A Java-based Parallel Platform for the Implementation of Evolutionary Computation for Engineering Applications

Chun Che Fung
School of Information Technology
Murdoch University
Murdoch, WA 6150, AUSTRALIA
L.Fung@murdoch.edu.au

Ja Bin Li
School of Engineering
Curtin University of Technology
Bentley, WA 6102, AUSTRALIA
lijb@iinet.net.au

Kok Wai Wong
School of Computer Engineering
Nanyang Technological University
REPUBLIC OF SINGAPORE
ASKWWong@ntu.edu.sg

Kit Po Wong
Department of Electrical Engineering
Hong Kong Polytechnic University
SAR CHINA RPC
eekpwong@polyu.edu.hk

Abstract- This paper proposes an extended version of a previously developed low cost parallel computation platform called Para Worker. The new system is termed Para Worker 2 which differentiates from the early system. The new proposed system adds enhanced features of improved dynamic object reallocation, adaptive consistency protocols, and location transparency as compared to the original system. The proposal is particularly useful for the implementation and execution of computational intelligence techniques such as evolutionary computing for engineering applications.

1 Introduction

Computational Intelligence (CI) or soft computing techniques mainly concern with Artificial Neural Networks (ANN), Fuzzy Logic (FL) and Evolutionary Computation (EC). Such techniques are viable approaches to solve many real-life and complex engineering problems. For example, in electrical power engineering, practical problems such as load-flow analysis, unit commitment, economic and environmental dispatch, maintenance scheduling, risk analysis for a complex network will require massive computational efforts. In particular, where traditional solution processes are not applicable due to the non-linearity and complexity of the problems, CI techniques has proven to be a viable alternative solution (Fung, Chow et al. 2000).

In the case of ANN and EC, one of the key features of these techniques is the inherent massive parallelism. This particular characteristic enables the solution algorithm to be represented and computed in a distributed environment. However, this is only feasible provided that the issue of high communication overhead is resolved. Over the past years, a large amount of researches have studied the implementation of ANN on parallel structures. Furthermore, parallel algorithms on EC's such as genetic algorithms have also been widely developed and tested (Wong, Li et al 1997).

By the nature of EC techniques, execution of the algorithm will be more efficient if it is carried out in a parallel computing platform. Depending on the design, such parallel platforms can be expensive if dedicated systems have to be designed and custom-built. Efforts have been carried out to overcome such problem. A possible solution is the Distributed Shared Memory (DSM) architecture. This provides a low-cost yet flexible and powerful means of constructing a parallel platform. In terms of the construction, a parallel platform can either be designed as homogenous, such as a cluster of computers, or heterogenous, such as a network of workstations (NOW). From the standpoint of cost, the heterogenous class platform is preferred as it almost requires no extra cost to build. It is now a common practice that NOW has been installed as a standard platform in most work environments. Furthermore, the speed of Ethernet has also been increased up to Gb/s level. This consequently has caused a relative low latency connection among the network and thereby reducing the communication overheads.

From the software perspective, in order to maintain a low overall system cost, it is prudent to use a programming language that has the following features:

- good portability and interoperability, that is, the program needs virtually no or minimal modification to run across various platforms;
- facilitate reusability;
- support of a certain degree of parallelism; and,
- support of synchronisation mechanism.

Java is one of the most popular object-oriented languages which meets the abovementioned requirements. Furthermore, additional feature of Java which supports complex engineering projects is a high degree of software reusability which permits the programmers to simplify the software development process. This again reduces the development cost.

0-7803-7804-0 /03/$17.00 © 2003 IEEE

The importance of threads should also be emphasised which is used to facilitate parallel algorithms. Thread can realize four classes of implementation of parallel algorithm: Master Mechanism, Peer Mechanism, Class Transfer Mechanism, and Message Passing Mechanism. Java provides a standard *Thread* class for the purpose of implementation of such algorithms. In addition, Java also provides several synchronisation mechanisms to access share variables or objects. For this reason and following the previous work in ParaWork (Fung, Chow et al.), Java has been adopted as the development language for the present study.

In the subsequent sections, this paper first reports the various types of DSM designs as described in the literatures (Amza, Cox et al. 1999). The strengths and weaknesses of these DSM approaches are also been discussed. Finally, a new design of DSM based on the DISK (DIStributed Kaffe) (Surdeanu and Moldovan 2002) distributed Java Virtual Machine (JVM) is proposed. The new DSM is term Para Worker 2, which is an extended project from the Para Worker project (Fung, Chow et al.). This paper is organised as follows: Section 2 gives an overview of the development of the parallel platform and Section 3 discusses the proposal on the design and implementation of Para Worker 2. Section 4 gives a comparison between the two versions of the Para Worker while Section 5 discusses the implementation. This is followed by a conclusion for this paper.

2 Overview

There are three major design approaches to implement parallel computers: Multiprocessors, Multi-computers, and Distributed Shared Memory (DSM). In the case of a multi-processors design, all the processors are confined and share the resources within a single, global address space. Read and write operations are both allowed for any processor to access the common address space. Communication among processors is thereby made through the shared memory. This architectural design is also termed as a tightly coupled parallel system. On the other hand, a multi-computers system comprises of a network of computers. Every computer in the network has it's own private memory space and I/O modules. Hence communication is via message-passing. In practice, multiprocessors systems are costly and complicated to build but they are relatively easy to program. On the other hand, multi-computers are just the opposite. Distributed Shared Memory architecture can be considered as an intermediate solution which takes advantages of the two design philosophies. In DSM, each computer has both local and shared memory space. The shared memory space is visible to all remote processors located in the same network. From a programmer's perspective, the read or write operations on the shared or local variables are being treated the same irrespective to their physical locations. In other words, variables stored in the shared memory space can be accessed by a remote processor in a same manner as variables being stored in the processor's local memory space.

There have been a number of implementations of the DSM design in the past years. Examples of such developments are Sequential Consistency, Release Consistency and Entry Consistency. Sequential Consistency follows closely to a single processor environment. IVY was the first page-based DSM machine implemented in this manner (Li and Hudak 1989). It used sequential consistency model, which is based on a *single writer multiple reader* protocol. In this case, only one node will be permitted to write to the shared page at anyone time whereas read operation can be concurrently carried out by the other processors. During the write operation, all the other nodes which intend to access to the same page have to be suspended and wait for the completion of the write operation. This leads to low system performance. Another shortcoming is the false sharing problem which is due to concurrent attempt by more than one processor to gain ownership of the same page although each processor is accessing to distinctly different sets of data. On the other hand, the lazy release consistency (LRC) model allows postponement of the update until a synchronisation variable is acquired by a remote processor. TreadMarks is a page-based DSM system proposed by Amza et al (Amza, Cox et al. 1996)based on the LRC approach. The *false sharing* problem is overcome by using different modes of *multiple writer protocol* which allow concurrently access by different processors to different sets of data within a page. This has resulted in an improvement in the system performance. Midway (Bershad, Zekauskas et al. 1993) is another shared-variable DSM system which implements the Entry Consistency model. The entry consistency model associates a synchronisation variable with every shared variable. The approach updates or invalidates shared variables associated with the acquired synchronisation variable. Another example is JUMP (Cheung, Wang et al.) which is a page-based DSM system built on a cluster of computers. JUMP implements the *migrating-home* protocol. The protocol is based on a new consistency model *Scope of Consistency* (ScC). This approach was implemented as user-level run-time library built on the UNIX environment. ScC bridges the gap between the inefficiency in LRC and the hard programming requirements in the Entry Consistency model. Subsequently, the Java-Enabled Single-System-Image Computing Architecture (JESSICA) (Ming 1999) was developed as a JVM-based middleware that runs on top of JUMP in the standard UNIX operating system. JESSICA supports parallel execution of multi-threaded Java

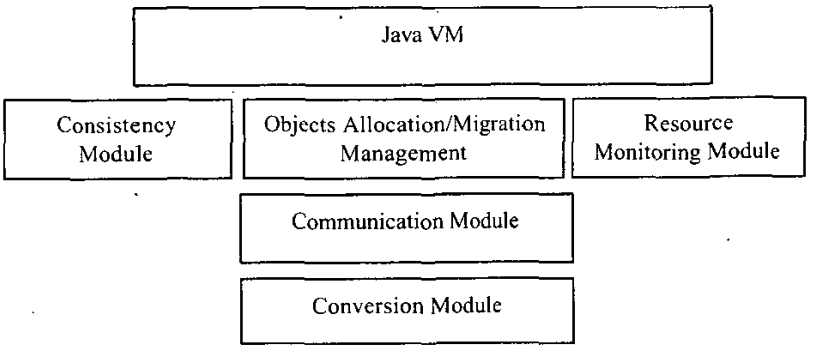

Fig. 1: Block diagram of the Para Work 2 architecture.

applications in a cluster of computers.

Other developments are the implementation of object-based distributed systems. Emerald (Black, Hutchinson et al. 1987) is a distributed object system which implements local communication among threads and objects through shared memory. Remote communication is carried out by Remote Procedure Call (RPC). RPC arguments are marshalled into one message. Pointer to the physical addresses can also be accessed remotely. However, objects migration is not supported and this being the main disadvantage of the system. The other disadvantage is that the locations of the shared variables are not transparent to the programmer.

Orca (Bal, Kaashoek et al. 1992) is another object-based system that supports transparent replication by a fast, reliable broadcast mechanism. The mechanism allows all object copies to be propagated simultaneously. Orca implements sequential consistency. Orca objects can be in one of two states: *single-copy* or *replicated*. Replicated objects are maintained consistent with the broadcast mechanism. Single-copy objects are accessed through RPC.

DISK (Surdeanu and Moldovan 2002) implements an object-based, multiple-writer consistency protocol (OMW). It also characterises the update-based, multiple-writer lazy release consistency protocol. DISK is a distributed Java virtual machine. Threads run on a set of nodes connected through software point-to-point TCP channels. Remote objects can be in either *read-only* or *read-write* states. An update-notice for each modified object is created, and only the differences (diffs) are propagated to the remote processor acquiring the synchronisation variable. Objects in DISK are automatically assigned to an unshared state until a remote thread accesses them. This results in a reduction in the overhead for the maintenance of consistency between the objects.

Java/DSM (Yu and Cos 1997) is a JDK 1.0.2-compliant distributed JVM implemented on top of TreadMarks. Java/DSM allocates objects and classes into heaps with TreadMarks shared-memory allocation routines. JavaParty (Philippsen and Zenger 1997) implements a Java-based distributed environment. JavaParty is built as a cluster of regular JVMs connected via Remote Method Invocation (RMI). JavaPary introduces a new class, *remote*, to mark objects as shared. A pre-processor is required to translate this code into RMI-based Java code.

3 Design of Para Worker 2

The design of the Para Work 2 is adapted from DISK introduced in (Surdeanu and Moldovan 2002). Fig.1 illustrates the block diagram of Para Worker 2. The proposed system improves the original system by adding:

- *adaptive protocols to the Distributed JVM*. Taking the advantages of adaptive protocols (Amza, Cox et al. 1999), Para Work 2 automatically chooses between the different protocols depending on the access patterns of the application and the workload in the network. It is recognised that the performance of a protocol is application dependent; and,
- *a dynamic object migration technique*, which allows the proposed system to dynamically balance the load on the network and processor, and effectively reduces the latency of the communication.

To achieve our goal, Para Worker 2 adds the Resource Monitoring module to the original system. The module is to monitor:

- network environments, such as link congestion and processor and recourses failures;
- load on its processor; and,
- access patterns of share objects by threads.

First of all, based on information provided by the monitoring module, the objects allocation/migration management decides the object reallocation based on the following conditions:

- Imbalance load on the resource. For example, a processor maintains too much or too few objects due to object creation and discard.

- High latency in the network due to traffic congestion by a large amount of messages sending to the same node.
- Faulty processors or links.
- Complexity of the application.

The function of the object allocation is important in DSM systems, especially in a dynamical varying environment. Furthermore, the Monitor module also informs the Consistency module the access patterns of the share objects accessed by the remote threads. The system in turn considers the adaptation of appropriated protocols accordingly to order to improve the performance of the application under execution. Similar to the system presented in (Amza, Cox et al. 1999), Para Worker 2 is to implement the following protocols:

- Single- and multiple-writer protocols;
- Invalidate and update protocols.

Para Worker 2 also includes two additional protocols that support object migration: Home-based and home-less protocols. First, in the single-writer protocol, it allows only one single writable object in the system at any given time; however, there are possible several read-only copies of the object stored in other processors. The processor currently holding the writable copy of the object is called the owner of the object. To write to an object in a single-writer protocol, a processor must obtain the ownership of the object from the previous owner at the synchronisation time. In contract, the multiple-writer protocol allows multiple writable copies of the share object to coexist within the system. These copies of objects are synchronised by the execution of the barrier code to join the objects as one copy according to their time-stamps. Taking advantages of the multiple-writer protocol, the system can improve running speed of the application by allowing different parts of the object to be modified concurrently. However, for some applications which allow only one writer at any time, the single-writer protocol can simplify the process of modification by avoiding the costs of twinning, diffing, and diff combining.

Secondly, the major difference between the invalidate and update protocols is the invalidate protocol which allows postponing of the modification of a share object until it is accessed by another thread. The tradeoffs between invalidate and update protocols are studied and presented in (Amza, Cox et al. 1996). Basically, significant amount of data is sent by the update protocols. However, access-fault and round-trip delay can be reduced dramatically with the advantage of speculative update protocols.

For the home-based protocols, the owner of a shared object is allocated to a fixed processor, or Home, which is responsible for the maintenance of the most up-to-date copy of the object. When a remote processor requests the object, the processor must communicate with the Home. In contrast to home-based protocols, in the homeless protocols, every processor has the updated copy of the object, a processor has to send out request to all processor to acquire the most updated copy of the object. The strength of the Home-based protocol is similar to the server-client algorithm. This effectively controls and keeps a copy of the object. However, the overhead issue is still in place. The Homeless protocol however reduces the overhead of the Home node but the overall amount of request messages is increased.

4 Comparison with Para Worker

The differences between two proposed systems are summarised as the following aspects:

4.1 Location Transparency

Para Work requires programmers to write separate codes for different machines such as servers and clients. The task is generally difficult and error-prone. Taking advantage of object allocation management, Para Worker 2 allows location of objects to be handled automatically. At the start of the application, threads are distributed throughout the network. There is unnecessary to write codes for different machines. Furthermore, programmer can avoid programming any complicated object synchronisation with the help of Para Worker 2. The system automatically detects the status of any object, such as share and unshared (Surdeanu and Moldovan 2002). For share objects, consistency function calls are inserted before object and class accesses instructions.

4.2 Programming Interface

Para Worker 2 uses JDK compliant Java Virtual Machine, programs which require little modification to be run on an existing system. However, in Para Worker, the network configuration such as the network topology has to be performed by the programmer prior to the execution of the program.

4.3 Run-Time Environment

Para Worker 2 allows objects and threads reallocating dynamically for the application to adapt changing run-time environment. For instance, threads stored in the faulty processor are transferred to other processor without restoring the system. In contrast, Para Worker does not consider dynamic changing of the network. Consequently, any failure occurred in the Para Worker system will require restoration of the application.

5 Implementing Evolutionary putation algorithms with Para Worker 2

Two approaches are commonly used to implement distributed Java virtual machines. The first approach is to build Java virtual machines on the top of a Memory Management Module, as illustrated in Fig. 1. Such module creates a Global Thread Space across a group of clusters or workstations. It in turn allows Java threads freely move

from one machine to another without user's knowledge of their physical location. The advantages of this approach are twofold. First, as memory management module is separated from Java virtual machine, minimal changes are required to be made to the original program code of the virtual machine. Further, maintenance and upgrading of the Java virtual machine will be relatively easier for a programmer. Second, it assists programmers to write efficient Java thread programs. As mentioned early, shared-memory structure provides easy programming environment to programmer.

On the other hand, DISK implementation is based on tightly connected Java virtual machine and consistency protocols. With modified Just-In-Time (JIT) compiler, DISK invokes consistency functions before object and class write-access instructions. One innovation of DISK implementation is that only write-access to shared objects or classes requires inserting consistency function calls. It thereby is able to reduce communication overheads. However, DISK approach requires extract efforts to maintain the software. Most importantly, the approach limits DISK to be implemented by hardware.

Implementation of Para Worker 2 follows the principle of Java/DSM and JESSICA. It implements Kaffe virtual machine on top of some software DSM programs, such as JUMP or TreadMarks. However, the memory management module in Para Worker 2 is capable to dynamically allocate objects according to balance law of the network.

Implementation of Evolutionary computation algorithms in the Para Worker 2 is achieved via threads. In the previous work, an implementation of Genetic Algorithm (GA), Simulated Annealing (SA), and Tabu Search (TS) was used to solve the Economic Despatch problem in electrical power engineering. A flowchart of the algorithm is shown in Fig. 2 Instead of pre-loading the execution programs in each machine within the network, the program is implemented as threads. Each thread is a computation agent implementing the integrated GA, SA and TS algorithm. During execution, each of such thread objects is distributed through the system automatically. Communication among these threads is viable by shared objects. Depends on the design of sharing objects, Para Worker 2 is able to simulate either cluster computing or parallel computing as described previously.

For cluster computing, a cube consisting of a number of processors, each one consists of a single thread, is used. After each specified generation, each node will communicate with its neighbouring node and sends its best chromosome, in terms of objects, to each neighbour. It also receives the best chromosome from the neighbour nodes. The received chromosome will be included in the regeneration process. This model of the cluster computation is illustrated in Fig. 3.

For parallel computing, a group of slave threads is working on a global object space to perform crossover and mutation. The data is then recombined to perform crossover and mutation by a super thread, called the master thread. After that, it is sent back to the slave threads for repeated manipulation of data until the maximum number of generations is reached. The model implemented for parallel structure is illustrated in Figure 4. Techniques of enhancement like SA and TS are incorporated in both the parallel and clustering structure.

As an example, for clustered structure processing, at the end of a number of generations, the best chromosome is passed to its neighbouring thread. In the case of parallel structure processing, each slave thread will process for a number of generations and the master will process for 100 generations and this is repeated for a number of rounds.

Due to the master thread has similar structure to the slave threads, the complexity of the program is effectively reduced. In particular, the Resource Monitoring Module will identify the load condition of the neighbouring nodes and the Object Allocation/Migration Management module will take care of the allocation of the threads to appropriate the appropriate nodes. The system is currently under development and comparison of the system performance to ParaWorker is studied.

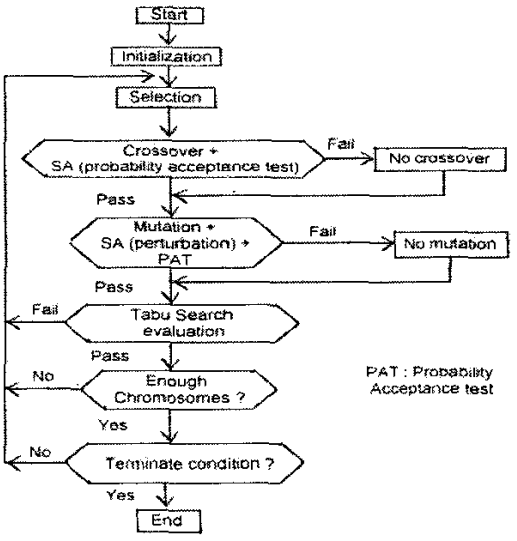

Fig. 2: An Integrated GA, SA and TS algorithm

6 Conclusion

This paper proposed an extended version of a previously developed low cost parallel computation platform called Para Worker 2. The new system is based on the Java/DSM distributed Java Virtual Machine. It adds enhanced features of improved dynamic object reallocation, adaptive consistency protocols, and location transparency. The protocols included are single- and multiple-writer

protocols, invalidate and update protocols, home-based and home-less protocols. The proposal is particularly useful for the implementation and execution of computational intelligence techniques such as evolutionary computing for engineering and other applications.

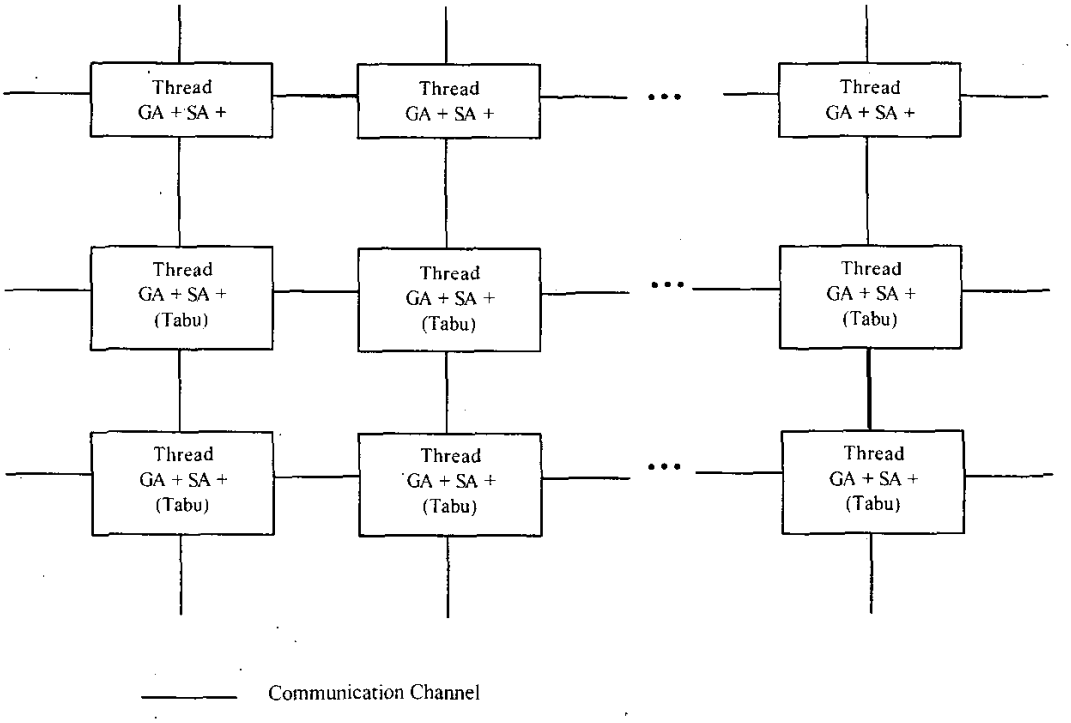

Fig. 3: Cluster Structure (Coarse-grain) via the thread parallelism

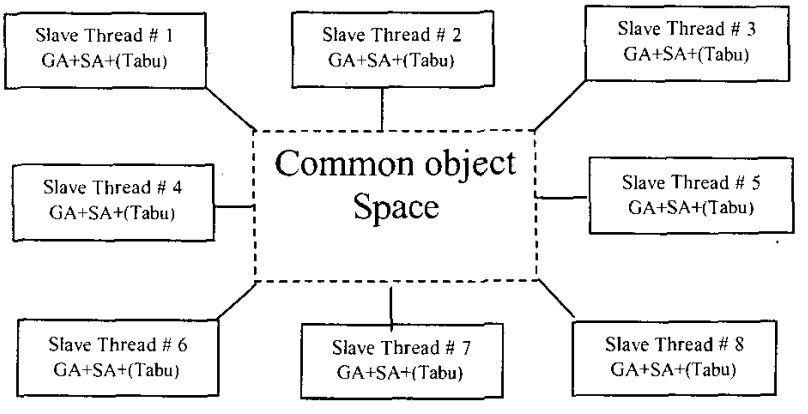

Fig. 4: Model of Multi-thread Implementation of EC.

References

Amza, C., A. L. Cox, et al. (1999). Adaptive Protocols for Software Distributed Shared Memory on Networks of Workstations. Computer. **29**.

Amza, C., A. L. Cox, et al. (1996). TreadMarks: Shared Memory Computing on Networks of Workstations. Computer. **29**.

Bal, H. E., M. F. Kaashoek, et al. (1992). "Orca: A Language for Parallel Programming of Distributed Systems." IEEE Trans. on Software Engineering **18**(3).

Bershad, B. N., M. J. Zekauskas, et al. (1993). The Mdway Distributed Shared Memory System. Proc. IEEE CompCon Conf.

Black, A., N. Hutchinson, et al. (1987). "Distribution and Abstract Types in Emerald." IEEE Trans. on Software Engineering **13**(1): 65-74.

Cheung, B. W. L., C.-L. Wang, et al. "A Migrating-Home Protocol for Implementing Scope Consistency Model on a Cluster of Workstations."

Fung, C. C., S. Y. Chow, et al. (2000). A Low-Cost Parallel Computing Platform for Power Engineering Applications. Advances in Power System Control, Operation and Management, 2000. APSCOM-00, Hong Kong, IEE.

Li, K. and P. Hudak (1989). "Memory Coherence in Shared Virtual Memory Systems." ACM Trans. Computer Systems **7**(4).

Ming, M. J. (1999). JESSICA: Java-Enabled Single-System-Image Computing Architecture. Hong Kong, University of Hong Kong.

Philippsen, M. and M. Zenger (1997). "JavaParty--Transparent Remote Objects in Java." Concurrency: Practice and Experience **9**(11).

Surdeanu, M. and D. Moldovan (2002). "Design and Performance Analysis of a Distributed Java Virtual Machine." IEEE Tran. on Parallel and Distributed Systems **13**(6).

Wong, K. P., Li, A., Law, M.Y. (1997). Development of constrained-genetic-algorithm load-flow method. Generation, Transmission and Distribution, IEE Proceedings-. **144:** 91-99.

Yu, W. and A. L. Cos (1997). Java/DSM: A Platform for Heterogenous Computing. Proc. ACM 1997 Workshop Java for Science and Eng. Computation.