Barriers to Creating a Secure MPI *

Ron Brightwell David S. Greenberg Brian J. Matt
Massively Parallel Computing Research Laboratory
Sandia National Laboratories
P.O. Box 5800 M.S. 1110
Albuquerque, NM 87185-1110
[bright,dsgreen,bjmatt]@cs.sandia.gov

George I. Davida
Department of Computer Science
University of Wisconsin-Milwaukee
P.O. Box 784
Milwaukee, WI 53201
davida@cs.uwm.edu

Abstract

This paper explores some of the many issues in developing security enhanced versions of MPI.

The problems that arise in creating a security enhanced MPI for embedded real-time systems supporting the Department of Defense's Multi-level Security policy (DoD MLS) are presented along with the preliminary design for such an MPI variant.

In addition some of the many issues that need to be addressed in creating security enhanced versions of MPI for other domains are discussed.

1 Introduction

The Message Passing Interface (MPI) has become the *de facto* standard for message passing in high performance parallel and distributed computing environments. As such MPI must deal with an wide spectrum of systems and security policies / environments.

These systems include single site machines ranging in scale from small MPP avionics systems for the Military [15] to extremely large MPP systems. In addition the systems can be distributed in nature, for example small collections of large machines [9]. Even large distributed collections of very modest machines with high latency have many uses.

The security requirements of MPI users also span a wide spectrum, from high assurance systems enforcing strict well-defined security policies, such as DoD MLS [16] and related policies (such as DoE policies), to corporate policies and security policies of universities.

The work that needs to be done to address such diverse range of systems has not been part of the MPI-1 standard work [13] or the on-going MPI-2 process [14]. Previous work in adding security features to MPI by Foster et. al. [9] has focused on adding security features to protect communication traffic both as part of process creation and for "computational" traffic.

Our work is currently focused on providing a version of MPI suitable for embedded real-time systems using the DoD MLS security policy. Systems of this type introduce problems for MPI implementations, including unidirectional message passing between certain processes. Many of the flavors of MPI message passing, both point-to-point and collective, were not designed to be compatible with unidirectional connections. Additions to MPI were necessary to deal with an external (to the process) specification of the process' "security environment" that is provided by the system. Also, extensions to the group / communicator API are required to allow MPI applications to create communicators that conform to "security contexts".

In addition to the above work we are exploring issues in creating secure MPI implementations suitable for other domains, including MPI-2 features for process creation and real-time.

In Section 2 we give a brief overview of MPI followed by a description in Section 3 of an SMPI design for embedded real-time systems supporting the DoD MLS security policy. In Section 4 we will present issues that are outside the context of embedded MLS systems and need to be explored before a more general SMPI can be created.

2 MPI

Readers desiring an in-depth description of MPI may wish to read the standard[13] or any of the summary references listed at http://www.mcs.anl.gov/mpi/mpich. In this section we concentrate on just a few features which have particular repercussions for attempts to create secure versions of MPI.

A defining feature of MPI is its ability to provide safe message passing contexts, or communicators, within a

*This work was supported in part by DARPA under ARPA Order No. C974.

DISCLAIMER

group or subgroup of processes. Communicators can be utilized by layered libraries to preclude the possibility of messages interfering with each other. Secure implementations may wish to interact with the communicators to add enforcement to the defined separation. The implementation of communicators is typically through collective communication among the processes involved.

MPI is typically implemented as a library which is linked to user code at compile or runtime. The library nature of MPI allows for flexible insertion of MPI variants and therefore allows easy experimentation with secure variants.

The core of MPI is, of course, its facilities for data movement. MPI supplies both point-to-point and collective data movement operations. In order to provide flexibility and capture as much of the pre-existing semantics as possible within a single standard, it defines several distinct semantics and progress rules for these data movement functions. While these semantics do not explicitly mandate a specific implementation, any secure implementation will have to address implications such as synchronization and acknowledgments within the protocols which realize the semantics.

We briefly describe the completion semantics of the different modes of peer communication defined within MPI. We concentrate on the need for acknowledgments, the need for synchronization, and the use of buffering.

In the standard mode, a send might begin before a matching receive is posted, but completion of the operation may also depend on the occurrence of a matching receive. It is left to the implementation to decide where and if standard mode send messages are buffered.

Likewise, the completion of a synchronous send is nonlocal, and is guaranteed to complete only after a matching receive operation has begun at the destination. Thus, this communication mode can be used as a rendezvous point.

The ready send mode offers an opportunity for optimization by insuring that a matching receive has been posted, possibly eliminating some of the protocol overhead needed by standard mode sends. Completion of a ready send is local.

Likewise, completion of a buffered send operation is also local. In this mode, the sender provides space to buffer outgoing messages so that the completion is independent of any action at the destination. The operation can finish whether or not a matching receive has been posted.

There are also some additional semantics mandated by the progress rules of non blocking communication that force both the sender and receiver to coordinate to insure proper completion of certain operations.

In addition to point-to-point communication, MPI contains several collective communication operations which provide the ability to do broadcast, gather, scatter, and reduction operations across a group of processes. As with point-to-point messages from different communicators, collective communication messages are guaranteed not to interfere with point-to-point messages within the same communicator. Within each communicator there is a safe message passing subspace for both peer and collective communication.

3 SMPI for Embedded Real-time MLS Systems

In this section we present a version of MPI suited for use in a embedded real-time system that implements the DoD MLS security policy: the PROSE operating system [15]. First we briefly describe the PROSE system and then the design and implementation of this version of SMPI. The PROSE system is a secure real-time operating system being developed by Hughes Aircraft, Intel and Sandia National Laboratories under DARPA sponsorship. For further details on this project see [15]. Here we summarize the current design for message passing and other aspects of PROSE that are relevant to SMPI.

3.1 The System

PROSE utilizes the System Build [1] approach developed by TIS and Hughes to provide access mediation at build and initialization time, rather than at run time. The Privilege Control Table Toolkit (PCT Toolkit) [10] produces a Privilege Control Table (PC Table) for each binary image that contains:

- A list of access privileges
- A mapping of each process that is visible to the image to its corresponding subject.

Applications are composed of one or more subjects, each of which may be at a different security level or compartment. Each subject is composed of one or more processes. A process' PC Table supplies information that allows it to send messages to those processes outside of its own subject. Messages to processes within the same subject are not mediated by the PROSE kernel, and therefore no PC Table information is required. All mediation by the PROSE kernel is done at message generation time rather than at message reception.

3.2 Implications

MPI was not designed to operate in environments such as the PROSE system. In our work we have incountered the following issues:

Unidirectional flows

The designers of MPI assumed that all communication

channels between processes are bidirectional. However, as seen above in the PROSE system, this is not the case [1].

Strict unidirectional links pose problems for MPI implementations. For example, the synchronization that exists between sender and receiver in certain modes of point-to-point communication is not compatible with unidirectional links. Note that the impact is at two levels; one is what the application code must see to conform to the MPI semantics, and the other is what might be available to the application because it is available to the application's MPI implementation. Unless the MPI implementation is protected from the application, one must conservatively assume that all information available to an MPI implementation is also available to the application. Hence, any message, including control messages between MPI implementations that can be modulated by the receiver application, is a problem. Examining the modes of MPI point-to-point one observes:

- **Buffered Mode** In this mode the sender application declares a buffer to MPI to manage on the sender's behalf. The receiver can affect the state of the buffer by how it receives messages. Since an error occurs if there is insufficient space in the buffer, the sender receives information from the receiver, therefore the link is not unidirectional. Depending on the implementation there may be still more information available to the sender.

- **Synchronous Mode** In this mode the send operation will only complete sucessfully if a matching receive operation has been posted. This creates information flow back to the sender.

- **Ready Mode** The semantics of ready mode do not require any information about the state of the receiver to be known by the sender. However, ready mode is an optimization opportunity of which the MPI implementation may choose to take advantage. Because its semantics are equivalent to that of standard mode and since the opportunity for optimization may not exist, many implementations simply make ready mode equivalent to standard mode.

- **Standard Mode** The semantics of standard mode point-to-point communication are flexible enough to allow communication without information flow back to the sender, provided infinite buffering at the receiver is assumed. Throttling of standard sends when the receiver is non-responsive induces back communication.

- **Cancelling send requests**[2] The success or failure of cancelling any nonblocking send request depends upon being able to determine the state of the message at the receiver. Unsuccessful cancellation means that the data will be received with no further assistance from the sender. This implies that the message has already been received, or that it has been successfully buffered at the receiver. Successful cancellation of a send operation means that the message has not yet been received [3] by the receiver. Both of these results provide some knowledge as to the state of the receiving process.

A secure implementation will have to either restrict the semantics of MPI data transfer or determine some other way to provide synchronization-like behavior, or accept the resulting backward information flow.

Strict unidirectional links would also impact how new MPI communicators are created, and how barrier synchronizations are performed. Implementations such as MPICH [11] would not work in a strict model. Either collective operations will have to be implemented to obey the directed nature of links (i.e. more tree-like) or some means of suitably restricting the back traffic will have to be found.

Security Identifiers

Consider an MLS environment with two sets of processes. One set runs at the secret level and the other set runs at the top secret level. The secret level processes take data from some source, perform some transformation on that data, and pass the results to the top secret processes.

What should MPI_COMM_WORLD look like for each set of processes? If every process' MPI_COMM_WORLD includes both the secret and top secret processes then the MPI collective operations will break when applied to MPI_COMM_WORLD.

If MPI_COMM_WORLD for each level process includes only the processes at that level then MPI_COMM_WORLD no longer represents "all the processes with which the local process can communicate."

Now extend this example to a pipeline consisting of three subjects, each at a different level. In addition to the question of what MPI_COMM_WORLD should look like for this more complex example, there is a question of how each subject identifies the other two subjects when establishing or initiating communication.

[1] Unidirectional flows appear on other security policies such as the Chinese Wall security policy [3, 17]. Violations of unidirectional flows are considered less of a problem in embedded systems than they are in general purpose systems.

[2] In most implementations, cancelling a send request is a nontrivial operation, partly due to a lack of completeness in definition in the standard. Few implementations support cancelling send requests, and such practice is generally discouraged by implementors.

[3] Or, in some cases, not yet probed.

3.3 Current Implementation

The MPI library (and hence the application) has to be given some way of determining the subject boundaries of the application. The PCT Toolkit provides this information to the MPI library through the use of a subject map table which maps application ranks to subjects. A process need only traverse this table in order to find the other ranks which are in its subject. This mapping makes it possible for the MPI library to determine the communication protocols for each destination rank.

Several mechanisms were considered for the way in which MPI obtains this table. Rather than rebuild the MPI library for each binary image or add an additional function call, the MPI library uses an external storage declaration that is resolved at link time with object code built from source code generated by the PCT Toolkit for each binary image within an application.

The MPI library must also be provided with a capability index for each destination rank which is outside of the sending process' subject. The MPI library again uses an external storage declaration for this mapping of destination rank to capability index which is resolved at link time by code generated by the PCT Toolkit for each binary image within an application. This approach allows the MPI library to be built only once for all of the applications in the system. Using destination rank to access the capability indexes allows for every application to obtain indexes in the same manner. Even though the method used to obtain an index is the same for all applications, the actual data stored in the index array will be different for each subject. This method also eases debugging in an environment that is absent of the PCT Toolkit.

In order to provide a more usable interface to the application programmer, the secure MPI library on PROSE provides an additional communicator, MPI_COMM_SUBJECT, which encompasses all of the ranks within the application that are in the same subject as the calling process. MPI_COMM_SUBJECT is essentially equivalent to the following:

```
MPI_Comm_split( MPI_COMM_WORLD,
                my_subject_id,
                my_rank,
                &MPI_COMM_SUBJECT );
```

Since unidirectional flows may exist between subjects, a multilevel application will no longer rely upon MPI_COMM_WORLD to provide a communication capability to all ranks. However, the meaning and use of MPI_COMM_WORLD will not change for unilevel applications. That is, MPI_COMM_WORLD and MPI_COMM_SUBJECT will be equivalent for single subject applications.

Communication will be established between two subjects by utilizing the MPI_Intercomm_create function or by some analogous MPI extension. The use of MPI_COMM_WORLD may be restricted to only providing a means of mapping from local ranks to global ranks.

At initialization, the library will interpret the subject mapping, set up protocols for each rank in the application, and prepare the process to receive MPI messages. We are currently investigating the protocols needed to implement the various flavors of MPI communication modes. The current implementation of MPI [4] assumes a unilevel application and makes extensive use of acknowledgments and remote memory reads. These mechanisms may not be available in a multilevel application containing subjects at different security levels. The ability to remotely read another process' memory has some very overt security implications. Similarly, mechanisms such as acknowlegdments form the basis upon which a covert channel may be built. Consequently, some of the different MPI send modes may not be possible, or may be restricted in some way in order to minimize the possibility or effectiveness of security related attacks. Once the protocols have been established, a barrier synchronization operation across all of the processes belonging to the same subject will be performed, insuring that each process is ready to receive MPI messages.

Since it is nontrivial to efficiently implement MPI collective operations in a multilevel application with unidirectional flows, collective operations will be disabled on MPI_COMM_WORLD. We are currently assuming MPI-1 functionality where it is illegal to do collective operations using an intercommunicator. We may choose to adopt the MPI-2 specification for intercommunicator collective operations at a later point.

4 Future Directions: Addressing Other Domains

Embedded real-time MLS systems represent only a small portion of all the systems that use MPI. Here are just a few of the issues that arise in other systems.

Is the application code considered trusted by the system? If it is then the application code can direct MPI to provide very fine grain protection of message traffic. One approach being considerd for MPI under this project is to create protected data types analogous to current MPI datatypes, similar to what was done with buffer encoding types in Secure PVM [8]. This approach can be used to provide different protection for control and data messages, see "Control and Data" below and also [9].

If the application code is not trusted, what parts if any of MPI should be protected from it? There are advantages to having MPI provide enforcement of a security policy. For example if the underlying operating system does not

provide cryptographic support for network traffic, or only supports point-to-point messaging, then considerable performance improvements can be realized by having MPI provide cryptographic protection. Allowing parts of MPI to be trusted make it a candidate to apply some innovative cryptographic techniques to both the privacy and authenticity of messages, see "Cryptographic Issues" and "Enforcement". However, MPI implementations are typically not developed using the methods for high assurance systems development.

Process creation in multi-host systems becomes a very complex issue as the diversity of the hosts increases. The negotiating of communication security between the new and existing processes is a more complex problem than what ISAKMP [12] addresses, since the current processes may not have homogeneous capabilities. There are other complexities such as the differing trust relationships between the hosts, differences in internal security functionality of the hosts, and protection requirements that differ in complex ways between runs of the same application code, see "Hetergenous Security" and "Protection Specification".

The performance of MPI applications will strongly influence the level of security/authenticity that can be achieved. It is conceivable that for some applications where very high performance is needed, the security that can be achieved (at costs that are commensurate with the application) may not be high. Such tradeoffs need to be studied and parameterized with respect to applications, policies and environments. Employing innovative cryptographic techniques will hopefully have significant impact on the the level of protection that is affordable.

Heterogeneous Security

Security policies and enforcement mechanisms that are considered need to be designed for environments that include a range of security levels and mechanisms. In some cases programs that are run on a high security may need at times to "borrow" systems across networks that may have a different security profile. It should be possible to run such programs without the need to recompile or reconfigure them. Such a capability needs to have the means for client and server programs to have the power of discovery as to the identity and security of their environments. Once a program has determined the security requirements and security capabilities of its environment, and the needed security related "clearances" at the remote objects, it can invoke the appropriate functions to protect, prove or verify identity.

Control and Data

It is necessary to consider the different needs for security and authenticity of control information and data that will flow across the networks. Each will have its own security needs and the mandated enforcement mechanisms. For some applications, the data privacy will be of utmost importance, while in some cases the security / authenticity of control information will also be important. The MPI interface provides an ideal environment to consider the separation of control information from the data. Similar to telephony, where voice and control information are routed on different lines, we consider the data and control flowing to a process (client or server) via entirely different paths. In fact, the separation of control and data paths will be necessary to provide different levels of protection. We then need to consider the binding of data to control. New primitives may have to be considered for MPI to implement this type of separation.

Cryptographic Issues

In considering enforcement mechanisms, it is necessary to consider the performance of such things as encryption hardware that is needed for privacy and verification of authenticity. Depending on the security profile of the workstations and servers, it is conceivable that different cryptographic hardware will be needed. It may be feasible to consider tamper resistant hardware in some applications where symmetric encryption can be used for authentication instead of a public key cryptosystem [7, 5]. Such options can significantly improve the performance of the systems. In some systems, tamper resistance may have the same qualitative enforcement properties as a public key system such as the RSA.

The privacy and authentication issues raised by MPI will require the study of different cryptographic techniques that may be needed for the many applications and security requirements. For broadcast messages, it may be feasible to use cipher-buses (where multiple messages are encrypted into a common ciphertext) to be sent to multiple clients, with each client decrypting what it is capable of decrypting [6]. The attraction of such systems is that individual encrypted messages do not even have distinguishable boundaries. This may make it difficult for detection of such messages for possible attacks.

Enforcement

In certain security policies, ordinary applications are trusted not to attempt to bypass security controls of the system or otherwise weaken the security of the system. In other policies, this is not the case. If th application is not trusted, then the issue of what role, if any, MPI can play in *enforcing* a security policy is of interest. This enforcement may include restricting where an application may send a message or what cryptographic protection must be applied to messages to particular destinations.

Such enforcement within the current MPI structure will be difficult. In order for a mechanism within MPI to be able to restrict the action of an application it must be tamper proof, un-bypassable, and always enforced. Typically, enforcement is provided with some hardware help. Modern microprocessors provide two crucial hardware protections: supervisor mode and address space separation. MPI is currently implemented as a user-level, compile or runtime

linked library. Since it inherits the user's privileges and runs in the same address space, it cannot use the hardware support to directly prevent the user from taking actions, including disabling controls within MPI.

Versions of Secure MPI that do not trust their applications and need to depend on SMPI for enforcement will thus have to explore alternate approaches. One approach is to build the applications using special code generation techniques [19] combined with strong integrity controls being used on the result. This approach has the advantage of maintaining the user-level nature of MPI but risks degrading performance.

Other approaches include eliciting the aid of the operating system kernel, trusted servers, and/or special message passing hardware. By exploring these options one can envision versions of MPI with portions of the "library" in a protected unbypassable portion of the system that is always invoked properly and the portions of this SMPI that are not security critical residing in user space.

Such relationships are not uncommon, an example is the relationship between the OS and user-level code that exists in Puma[18] portals. In Puma portals, message passing responsibility is shared between user-level libraries which set up the required portal structures and supervisor-level OS kernel functions which perform the data transfer and which can enforce access controls.

A similar approach to this "split MPI" is by implementing security critical portions of SMPI within a Myrinet[2] LANAI control program.

Protection Specification.
Specifing how to protect communications is a complex task; for example:

- Traffic moving between the the nodes of a MPP machine would likely require no cryptographic protection. On a local network, the same traffic may require a certain level of protection, and the same traffic moving accross the Internet may require still another (higher) level of protection.

- The protection required will vary depending on the problem being solved. While the application code may not change, only the data being used and and generated requires a change in protection level. Another possibility is that the algorithm (code) needs to be protected while the data does not, or the data only requires minimal protection.

- Certain portions of an application may require different levels of protection than others. For example, a system that is composed of a proprietary subsytem and other less sensitive subsystems. A simulation of such a system may need to strongly protect only what pertains directly to the proprietary subsystem. It is assumed that an adversary could not cost-effectively deduce the proprietary information from the other portions of the simulation.

Protection specification can be combined with the discovery process discussed earlier to arrange for the proper cryptographic techniques to be used. [4]

There are still other issues that need to be addressed in developing secure MPI variants such as key management and how best to provide security for MPI applications not written for SMPI

5 MPI-2 and MPI Real-time

The recently approved MPI-2 document extends the functionality of MPI in many different areas, providing an interface for the following:

- dynamic process spawning
- dynamic process attachment
- one sided communications
- extended collective operations
- parallel I/O

These additional features have security implications which need to be addressed, not only to discover the possible places in which security features need to exist, but also to identify possible mechanisms for implementing these features. For example, dynamic process spawning will most likely require some form of authentication, which may make use of the `MPI_INFO` mechanism or the `mpiexec` process startup mechanism.

The ongoing MPI Real-time (MPIRT) effort is an attempt to leverage the functionality of MPI in real-time and embedded systems. Unlike previous MPI standards, MPIRT's basic building block for communication is a channel, which is a unidirectional communication pipe upon which a quality of service can be obtained. Again, further investigation is needed to determine the security issues and available mechanisms for conveying security information associated with these channels.

6 Remarks

The reader needs to be aware that there are limits to how secure a system can be simply by enhancing MPI. Other aspects of the system, in particular the operating system and proper administration have a large role to play. In addition, the security policy enforced by the system must be

[4] This specification can go further, it may include mininum Common Cirteria ratings for systems to be used for a computation.

the correct one for the needs of the system's users and the (threat) environment in which it operates. The environment in which a system operates is a constantly evolving one. While many users only need to worry about the prevalent threats of today, other users and system developers need to worry about tomorrow's threats as well.

7 Acknowledgements

The authors would like to acknowledge the many contributions of the members of the JRTOS team from Hughes Aircraft Company, Intel Corporation, and Sandia National Laboratories.

References

[1] J. P. Alstad et al. The role of system build in trusted embedded systems. In *Proceedings of the 13th National Computer Security Conference*, volume 1, Oct. 1990.

[2] N. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet-a gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, February 1995.

[3] D. F. Brewer and M. J. Nash. The Chinese wall security policy. In *Proceedings 1989 IEEE Computer Society Symposium on Security and Privacy*, pages 206–214, May 1989.

[4] R. Brightwell and L. Shuler. Design and implementation of MPI on Puma portals. In *Proceedings of the Second MPI Developer's Conference*, pages 18–25, July 1996.

[5] G. I. Davida and B. J. Matt. Arbitration in tamper proof systems. In *Proceedings Advances in Cryptology - Crypto '87*, pages 216–222, Aug. 1987.

[6] G. I. Davida, D. L. Wells, and J. B. Kam. A database encryption system with subkeys. *ACM Transaction on Data Bases*, 6(2), 1981.

[7] Y. Desmedt and J.-J. Quisquater. Public-key systems based on the difficulty of tampering. In *Proceedings Advances in Cryptology - Crypto '86*, pages 111–117, Aug. 1986.

[8] T. Dunigan and N. Venugopal. Secure PVM. Technical Report ORNL/TM-13203, Oak Ridge National Laboratory, August 1996.

[9] I. Foster, N. T. Karonis, C. Kesselman, G. Koeing, and S. Tuecke. *A Secure Communications Infrastructure for High-Performance Distributed Computing*, 1996. http://www.mcs.anl.gov/zipper.

[10] R. Gotfried and T. Woodall. PCT toolkit: An implementation of the system build approach. In *Proceedings of the 19th National Information System Security Conference*, October 1996.

[11] W. Gropp, E. Lusk, and A. Skjellum. *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*, July 1996. http://www.mcs.anl.gov/mpi/mpicharticle/paper.ps.

[12] D. Maughan, M. Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Management Protocol (ISAKMP)*, February 1997. ftp://ietf.org/internet-drafts/draft-ietf-ipsec-isakmp-07.ps.

[13] Message Passing Interface Forum. *The Message Passing Interface Standard*, November 1995. http://www.mcs.anl.gov/mpi/mpi-report/mpi-report.html.

[14] Message Passing Interface Forum. *MPI-2: Extensions to the Message Passing Interface*, October 1996. http://www.cs.wisc.edu/~ lederman/mpi2/mpi2-report.ps.Z.

[15] H. Nag, R. Gotfried, D. Greenberg, C. Kim, B. Maccabe, T. M. Stallcup, G. Ladd, L. Shuler, S. Wheat, and D. van Dresser. PROSE: Parallel Real-time Operating system for Secure Environments. In *Intel Supercomputing '96 Proceedings*, June 1996.

[16] National Computer Security Center, DoD. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985.

[17] R. Sandhu. A lattice interpetation of the Chinese Wall Policy. In *Proceedings 15th NIST-NCSC National Cimputer Security Conference*, pages 329–339, Oct. 1992.

[18] L. Shuler, C. Jong, R. Riesen, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup. The Puma operating system for massively parallel computers. In *Proceeding of the 1995 Intel Supercomputer User's Group Conference*. Intel Supercomputer User's Group, 1995.

[19] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. In *Proceedings 14th ACM Symposium on Operating Systems Principles*, pages 203–216, Dec. 1993.