Performance evaluation of DECK combining multithreading and communication on Myrinet and SCI clusters

7 authors, including:

Fausto Richetti Blanco
Universidade Federal do Rio Grande do Sul
4 PUBLICATIONS **21** CITATIONS

SEE PROFILE

Marcos Ennes Barreto
The London School of Economics and Political Science
85 PUBLICATIONS **851** CITATIONS

SEE PROFILE

Philippe Olivier Alexandre. Navaux
Universidade Federal do Rio Grande do Sul
443 PUBLICATIONS **2,405** CITATIONS

SEE PROFILE

Performance evaluation of DECK combining multithreading and communication on Myrinet and SCI clusters*

César De Rose‡ Fábio Oliveira¶ Fausto Blanco‡ Marcos Barreto¶
Philippe Navaux¶ Rafael Ávila¶ Tiago Ferreto‡

¶ Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil

‡ High Performance Research Centre (CPAD)
Catholic University of Rio Grande do Sul
Porto Alegre, Brazil

Abstract

This paper presents a performance evaluation of DECK (Distributed Execution and Communication Kernel), a multithreaded parallel programming environment for clusters of SMPs, with the parallel implementation of the classical Mandelbrot fractal generation and Laplace's Equation algorithms. The applications have been run on Myrinet and SCI clusters and the results are compared to corresponding MPI implementations. The comparison shows that DECK is able to achieve very good performance when multithreading is combined with communication, without the need of multiple processes on a single machine.

Keywords: cluster computing, multithreading, parallel programming environment, Myrinet, SCI.

1. Introduction and context

Current high-speed communication technologies such as Myrinet [3] and SCI [8] are usually interfaced by low-level programming libraries such as GM [5], BIP [13] and SISCI [4], which have been introduced in order to efficiently exploit the low latency and high throughput offered by those technologies. However, such libraries present a rather complex API and are not targeted at the end user, but instead they serve as a basis for the implementation of higher level environments such as MPI [10]. Still, one of the most common characteristics of clusters, which is the availability of SMP nodes, is usually not taken advantage of, and frequently not even supported, given that many MPI implementations are not even thread-safe.

Following this idea, we have designed and implemented DECK (Distributed Execution and Communication Kernel) [2], an environment for parallel programming on SMP clusters. The main goal of DECK is to try to efficiently integrate multithreading and communication in a single environment. In this paper we present a performance evaluation of two flavours of DECK, running on Myrinet and SCI clusters, with the parallel implementation of two algorithms usually present in the

*Results for Myrinet have been measured at the High Performance Research Centre (CPAD), PUCRS/HP; results for SCI have been measured at the Institute of Informatics, UFRGS

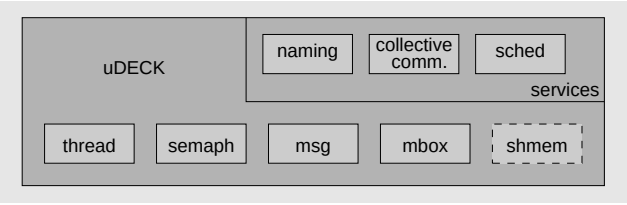

Figure 1. The internal structure of DECK.

HPC community, namely the Mandelbrot fractal generation [9] and the solving of Laplace's Equation [12]. In addition, we compare the obtained results to those of equivalent implementations executed with versions of MPI for both communication technologies.

The paper is structured as follows: in Section 2 we present some background information with an overview of DECK and its implementations for Myrinet and SCI; in Section 3 we describe the applications, their implementation on DECK and MPI and analyse the obtained results; finally, Section 4 brings the authors' conclusions and future directions.

2. The DECK environment

The API of DECK is conceptually object-oriented, providing basic abstractions for parallel applications, such as threads and mail boxes, and more elaborate services like naming and collective communication. Figure 1 shows the internal structure of DECK. The bottom layer is called μDECK, being responsible for the basic abstractions: threads, semaphores, messages and mail boxes. The upper layer of DECK is a service layer, whose services can be chosen at compilation time.

Two implementations of DECK are currently available, for both Myrinet and SCI communication technologies. Multithreading is implemented on both by making use of standard POSIX Threads [7] calls. The communication issues are described next.

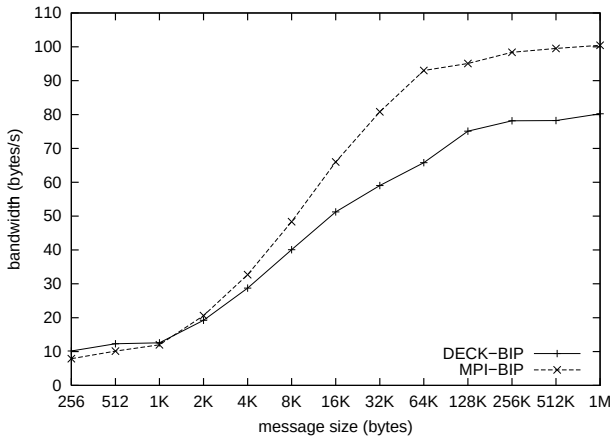

Figure 2. Bandwidth of DECK and MPI-BIP on Myrinet.

2.1. DECK/BIP

For the implementation of DECK on Myrinet [1] we have used the BIP library, mainly because of its performance and simple API. BIP provides basic point-to-point communication primitives (asynchronous *send()* and *recv()*) on Myrinet, with the constraint that the exchange of large messages (in BIP terms) must follow a *rendez-vous* semantics, which means that a given *recv()* must always be posted before the corresponding *send()*. In order to accomplish that in DECK, we have used a handshaking protocol where the sender makes use of small request messages before sending a large message. Such requests are handled, on the receiver side, by a dedicated thread, the *rv-daemon*, created at initialisation time.

This additional thread may have a significant impact on communication performance depending on the communication pattern. Figure 2 shows a raw performance evaluation of DECK and MPI-BIP [15], an MPI implementation on top of BIP. The graph shows the influence of the rv-daemon for messages larger than 1K, where the performance of DECK is around only 80% of that of MPI.

When a thread posts a short message, this one is sent directly to the destination mailbox, since BIP is able to store it in its internal buffers. In the case of large messages, it sends a short message to the rv-daemon on the receiver side. This thread verifies whether there is a buffer ready for receiving the large message and responds. The sender thread keeps asking the rv-daemon on the receiving side until it gets a positive response, then finally sends the message.

A prior raw performance evaluation [1] has revealed that DECK/BIP is able to achieve 7μs of latency on our Myrinet cluster.

2.2. DECK/SCI

The SCI version of DECK [11] makes use of the SISCI API. SISCI—Software Infrastructure for SCI—is a specification of standard primitives for SCI programming, based on the shared segment notion, proposed by a group of partners from both the academia and the industry. One implementation of SISCI is available in the SSP (Scali Software Platform), a software package distributed with Scali Wulfkits [14], upon which our SCI cluster is based.

Three communication protocols were developed for the implementation of DECK on SCI, in order to guarantee low latency and minimal overhead for small messages, as well as high bandwidth for large ones.

The first protocol, specialised in exchanging small messages, achieves low latency by integrating communication and signalling into the same SCI packet. Regardless of the real message size, this protocol always sends a single 64-bytes SCI packet, where the last byte contains an identifier that allows the receiver to get notified about the message arrival. By doing so, it is possible to achieve a minimal latency of 4.66 μs. This protocol was implemented for exchanging messages ranging from 0 to 62 bytes.

In order to allow greater messages to be sent, it was necessary to implement a general-purpose protocol. Such protocol is more generic, being capable of dealing with messages of virtually arbitrary sizes. Besides the user message, it sends an additional SCI packet for signalling the message arrival. However, this general-purpose protocol limits the maximum achievable bandwidth to a value really below the capabilities of the SCI technology, as the message is copied to the user buffer only after it has been totally transmitted over the network. This behaviour is particularly undesirable for large messages.

Hence, we had to devise another protocol, given that the main purpose of DECK/SCI is to approach the hardware limits. The third protocol employs a zero-copy communication mechanism, in the sense that the message is directly sent to the user buffer, without any additional copy of the message from SCI shared memory to local memory. So, the sender first asks the receiver what is the address of the user buffer and waits for the response. Then, the receiver informs the address where the message is expected to be sent to and, accordingly, the sender transmits the message through the SCI network. Finally, the sender tells the receiver that the communication is finished. Such protocol is clearly synchronous, since it requires a handshaking between sender e receiver. The shared-memory nature of SCI makes possible to devise such a scheme, avoiding unnecessary and expensive additional memory copies.

The designed mechanism for zero-copy is able to boost the maximum achievable bandwidth to 84 Mbytes/s, which is very near the raw performance of the SCI network.

Figure 3 shows the raw performance evaluation of DECK/SCI and ScaMPI, the Scali implementation of MPI [6]. The graph reveals that the performance of DECK/SCI is clearly better than that of ScaMPI. Note the influence of the zero-copy mechanism, which rapidly increases the bandwidth for messages starting from 8 Kbytes. DECK/SCI was configured to use the zero-copy protocol for messages greater than 8 Kbytes; this threshold, however, is user-configurable.

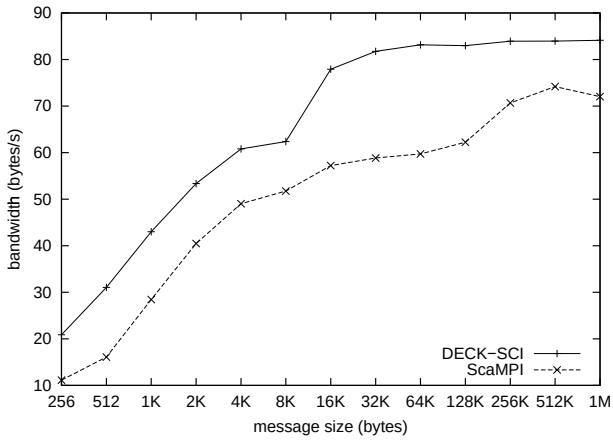

Figure 3. Bandwidth of DECK and ScaMPI on SCI.

In contrast to DECK/BIP, DECK/SCI is more efficient than MPI for all message sizes. This is, in part, due to the fact that DECK/SCI does not use any additional thread to manage communication. Furthermore, all three DECK/SCI protocols efficiently exploit the stream buffers of the SCI adapters.

3. The implemented applications

3.1. Execution environment

In order to investigate the potential and the feasibility of the proposed parallel programming environment for the target system, we have implemented two parallel algorithms that make heavy use of both computation and communication on the target system. These applications have been run on two SMP clusters available at our universities:

- a Myrinet cluster at the High Performance Research Centre (CPAD — PUCRS/HP); the cluster is composed of 16 nodes, each one being a HP E60 NetServer with two Intel Pentium III 550MHz processors and 128M DRAM, resulting in a 32 processor system; the nodes are connected by a 1.28 + 1.28Gbit/s Myrinet network based on Lanai7 32-bit cards; the operating system is Slackware Linux 7.1 and BIP is used to drive the Myrinet boards. The MPI implementation used in the tests is MPI-BIP

- an off-the-shelf SCI cluster available at the Institute of Informatics of UFRGS[1], composed of 4 bi-processed Pentium III 500MHz nodes, each with 256M of RAM (totalising 8 processors); the nodes run Conectiva Linux[2] version 5.0 and Scali Software Platform version 2.0.2, which includes ScaMPI and SISCI.

All the results have been obtained using the maximum number of nodes on each cluster, each figure being a mean of 10 executions.

3.2. Mandelbrot fractal

This application calculates the Mandelbrot set and draws a two-dimensional picture of it. It iterates the Mandelbrot function for every pixel on the picture, and sets the colour of the pixel according to the iteration where the orbit "escapes", with a maximum iteration value of 17500. There are no dependencies between the calculations done in different regions of the picture but different regions may be easier to calculate than others.

Our parallel version of the application uses a master/slave model where the master is responsible for distributing parts of the picture to be calculated and for drawing the already calculated parts. The picture is partitioned in horizontal slices and slaves keep requesting new slices to calculate until all slices are ready. Slaves communicate only with the master in a 1:n pattern.

Figures 4 and 5 present the results for a 600×600 fractal calculated with different number of slices, varying from 16 to 512. A higher number of slices results in smaller messages and

[1]Partially financed by FAPERGS, FINEP and CNPq
[2]A Brazilian derivative of Red Hat Linux.

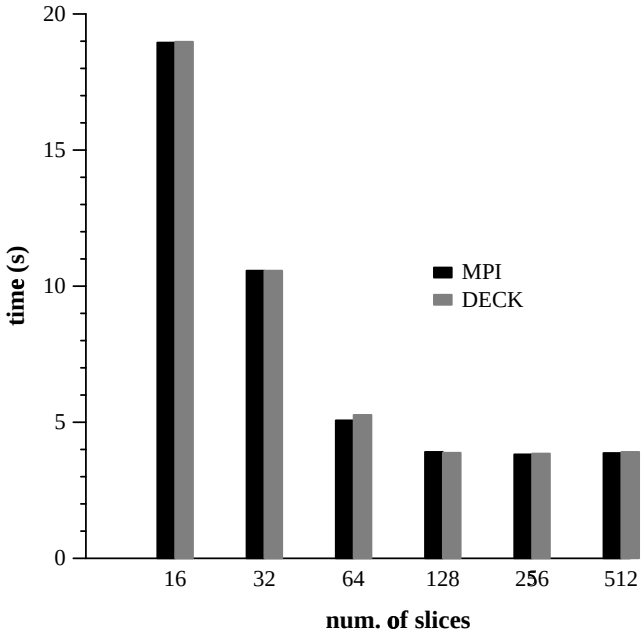

Figure 4. Results for the Mandelbrot algorithm on Myrinet.

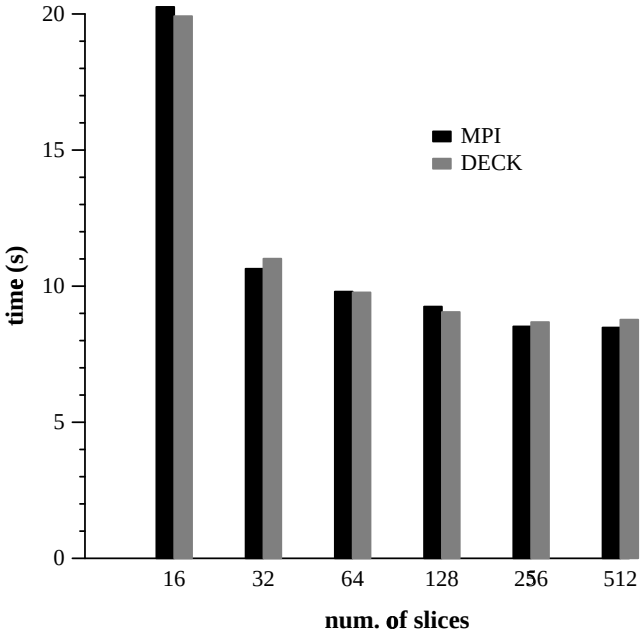

Figure 5. Results for the Mandelbrot algorithm on SCI.

better load balancing but increases the number of messages.

On MPI we have run 2 processes on each node, and on DECK the implementation uses two threads for the calculations and one dedicated to communication.

The results obtained with DECK are practically equivalent to those of MPI, with differences lying mostly on the tenths of seconds. On Myrinet the best results are obtained for a number of slices between 128 and 256. Notice that in this application the influence of the rv-daemon does not affect the final results due to the more irregular communication pattern, confirming that the use of threads can be suitable for applications involving high-performance communication.

One can notice a slight increase in the execution time when 512 slices are used; we believe this is a tendency from this point on, since slices become too small to compensate the time spent to send them between nodes. In other words, the grain of parallelism in this case is too small.

On SCI the behaviour is the same, with execution times also starting to increase at 512 slices (the absolute values are higher since the SCI cluster has only 4 nodes). DECK is slightly better than MPI for 64 and 128 slices.

3.3. Laplace equation

This application calculates the temperature distribution in a slab of a hypothetical homogeneous material completely insulated on the edges. Initially, the slab is at one uniform temperature throughout and a heat source is applied to one of the borders. The Laplace equation is used to solve this problem, being applied by means of an iterative method. The surface of the slab is divided in square sections, where each intersection is a point in a grid. The finer the grid, the more accurate the approximation, and the larger the problem.

From the adaption of the Laplace equation to a computational method, in this case the Gauss-Seidel approach [12], the temperature of a given grid point at a given iteration is taken to be the

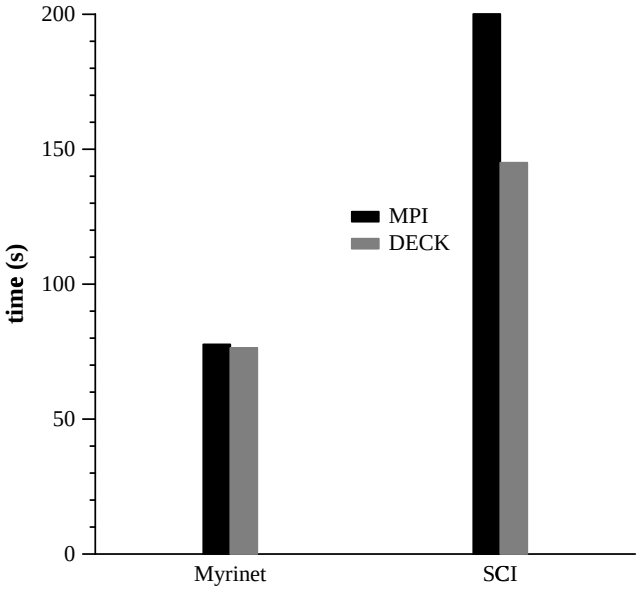

Figure 6. Results for Laplace's Equation.

average of the temperatures of the four surrounding grid points at the previous iteration. Border temperatures are always kept constant.

Our parallel version of the application partitions the image in rectangular regions, assigning each region to a node. Figure 6 presents the results for DECK and MPI for a slab of size 600×600.

Execution times are shown for both Myrinet and SCI distributing the regions equally among the available nodes. The same approach for achieving multiprocessing has been used, i.e., on MPI each machine executes two MPI processes, and on DECK two calculating threads are created.

In this application DECK has shown more encouraging results, presenting the same results as MPI on Myrinet and significantly better results on SCI. To our understanding, this behaviour is caused by different synchronisation semantics which depend on the message sizes. DECK messages are always asynchronous up to 8Kbytes, and the zero-copy protocol is used from this point on. As can be observed in Figure 3, ScaMPI keeps a rather constant evolution in performance throughout the curve, while DECK boosts bandwidth starting at 8Kbytes. Besides, to our knowledge, ScaMPI does not make use of a zero-copy mechanism, employing a *rendez-vous* (synchronous) protocol instead.

4. Conclusions and future directions

The experiments described in this paper have shown a performance evaluation of two implementations of the DECK environment when compared to equivalent MPI versions. In general, the performance presented by DECK with multiple threads is at least as good as that presented by the analysed MPI implementations, which represents an interesting alternative for the exploitation of SMPs in parallel applications.

In addition, in some cases, depending on the communication pattern presented by the application, DECK is able to outperform MPI running with multiple processes per node, as shown in Laplace's Equation. On the other hand, the raw performance evaluation of DECK on both technologies does not correspond to the results measured for the Mandelbrot algorithm.

Our next directions are the fine-tuning of DECK/BIP, with the goal of achieving better performance, and the implementation of additional applications to make a more complete validation of our programming environment. The group has the intention, in the future, of allowing the user to join both Myrinet and SCI technologies in a single application.

5. Acknowledgements

This work has been partially supported by grants from CAPES and CNPq.

References

[1] M. Barreto, R. Ávila, R. Cassali, A. Carissimi, and P. Navaux. Implementation of the DECK environment with BIP. In *Proc. of the First Myrinet User Group Conference*,

pages 82–88, Lyon, France, 2000. Lyon, INRIA Rocquencourt.

[2] Marcos Ennes Barreto. DECK: Um ambiente para programação paralela em agregados de multiprocessadores. Master's thesis, PPGC da UFRGS, Porto Alegre, 2000.

[3] N. Boden et al. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, February 1995.

[4] F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B. D. Johnsen, H. Kohmann, R. Nordstrøm, and P. Werner. Low-level SCI software requirements, analysis and predesign. Technical report, ESPRIT Project 23174 — Software Infrastructure for SCI (SISCI), May 1998.

[5] GM. Available at http://www.myri.com/GM, December 1999.

[6] L. P. Huse, K. Omang, H. Bugge, H. Ry, A. T. Haugsdal, and E. Rustad. ScaMPI—design and implementation. In Hermann Hellwagner and Alexander Reinefeld, editors, *SCI: Scalable Coherent Interface: Architecture and Software for High-Performance Compute Clusters*, volume 1734 of *Lecture Notes in Computer Science*, pages 249–261. Springer, Berlin, 1999.

[7] IEEE. Information technology—portable operating system interface (POSIX), threads extension [C language]. IEEE 1003.1c-1995, 1995.

[8] IEEE. Scalable Coherent Interface. IEEE 1596-1992, 1992.

[9] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. W. E. Freeman and Company, New York, 1982.

[10] MPI Forum. The MPI message passing interface standard. Technical report, University of Tennessee, Knoxville, April 1994.

[11] Fábio Abreu Dias de Oliveira. Uma biblioteca para programacao paralela por troca de mensagens de clusters baseados na tecnologia SCI. Master's thesis, PPGC da UFRGS, Porto Alegre, 2001.

[12] W. H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University, Melbourne, second edition, 1994.

[13] Loïc Prylli and Bernard Tourancheau. BIP: A new protocol designed for high performance networking on Myrinet. In José Rolim, editor, *Proc. of PC-NOW'98*, volume 1388 of *Lecture Notes in Computer Science*, pages 472–485. Berlin, Springer, 1998.

[14] Scali homepage—scalable Linux systems—affordable supercomputing. Available at http://www.scali.com, April 2000.

[15] Roland Westrelin. Une implémentation de MPI pour réseaux locaux à très haut débit: MPI-BIP. In *Proc. of the 11th RENPAR (Rencontres Francophones du Parallélisme)*, Rennes, 1999.