Network Fault Tolerance in Open MPI

Galen M. Shipman[1], Richard L. Graham[2], and George Bosilca[3]

[1] Advanced Computing Laboratory, Los Alamos National Laboratory
gshipman@lanl.gov
[2] National Center for Computational Sciences, Oak Ridge National Laboratory
rlgraham@ornl.gov
[3] University of Tennessee, Dept. of Computer Science
bosilca@cs.utk.edu

Abstract. High Performance Computing (HPC) systems are rapidly growing in size and complexity. As a result, transient and persistent network failures can occur on the time scale of application run times, reducing the productive utilization of these systems. The ubiquitous network protocol used to deal with such failures is TCP/IP, however, available implementations of this protocol provide unacceptable performance for HPC system users, and do not provide the high bandwidth, low latency communications of modern interconnects. This paper describes methods used to provide protection against several network errors such as dropped packets, corrupt packets, and loss of network interfaces while maintaining high-performance communications. Micro-benchmark experiments using vendor supplied TCP/IP and O/S bypass low-level communications stacks over InfiniBand and Myrinet are used to demonstrate the high-performance characteristics of our protocol. The NAS Parallel Benchmarks are used to demonstrate the scalability and the minimal performance impact of this protocol. Communication level micro-benchmarks show that providing higher data reliability decreases bandwidth by up to 30% relative to unprotected communications, but provides performance improvements of a factor of four over TCP/IP running over InfiniBand DDR. In addition, application level benchmarks (communication/computation) show virtually no impact of the data reliability protocol on overall run-time.

1 Introduction

The ever increasing complexity and scale of HPC systems increases the likelihood of hardware and software component failure in these systems. The use of commodity (or near commodity) off-the-shelf components to build many such systems further aggravates the problem, as these are often not engineered to provide end-to-end hardware reliability; either ignoring such reliability issues or leaving it to software layers to provide. The ubiquitous software solution for end-to-end reliability is provided by TCP/IP communications stacks. The performance provided by such commonly available stacks does not meet the requirements of the HPC community, providing only a small fraction of the communications performance afforded by the networking hardware.

A.-M. Kermarrec, L. Bougé, and T. Priol (Eds.): Euro-Par 2007, LNCS 4641, pp. 868–878, 2007.

While HPC system failures occur in a variety of ways, this paper focuses on a software architecture aimed at detecting and correcting failures in network data transmission. The goal of this design is to provide end-to-end protection against such failures, while providing communication performance commensurate with that of the underlying hardware. These failures may occur in a number of layers including software stacks, firmware or in hardware. They may include transient failures, such as Network Interface Card (NIC) resets, or more permanent failures, such as failed NICs. These failures may result from the normal statistical failure rate associated with large component counts, or may be the result of software or hardware defects, which may be fixed over time. Therefore addressing these issues is required for effective system utilization over the lifetime of these systems.

As costs are incurred when providing these fault tolerant features, the Modular Component Architecture (MCA) [1] of Open MPI is used to provide these as run-time selectable options. When these features are not selected, there is no impact on the the default high-performance configuration of Open MPI.

The remainder of this paper is organized as follows: Section 2 presents a brief overview of previous work. Next, Section 3 discusses the network fault tolerance architecture in Open MPI. Results are discussed in Section 4. Conclusions are discussed in Section 5.

2 Background

There have been several previous efforts to deal with network failure in a manner transparent to the calling application. The TCP/IP [2] stack deals with both transient data corruption, as well as with transient and permanent network failures. However, since TCP/IP is a general purpose network stack, designed to deal with a wide variety of network failures, including lossy data transmissions, flow control, and congestion control issues, these implementations do not provide the level of performance required by HPC applications. Network communication on HPC systems is normally generated only by the applications using these systems, and as such suffer little interference from system services or other applications. In addition, the networks used in these systems often possess a high degree of reliability with low error rates. These operating environments allow for reliability protocols which provide higher overall performance than general purpose reliability protocols which often provide unnecessary (often costly) features.

There have also been several attempts to provide different aspects of network fault tolerance specific to HPC systems. One of the goals of the LA-MPI project [3,4] was to provide reliable network communications. It uses timers on ACKs to detect dropped packets and either the TCP/IP checksum or a Cyclic Redundancy Check (CRC) to detect corrupted packets. The work presented in this paper draws upon prior work in LA-MPI but is entirely new in terms of software and protocol providing better performance while also adding new features such as network fail-over and protocol tuning for specific operating environments. The VMI project [5] also provides a way to deal with network errors, and recently, the MVAPICH project implemented network fail-over using the uDAPL interface [6].

3 Open MPI - Network Fault Tolerance

3.1 Open MPI's Point-To-Point Architecture

Open MPI's point-to-point architecture has been described in great detail elsewhere [7], and will be described very briefly in this section. Figure 1 provides a graphical depiction of this design.

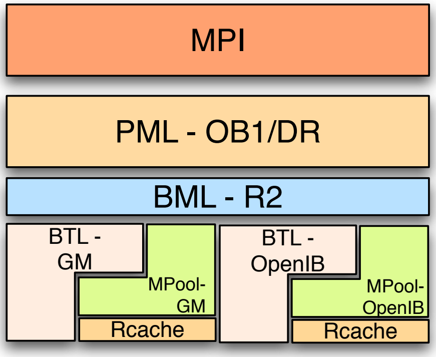


Fig. 1. Open MPI's Layered Architecture

Open MPI uses a layered design, with the aid of the the Modular Component Architecture, to achieve the flexibility in the features this code base offers. The MPI layer interfaces with the point-to-point communications implementation of Open MPI via the Point-To-Point Management Layer (PML). Currently, there are multiple PMLs supported, such as the OB1 PML [8] and the Data Reliability PML (DR), described in this paper.

The OB1 PML is aimed at providing the best point-to-point communications performance possible making use of all available communications resources, and the DR PML provides fault tolerance for this type of communications. Both PML's use the Byte Transfer Layer (BTL), the BTL-Mangement-Layer (BML), the Memory Pool (MPool), and optionally the Registration Cache (RCache) components in implementing these communications protocols.

3.2 Data Reliability DR Overview

Open MPI provides transparent user-level reliability over a variety of networks and network APIs. This user-level reliability is encapsulated in a single Modular Component Architecture component, PML DR or just DR. DR implements the point-to-point semantics of MPI while providing several network fault tolerance features. DR provides protection from a number of failure scenarios:

- Dropped Data
- Corrupted Data
- Catastrophic NIC Failure (Fail-over When other Communication Paths Exist)
- Network Agnostic Protection via Local Completion Watchdog Timers and ACK based Timeouts
- Network Specific Protection via Registered Error Handlers

Similar in functionality to PML OB1 [8], DR also makes use of the BTL (byte transfer layer) which abstracts the underlying network API in a uniform manner. This uniform network abstraction API allows DR to provide network fault tolerance in a network agnostic fashion without relying on other network abstraction libraries. The BTL abstraction is high-performance by design and when used with PML OB1 performance is similar to other MPI libraries. In using the high-performance BTLs the additional costs of reliability are isolated to the PML DR component.

3.3 VFRAG Protocol

A unique feature of PML DR is its user-level reliability protocol. A key component of the protocol is the Vector of Fragments (VFRAG). The VFRAG acts as a unit of acknowledgment and allows selective retransmission in a straight forward manner. Each MPI message is divided into N virtual fragments. Each virtual fragment is made up of 64 smaller fragments. The total size of the VFRAG is therefore $64 * S_{max}$ where S_{max} is runtime configurable. For example, given a MPI level message of size $4MB$ and $S_{max} = 16K$ the number of VFRAGS would be $4MB/(16K * 64) = 4$. Figure 2 illustrates the VFRAG structure.

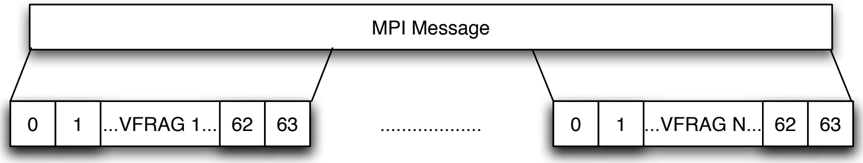


Fig. 2. VFRAG Layout

This reliability protocol begins by allocating (via a free-list) a VFRAG descriptor. The VFRAG descriptor contains two timers, a local completion timer and a remote ACK timer. Upon scheduling fragment 0 of the current VFRAG the local completion timer is initialized and started and the number of pending fragments within the VFRAG is set to 1. Subsequent fragments within the same VFRAG are also scheduled and the number of pending fragments is incremented. As notification is received from the BTL that the fragment was sent and local completion occurred, the number of pending fragments is decremented and the timer is reset with a new timeout (as long as there are pending fragments).

The local completion timer gives some indication of VFRAG progression though it is not definitive. Relying solely on local completion as indication of message progression places reliability of completion semantics with the network interface. In order to protect the VFRAG from unreliable network completion semantics an ACK timer is set when the last fragment of the VFRAG is scheduled for transmission. The ACK message from the receiver contains a bit-mask indicating which of the 64 fragments of the VFRAG were delivered successfully. This two-stage timeout reduces the overhead of reliability by aggregating acknowledgments. In addition, recovery is optimized through local completion timeouts and selective retransmission via the bit-mask ACK.

In addition to the local watchdog timer and the remote ACK timer PML DR can also recover from asynchronous errors from the BTL. During initialization DR registers an asynchronous error handler with the BTL. This error handler can take appropriate action such as failing over traffic to another BTL. Fail-over can also occur based on tunable retransmit levels on a per VFRAG basis. This is allowing fail-overs to occur either as a result of a network reported error or timeouts of the local watchdog timer or remote ACK timer. This is a major difference in contrast of relying solely on network reported errors as described in [6].

This protocol provides several benefits. CRC/Checksum size is configurable based on S_{max}. That is each fragment of size S_{max} carries an associated CRC/ Checksum in its fragment header. ACK aggregation is tunable based on the total size of the VFRAG. Selective retransmission is provided by the bit-mask ACK. Along with these benefits there are additional costs including:

1. Latency increased due to specific acknowledgment
2. Bandwidth decreased due to additional protocol overhead
3. CPU Availability decreased due to additional protocol overhead

In recognition of these costs, DR takes advantage of various MPI semantics to limit protocol impact. For example, DR can mark MPI completion of a message as soon as it is buffered instead of waiting for remote ACK of the message thereby hiding some of the additional protocol costs of reliability. In addition, the performance impact can be tuned by selecting the recovery cost/performance ratio appropriate to a given operating environment. For example, if the operating environment includes a highly reliable network with few failures S_{max} can be increased such that each ACK protects a larger amount of data. This increases recovery costs while decreasing the cost of reliability. CSUM/CRCs can be enabled/disabled on a per BTL or global basis thereby allowing the user or system administrator to choose Main-memory to Main-memory protection at a fine or coarse granularity.

While PML DR provides protection from a number of different network failure scenarios, it does not protect against every eventuality. DMA operations initiated by the NIC which corrupt the target address cannot be protected against with DR as the DMA may corrupt random memory locations on the host. Network stack/driver errors can sometimes result in kernel panics or processes hanging in uninterruptible sleep from which the user level process cannot recover.

3.4 Data-Type Engine

The responsibility to pack, unpack and compute the checksum belongs to the Open MPI data-type engine. A full description of the internals of the data-type engine is out of the scope of this paper, thus we will give only a brief description of the data-type engine mechanisms involved in DR.

In order to allow the PML to fragment a message the data-type engine has been modified to work on segments, using another entity called *convertor*. All operations (packing and unpacking) are limited by the number of bytes requested by the PML. In order to satisfy this requirement, the data-type engine keeps track of the last state of the current operation (pack or unpack). Once an operation is completed, the current internal state of the *convertor* is saved. The next operation will continue from this saved position. Therefore, in Open MPI, there is no need to pack the full user data before a send, nor to unpack it in one operation on the receiver side. This approach allows Open MPI to apply different optimization to the packing and unpacking process, as well as to the checksum computation. As an example, we can limit any operation to the amount of data that is cache friendly on the current architecture. Another optimization, is the pipeline that is created using the pack, unpack and the network communication for each of the fragments as illustrated in Figure 3.

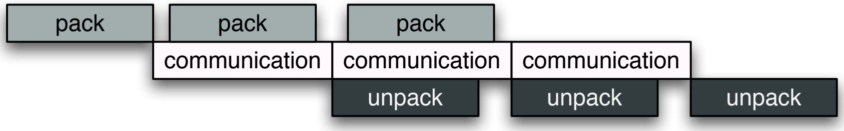


Fig. 3. Overlapping packing/unpacking with the communications using the data-type engine

The default checksum computation is a fast 32 bit algorithm. This algorithm has been modified to work on the same principle as the data-type engine, i.e. in a segmented way. When the checksum is enabled and the network device requires memory copies, the memory copies and the checksum computations are interleaved in order to reduce cache pollution. If there is no need for memory copies the checksum is computed directly on the user buffer.

4 Results

4.1 Experimental Setup

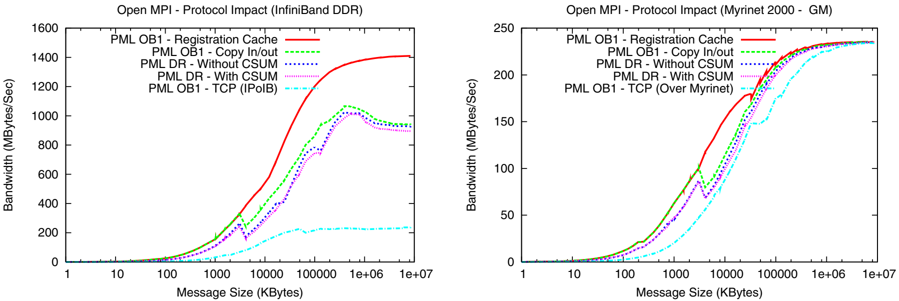
The NAS Parallel benchmark (NPB) [9] were run on a 1290 node cluster (4 segments of 258 nodes). Each node has 2 Single Core AMD Opteron 252 processors, 8 GBytes of memory, 1 Mellanox InfiniBand MT25204 InfiniHost III Ex adaptors connected via a Voltaire SDR switch. All other experiments were performed on a

4 Node test cluster. Each node has 2 Dual Core AMD Opteron 270 processors, 4 GBytes of memory, 2 Mellanox InfiniBand MT25208 InfiniHost III Ex adaptors each on a dedicated PCI-Express 16X bus and 1 Myricom Myrinet 2000 PCI-X "D card" NIC on a 133 MHz PCI-X bus. Myricom adaptors are connected via a Myricom 2000 switch. Mellanox adaptors are connected via a DDR (double data rate) Silverstorm switch. Each node was installed with Fedora Core 5, Open MPI Trunk Revision 12736, OFED 1.1, and GM 2.1.26.

4.2 Results and Analysis

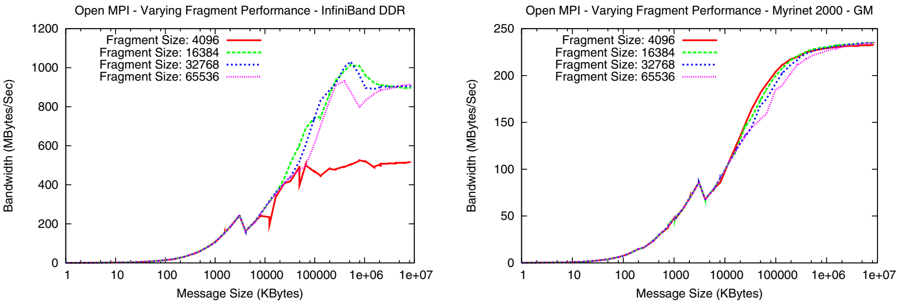
To examine the performance impact of our data reliability protocol we used the NetPipe [10] benchmark as illustrated in Figure 4(a) and Figure 4(b). Four different protocols were examined, Single RDMA GET with registration cache, PML OB1 Copy In/Out using send/recv, PML DR with checksums and PML DR without checksums. On both InfiniBand and Myrinet 2000, the highest performance was obtained using the Single RDMA protocol due to buffer reuse in the NetPipe benchmark and the high performance of RDMA. On InfiniBand PML DR bandwidth performance is substantially lower than the single RDMA protocol although most of this performance difference is a result of using copy in/out protocols, this is not as apparent over Myrinet as the memory bandwidth surpasses the network bandwidth. When we compare the performance of PML DR with the high-performance PML OB1 using copy in/out we see that DR incurs a small overhead due to protocol processing and a slightly higher overhead due to checksum costs on both InfiniBand and Myrinet 2000. The small relative impact of checksums can be attributed to an integration of the checksum/crc with the data-type engine. Of note in the InfiniBand results is the performance degradation at larger message sizes, this is a result of a memory bandwidth bottleneck on this particular architecture/network. When compared to TCP/IP over InfiniBand (using the high-performance IPoIB stack), PML DR provides a substantial performance increase throughout the bandwidth curve. Even in the bandwidth limited Myrinet 2000 the performance of PML DR surpasses TCP/IP over this interconnect.

As discussed earlier PML DR performance can be adapted for a given operating environment. One such adaptation is varying the size of each fragment. Larger fragments provide better performance (to a point) while smaller fragments are better protected by their associated checksum. Figures 5(a) and 5(b) demonstrates the effect of changing the fragment size. While a fragment size of 4096 is better protected by its checksum, the performance degrades over InfiniBand as the upper layer cannot effectively keep the network pipe full. Performance increases as the fragment size increases up to 16K which gives the best performance on InfiniBand in this environment at the expense of a less effective checksum. Recovery costs are also higher at 16K fragment sizes as the retransmission of dropped or corrupted fragments is on entire fragments. Changing the size of the fragments when running over Myrinet 2000 has little impact, again due to the network bottleneck relative to memory bandwidth.



(a) InfiniBand (b) Myrinet 2000

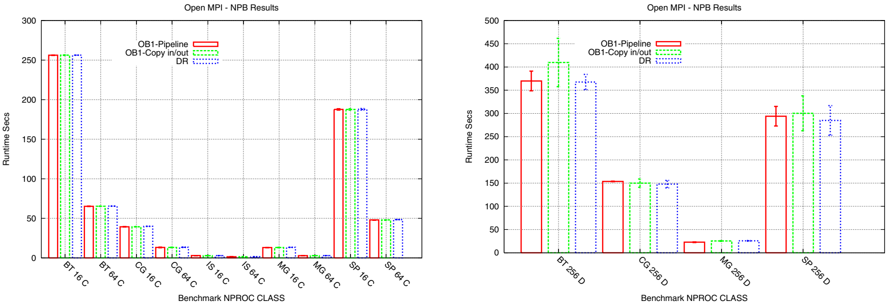
Fig. 4. PML Protocol Performance



(a) InfiniBand (b) Myrinet 2000

Fig. 5. Variable Fragment Size - Performance Impact

In order to better assess the performance impact of DR in medium scale clusters the NAS Parallel benchmark (NPB) were used. The BT, CG, MG and SP benchmarks were run with 16 and 64 processors with a problem size of class C (second to largest) and 256 processors with a problem size of class D (the largest). The IS benchmark was only run at 16 and 64 processors with a problem size of class C because class D is not available for IS. Each benchmark was run 3 times and the average runtime is shown as a bar with an additional error bar centered at the top indicating the standard deviation. As illustrated in Figure 6(a) the additional protocol overhead of DR has very little impact on more realistic benchmarks at the class C size with a very small standard deviation (almost not visible in this Figure). Figure 6(b) illustrates the impact on class D size with 256 processors. For each of these runs the performance differences between the 3 protocols is within the standard deviation. The higher standard deviation is expected as the runtime and problem size of class D is more likely to be impacted by memory caching effects and network congestion. These benchmarks indicate that added benefits of DR may come at little or no cost to a wide variety of parallel problems.



(a) 16,64 Processors, Class C

(b) 256 Processors, Class D

Fig. 6. NPB Benchmarks - InfiniBand

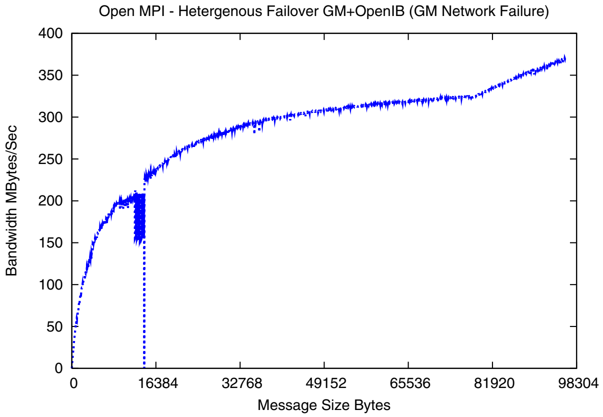


Fig. 7. DR - Failover (Myrinet to InfiniBand)

In addition to network retransmission of dropped and corrupted fragments, PML DR provides network fail-over when more than one path exists to a given peer. In this experiment Myrinet and InfiniBand are both used for message scheduling. Each fragment of the message can be scheduled on either interconnect with the number of fragments scheduled to each based on its relative bandwidth. In Figure 7 we demonstrate the effect of failover on bandwidth. After the Myrinet network on one of the hosts is disconnected from the switch, bandwidth becomes more sporadic as data is rescheduled on timeouts over the InfiniBand interface. Once the number of retransmissions exceeds a configurable threshold, Myrinet BTL is disabled and from that point on all data is scheduled over the InfiniBand interface. Similar results occur when InfiniBand is disconnected. The mechanism is somehow different as InfiniBand will deliver an asynchronous error to PML DR from the OpenIB BTL as a result of a completion queue error. DR responds to the asynchronous error rather than waiting for the exceeding of the retransmission threshold. The ability to detect network failure outside of asynchronous errors allows DR to respond to failure of a variety of networks and network API semantics.

5 Conclusion

As the use of commodity or "near commodity" networks continues to increase in large scale HPC systems, and the size of these systems continues to grow, robust user level libraries can enhance the reliability of network communication. While theoretical error rates remain relatively low for virtually all high-performance networking technologies, some larger scale installations have also shown that hardware, software, or firmware bugs are often long lived and transient. Facilitating useful science throughout the lifetime of these systems is important and may benefit from reliable high-performance network techniques. In this work we have described methods of dealing with network errors that help improve the reliability of network communications, with minimal impact on application performance, thus providing a means to improve the effectiveness of HPC scale simulation clusters.

Acknowledgements. Los Alamos National Laboratory is operated by Los Alamos National Security, LLC for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC52-06NA25396. Project support was provided through ASCI/PSE and the Los Alamos Computer Science Institute, LA-UR-06-8492. This work is funded by subcontract #R7B127 from Rice University under prime contract #12783-001-05 49.

References

1. Squyres, J.M., Lumsdaine, A.: The component architecture of open MPI: Enabling third-party collective algorithms. In: Getov, V., Kielmann, T. (eds.) Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications, St. Malo, France, pp. 167–185. Springer, Heidelberg (2004)
2. RFC793: Transmission control protocol. DARPA Internet Program Protocol Specification (September 1981)
3. Aulwes, R.T., Daniel, D.J., Desai, N.N., Graham, R.L., Risinger, L.D., Sukalski, M.W., Taylor, M.A., Woodall, T.S.: Architecture of LA-MPI, a network-fault-tolerant MPI. In: Los Alamos report LA-UR-03-0939, Proceedings of IPDPS (2004)
4. Graham, R.L., Choi, S.E., Daniel, D.J., Desai, N.N., Minnich, R.G., Rasmussen, C.E., Risinger, L.D., Sukalksi, M.W.: A network-failure-tolerant message-passing system for terascale clusters. International Journal of Parallel Programming 31(4) (August 2003)
5. Pakin, S., Pant, A.: VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management. In: Proceedings of The 8th International Symposium on High Performance Computer Architecture (HPCA-8), Cambridge, MA (February 2002)

6. Vishnu, A., Gupta, P., Mamidala, A.R., Panda, D.K.: A software based approach for providing network fault tolerance in clusters with udapl interface: Mpi level design and performance evaluation. In: Proceedings of 2006 International Conference for High Performance Computing, Networking, Storage and Analysis (2006)
7. Graham, R.L., Barrett, B.W., Shipman, G.M., Woodall, T.S., Bosilca, G.: Open mpi: A high performance, flexible implementation of mpi point-to-point communications. Parallel Processing Letters (accepted, January 2007)
8. Shipman, G., Woodall, T., Graham, R., Maccabe, A., Bridges, P.: Infiniband scalability in open mpi. In: Proceedings, 20th IEEE International Parallel & Distributed Processing Symposium, IEEE Computer Society Press, Los Alamitos (2006)
9. Bailey, B., Barton, B.,Carter, D., Fatoohi, F., Frederickson, L., Schreiber, S., Venkatakrishnan, W.: NAS parallel benchmarks (1994)
10. Snell, Q., Mikler, A., Gustafson, J.: NetPIPE: A Network Protocol Independent Performace Evaluator. In: IASTED International Conference on Intelligent Information Management and Systems (June 1996)