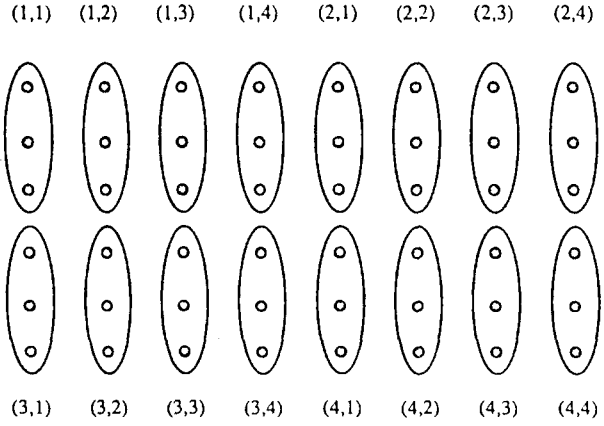
This experiment demonstrates the effects on system performance of varying the speed of IPC mechanism for three system sizes. In the experiment the IPC speed was varied by the factor of 2. Speedups are presented



(a) 4-PE system

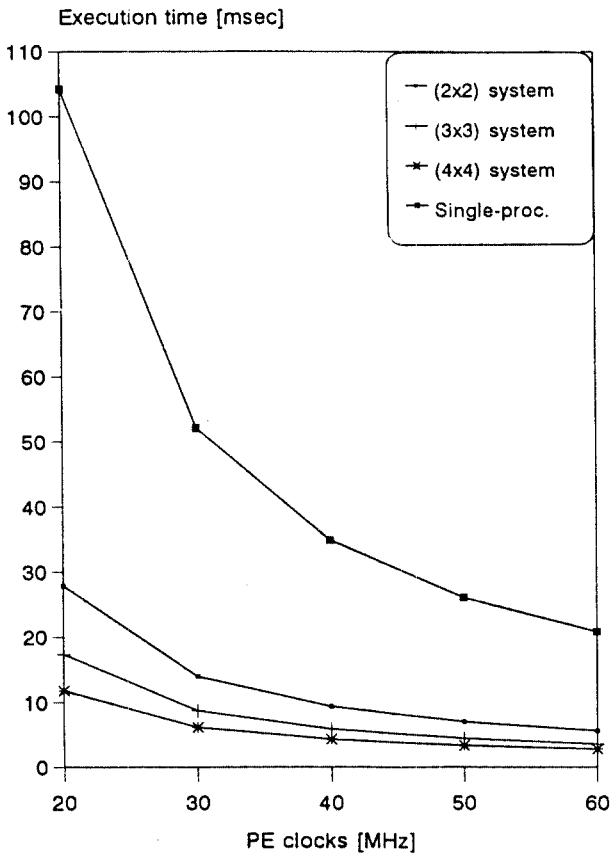


(b) 9-PE system

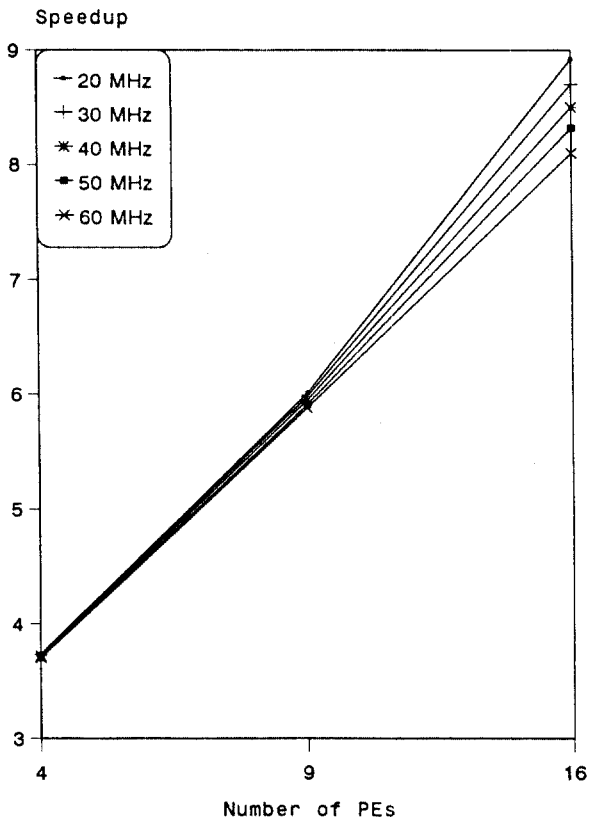


(c) 16-PE system.

〈**Figure 3**〉 **Problem partitions**



〈Figure 4〉 Execution times for various PE clocks.

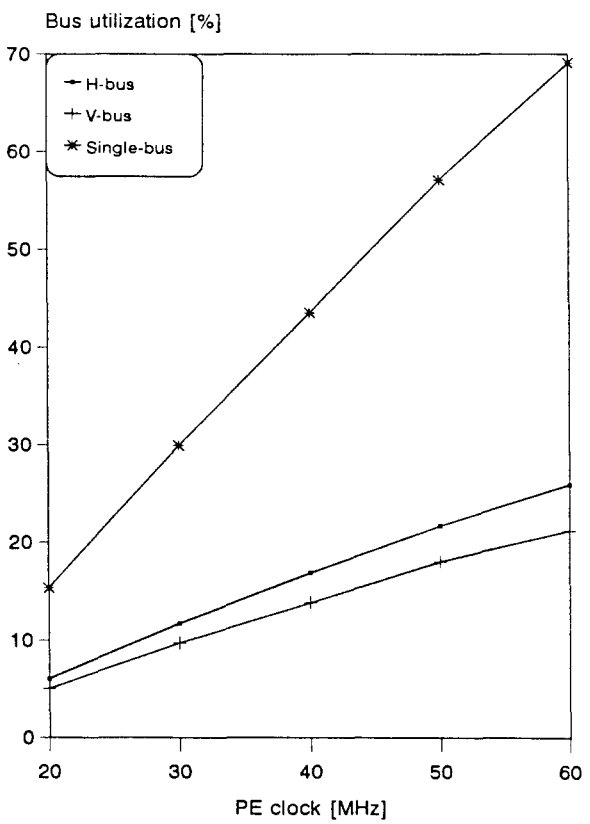


〈Figure 5〉 Speedups for various numbers of PEs.

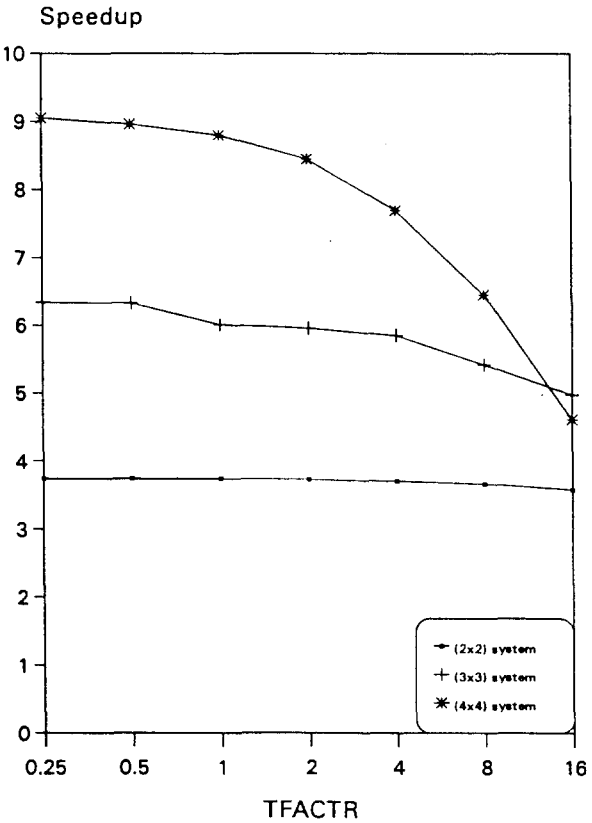
in 〈Figure 7〉 where x-axis is scaled by $\log_2$ (TFACTR) and the reference value (0) of the x-axis is the IPC speed of the baseline system discussed in the previous section. Larger TFACTR value means slower IPC speed. As IPC speed decreases, speedup decreases for all cases because the slower IPC speed increases execution times for all parallel solutions but has no effect on execution time of the single-processor system. When $\log_2$(TFACTR) approaches 4 (i.e., IPC speed is 16 times slower than the baseline system), (4x4) system is slower than (3x3) system because of the increased time for interprocessor communications and the increased number of iterations required: 22 iterations required for (3x3) system and 27 iterations for (4x4) system.

In this simulation, the experiment with TFACTR = 0.0 (ideal IPC mechanism) is performed. It is not practical, but the experiment is possible by simulation. In case of the 4-PE system, speedup is less than ideal (only 3.75) even though all four PEs have equal workloads and there is no delay in the IPC mechanism. The reasons are the imbalance in the number of convergence tests and the execution time of instructions for processor synchronization. This result indicates that parallel processing induces inherent overhead.

〈Figure 8〉 shows bus utilizations of the H/V-bus system and the single-bus system containing 16 PEs. As IPC speed decreases bus utilizations increase for all systems because the ratio of IPC time to the total execution time increases. The increasing rate of the single-bus system is higher than that of the H/V-bus



〈Figure 6〉 Bus utilizations for 16-PE system.



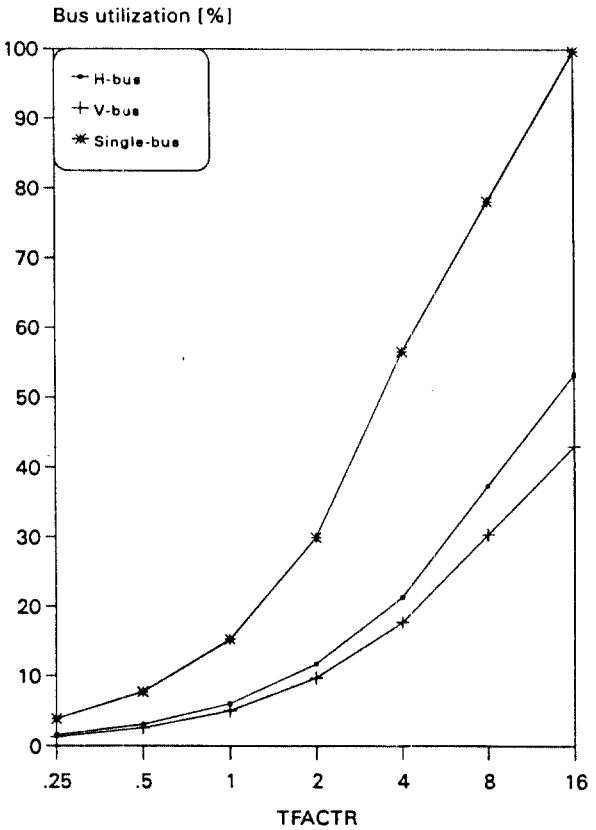
〈Figure 7〉 Speedups for various TEACTRs.

system due to higher traffics. When IPC speed is 16 times slower than the baseline system the bus utilizaiton of the single-bus system with 16 PEs is 99.71 %; essentially the bus is in continuous use.

5. CONCLUSIONS

The simulator made it possible to evaluate the performance of the H/V-bus system without building hardware. With the simulator it is relatively easy to change system size and component functions. By parameterizing key system characteristics (viz., PE speed and IPC mechanism speed) it was possible to study system performance over a wide range of conditions in a straightforward manner.

The use of the high-level simulation language SLAM, relieves the simulation programmer of writing subprograms for commonly encountered functions such as event scheduling, statistics collection, and sequencing events in proper chronological order. In addition to producing snapshots of the simulated system at different instants, the simulator generates a variety of statistics on factors such as processing element utilization, bus utilization and queue behavior. Since system operations are represented by event routines, the resolution to which events were simulated was selected to meet the objectives of the simulation which focuses on the operation of the IPC mechanism. In addition, simulation efficiency was greatly improved by eliminating computations not pertinent to the objectives; for example, using the next-event approach it



〈Figure 8〉 Bus utilizations for various TEACTRs.

was possible to avoid simulating the operations performed in loops in which system components wait for events to occur.

When a PDE is solved by the Gauss-Seidel method on the simulator of the H/V-bus system, speedups (and efficiencies) obtained for the baseline system with 4, 9, and 16 PEs are 3.74 (93.5 %), 6.01 (66.8 %), and 8.8 (55.0 %) respectively. Results demonstrate that speedup does not increase linearly and efficiency decreases when the number of PEs increases.

When a problem is partitioned for parallel solution, the partition itself limits the speedup that can be achieved because it generally results in the assignment of unequal computation tasks to individual PEs. For example, in the PDE solved on the 16-PE system the speedup is 9.13 (rather than an ideal speedup of 16) when the IPC mechanism introduces no delays and the processor characteristics are those specified for the baseline system. The experiments conducted with the simulator indicate that the speed of the IPC mechanism can also have a significant effect on speedup. In particular when IPC mechanism speed is varied over the range from the ideal case to a case in which the IPC mechanism speed is one-sixteenth of that specified for the baseline system, the speedup decreases to 4.61 in the 4x4 H/V-bus system and to 3.06 in the single-bus system with 16 PEs. These results present a strong argument in support of using a relatively high-performance IPC mechanism.

References

[1] Adkins, G., and Pooch, U.W., A Simulation Study of the IBM 370/168 and AMDAHL 470/V6, *Proceedings of the 10th Annual Simulation Symposium*, 1977, pp. 17-40.

[2] Banks, J. Carson, J.S., *Discrete Event System Simulation*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984

[3] Baudet, G.M., *The Design and Analysis of Algorithms for Asynchronous Multiprocessor*, Ph. D. Dissertation, Carnegie-Mellon University, Pittsburg, PA, 1978.

[4] Bell, T.E., Objectives and Problems in Simulating Computers, *Proceedings of the Fall Joint Computer Conference*, AFIPS Press, 1972, pp. 287-297.

[5] *Bipolar Microprocessor Logic and Interface Data Book*, Advanced Micro Devices, 1983.

[6] Burden, R.L. Faires, J.D. Reynold, A.C., *Numerical Analysis Prindle*, Weber & Schmidt, Boston, MA, 1978.

[7] Chang, Y.F., Development, implementation, and testing of a run time control mechanism for the H/V-bus parallel processor simulation system, *PPSP Memo*, no. 22, Arizona State University,

Tempe, AZ, 1984.

[8] Corrigan, B.E. Johnson, E.L., An Evaluation of 8085-based Multiprocessing on a Timeshared Bus, *IEEE Mocro,* vol. 7, 1985, pp. 11-21.

[9] Deminet, J., Experience with Multiprocessor Algorithms, *IEEE Trans. Computers,* vol. C-31, 1982, pp. 278-288.

[10] Ferrari, D., *Computer System Performance Evaluation,* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[11] Holliday, M., Stumm M, Performance Evaluation of Hierarchical Ring-Based Shared Memory Multiprocessors, IEEE Trans. on Computers, Vol.43, No.1, January 1994, pp. 52-67.

[12] Huesmann, L.R., Goldberg, R.P., Evaluating Computer Systems through Simulation, *Computer Journal,* Vol. 10, 1967, pp. 150-156.

[13] Katz, J.H., Simulation of a Multiprocessor Computer System, *Proceedings of the Spring Joint Computer Conference,* AFIPS Press, 1966, pp. 127-139.

[14] Kim, J.H., *Simulation and Performance Analysis of Interprocessor Communication Mechanism for MIMD system,* Ph.D. Dissertation, Arizona State University, Tempe, AZ, 1988.

[15] Kim, J.H., Program Listings for H/V-bus System Simulator, *PPSP Note,* no. 7, Yonsei University, 1992.

[16] Kumar, M. So, K., Trace Driven Simulation for Studying MIMD Parallel Computers, *Proceedings of International Conference on Parallel Processing,* 1989, pp. 68-72.

[17] Macdougall, M.H., Computer System Simulation: An Introduction, *Computing Surveys,* vol. 2, 1970, pp. 191-209.

[18] Navaux, P., SSIP - A Processor Interconnection Simulator, *Proceedings of the 10th World Congress on System Simulation and Scientific Computation,* IMACS Press, 1982, pp. 329-332.

[19] Pritsker, A.B., *Introduction to Simulation and SLAM II,* John Wiley & Sons, New York, New York, 1984.

[20] Raskin, L., *Performance Evaluation of Multiple Processor System,* Ph.D. Dissertation, Carnegie-Mellon University, Pittsburg, PA, 1978.

[21] Smith, J.E., A Simulation Study of Decoupled Architecture Computers, *IEEE Trans. Computers,* vol. C-35, 1986, pp. 692-702.

[22] Sohi, G.S., High-Bandwidth Interleaved Memories for Vector Processors - A Simulation Study, IEEE Trans. on Computers, Vol.42, No.1, January 1993, pp. 34-44.

● 저자소개 ●

김종현

1976년 연세대학교 공과대학 전기공학과 졸업(공학사)
1981년 연세대학교 대학원 전기공학과 졸업(공학석사)
1988년 Arizona State Univ. 컴퓨터공학과 졸업(Ph.D)
1976년~1982년 국방과학연구소 연구원
1982년~1983년 금오공과대학 전자공학과 전임강사
1988년~1990년 한국전자통신연구소 프로세서구조연구실장
1990년~현재 연세대학교 원주캠퍼스 전산학과 부교수