MMK - A Distributed Operating System Kernel with Integrated Dynamic Loadbalancing

Thomas Ludwig
University of Hamburg
206 PUBLICATIONS **9,192** CITATIONS

MMK - A Distributed Operating System Kernel with Integrated Dynamic Loadbalancing

Thomas Bemmerl, Thomas Ludwig

Lehrstuhl für Rechnertechnik und Rechnerorganisation
Technische Universität München
Institut für Informatik
Postfach 20 24 20, Arcisstr. 21
D-8000 München 2, FRG
Tel.: 0049-89-2105-8247 or - 2382
E-mail: bemmerl@lan.informatik.tu-muenchen.dbp.de

The paper presents an operating system kernel for highly parallel supercomputers, which was implemented on an iPSC/2 Hypercube with 32 processors. The kernel offers a process model, which is well suited for most partitioning strategies of parallel algorithms. The base for the efficiency of this object oriented, global, and dynamic programming concept are advances in communication network technologies (virtual fully connection) of some new parallel supercomputers. After presenting the functionality and the implementation of MMK (Multiprocessor Multitasking Kernel), the paper reports on an improved programming methodology based on a combination of data and task partitioning which leads to efficient computations on virtual fully connected highly parallel machines. MMK is an integral part of the TOPSYS-project (TOols for Parallel SYStems) and all tools support the MMK programming model.

1. Motivation and State of the Art

In the last years, research and development teams have proposed many scalable and distributed machines with different architectures. It has been shown, that good speedups are gained with these computers for several classes of applications ([CHE88],[BOD85],[TRO86]). The major drawback of these, partly commercially available, parallel computers is their complicated way of programming. Therefore, in the future the way of using scalable parallel computers and the programming productivity of these machines have to be improved. In the following we will concentrate on problems arising from the distribution of resources, especially of memory. Most of the problems mentioned are not relevant for shared memory multiprocessors.

One of the major reasons for the complicated way of programming existing scalable parallel machines is their inhomogeneous interconnection network. A typical feature of such parallel machines is, that the efficiency of the interprocessor communication depends on the localization of the communicating processors within the multiprocessor network. Examples of such parallel machines are hypercubes with store-and-forward communication [SEI85], all nearest neighbour architectures ([HAE85], [PAR88]), and cluster configurations [GIL88].

Consequences of these inhomogeneous interconnection networks for the way of program-

Partly funded by the German Science Foundation under contract number SFB 0342, project A1.

ming scalable distributed machines are the following:

- The parallel algorithms have to be adapted to the communication network.
- The mapping of the parallel programs onto the nodes of the parallel machine is difficult, because the programmer has to keep in mind the varying communication efficiency.
- The implementation of dynamic schemes for multitasking and multiprogramming is very difficult and inefficient. Therefore these machines are often only used for static application structures (numerical applications) with data partitioning.

Although it is well known that a strong separation between writing applications and keeping in mind architectural details is highly desirable, most of the actual parallel computers do not support a sufficiently high level of programming, for example the programmer has to keep in mind during writing his program the physical location (node number) of the communication partners and the operations and objects are often only locally (on the local node) available.

Computer architectures tried to eliminate the disadvantages of the inhomogeneous interconnection networks. Since some years, scalable parallel computers with distributed resources are coming up, offering more homogeneous interconnection networks. These machines offer at least virtual fully connected networks with a high communication bandwidth ([ARL88], [ANN87], [WHI88]). The availability of such homogeneous architectures supports the efficient implementation of more dynamic and homogeneous programming models and operating system kernels with global objects and operations. This was the motivation for the design and development of our dynamic object oriented operating system kernel MMK (Multiprocessor Multitasking Kernel) with global objects and operations, which will be explained in the rest of the paper. At first we will give a short description of available process models and kernels:

- The process models used in writing real applications for parallel computers are based on parallel extensions of sequential programming languages (C, Fortran, Pascal) ([PIE88], [SCH88],[SEI85],[HAE85]). In these language extensions, constructs for parallel processes and message passing between them are offered. These process modells are implemented via small operating system kernels or runtime libraries. An advantage of this approach is the close relation to conventional programming languages. The major drawback is the knowledge of node and process identifiers the programmers have to have.
- Another group of researchers proposes completely new parallel programming constructs. In this class of programming constructs, many different approaches are available based on the paradigms of procedural programming [WHI88], object oriented programming [AHU86], functional programming [HAL87] and logic programming [SHA86]. Problems arising with these approaches are compatibility and the effort necessary for rewriting huge application packages in these languages.

Most of the process modells support multitasking on the processing nodes, but there are also implementations which limit the number of processes to one (i.e. [HAE85]). This restriction results from a concentration on numerical applications, where multitasking is not necessary.

2. Features, Design Concepts, and Implementation of the MMK

The previous paragraph showed, that there are at the moment two contrary approaches for distributed memory machines. MMK is a compromise between the two approaches, but is closer to the conventional one because of compatibility reasons. In the MMK programming model we tried to enrich the static language extensions of the conventional approach with more dynamic constructs and global operations and objects. The result was a dynamic, transparent process model which allows the programmer to leave the specific architecture out of consideration during program construction. The key features and design concepts of the MMK are:

- MMK offers a transparent multitasking process model, which means that programmers can define multiple parallel processes. During program construction programmers neither have to keep in mind processor numbers nor locations of processes on processor nodes.
- When using the MMK programming model, the programmer thinks about his parallel program in an object oriented style. MMK offers active objects (tasks), communication objects (mailboxes), synchronization objects (semaphores), and storage objects (memory) with the usual operations the programmer can deal with. Object manipulation is only possible via the defined operations (for semantic details see part 3).
- All objects and operations are locally and globally available without differences in usage.
- The MMK offers only a minimum set of objects which reduces the complexity of implementing dynamic object (process) migration, dynamic load balancing strategies and monitoring support.
- All objects of a parallel program based on MMK can be dynamically created and deleted. This dynamic characteristic of the MMK supports also dynamic load balancing strategies

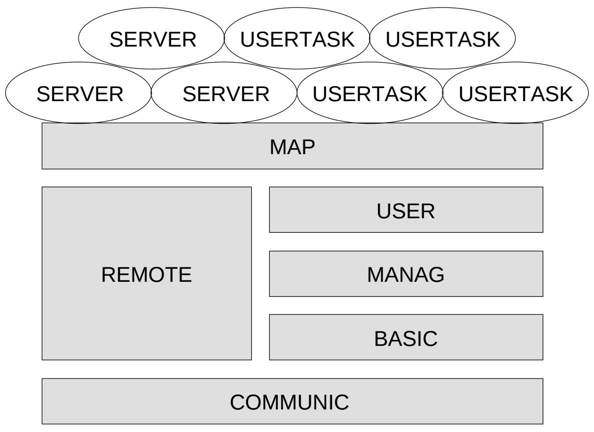


Figure 1: Hierarchical structure of the MMK.

and the implementation of non-numerical applications based on code (task) partitioning.

- Within the actual version of the MMK we implemented an experimental concept for dynamic object migration between processor nodes. This migration scheme is at the moment restricted to passive objects (mailboxes, semaphores).
- The intention to increase programers' productivity also motivated a project where interactive development tools for MMK-based applications were implemented [BEM88]. These tools (debugger, performance analyzer, visualizer) are supported by monitoring hooks and interfaces integrated in MMK delivering necessary runtime information.

One of the main implementation concepts of the MMK was portability. Therefore the only requested functionality of the underlying hardware are primitives, which implement a message passing concept; i.e. routines for sending messages to and receiving messages from other processor nodes. Another implementation concept supporting the portability of MMK is the highly modular and hierarchical structure. The overall structure of MMK is illustrated in figure 1 and can be explained in the following way:

- The MMK is fully distributed over the nodes of the multiprocessor. This means, that the same kernel runs at each node of the multiprocessor.
- The lowest layer implements the basic communication routines for passing messages to other nodes based on the interconnection network of the parallel machine.
- The layers BASIC, MANAG and USER can be compared with an implementation of a multitasking kernel on a single processor. These layers contain the local scheduling, state-, time-, object- and memory-management, and the local system calls for tasks,

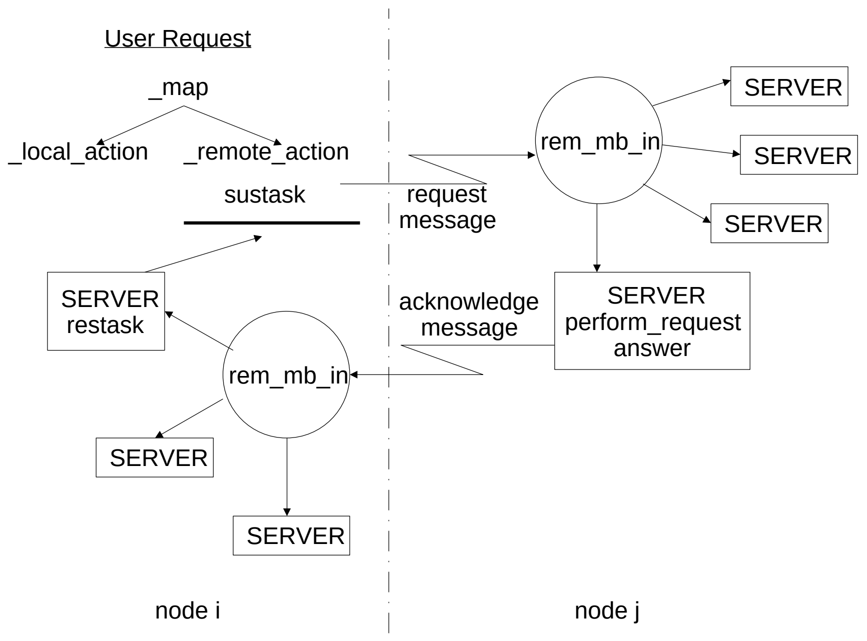


Figure 2: Remote procedure call.

semaphores, mailboxes, and memories.

– The REMOTE layer is responsible for forwarding remote actions (system calls etc.) to the nodes, where they have to be executed.
– The MAP layer identifies the location of the objects in the global object space. It determines the nodes where objects are located and initiates remote actions to these nodes.

Apart from the overall implementation structure, the MMK contains some interesting implementation details, which are explained next.

The migration of objects is implemented in the MAP layer. The migrate system call is implemented the way, that a new object with the same state as the old one at the source node (migration source) is created on the destination node (migration destination). Operations on the migrated object are handled with an algorithm, where the source node sends the new node number to the requester of the operation. The requester forwards the operation once more to the known destination node and makes an update of his object localization table.

Operations on remote objects are implemented via a server concept which is based on the well known remote procedure call (rpc). Remote actions are handled via defined mailboxes and corresponding server-tasks at the remote and the local processor node (see figure 2).

The MMK is totally written in C and at the moment implemented on an 8-processor iPSC Hypercube based on 80286 and an iPSC/2 Hypercube with 32 processors based on 80386. The kernel offers the described process model via a library which is linked to every node program of the multiprocessor.

3. Conventional Programming Based on MMK

We will now discuss in detail the objects of MMK and their corresponding operations and explain, in which way the MMK can be used for implementing parallel programs.

As already mentioned, the MMK operating system supports four different object types: tasks, mailboxes, semaphores and memories. The programmer can manipulate these objects only via the operations defined on them.

The active parts of an implementation based on MMK are the tasks. Each task is at each time in one of the following states: terminated, running, ready, waiting, suspended, or waiting suspended. Waiting means that the task is waiting for an access to an object of type mailbox, semaphore, or memory or for timeout. The state changes of the task are initiated by activities of the scheduler or by expilicit system calls (for more details see [BEM87]).

The communication between tasks is realized with mailboxes, which allow various forms of interprocess-communication. First of all they support direct communication from one sender to one receiver. Furthermore mailboxes provide means for communication from sender to any a receiver which is useful for certain types of applications. It is possible to define mailboxes with or without buffering, which has to be specified during creation time. With the mailboxes and the timeout mechanism of the mailbox system calls the user can for example implement asynchronous message passing (no-wait-send), synchronous message passing (rendezvous), buffered and unbuffered message passing.

For the synchronization of tasks the MMK offers counting semaphores with the usual operations defined on them. These semaphores include a timeout mechanism and the semaphore queues are organized in first-in-first-out manner.

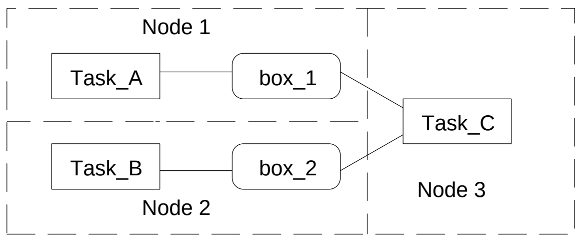
Finally, there is the object memory, which provides a simple facility of implementing a global memory on a loosly coupled parallel computer. The object of type memory is physically located on one single processor node but can be accessed to by each processor in the parallel computer. Of course, there can be more than one object of type memory. The realization of the memory object type is an attempt for investigating the problems resulting from virtual global memories on multiprocessors with distributed memory (see also [AHU86],[KES89],[RAS88]).

The first thing the programmer has to do is to devide the application into parts, which can potentially be computed concurrently. These parts form the tasks of the program. If there is any communication necessary between tasks, we have to add some mailboxes, semaphores, or memories. As a result we get a task graph describing the static structure of our application program. Nowadays most of the work necessary in this phase can be undertaken with computer-aided specification tools. At our department we adapted the task map feature of the specification system CARD-tools to facilitate the work with MMK programs [TER87].

After having finished the design of the task graph and the programming of the task bodies we can compile the already available source codes. The next step to do is the mapping of the MMK-objects onto the processor nodes of the machine. We will discuss this point in detail in the next paragraph. The information resulting from the mapping is also compiled and linked together with the precompiled sources. The linker yields an object code file, which now can be loaded into the nodes of the parallel computer.

Even if the programming language allows to hide such machine relevant parameters as node numbers, there is a point in the software development cycle, where we have to map the objects of our application program onto the processing nodes of our machine. The mapping is done via a text file which includes the names of all objects to be mapped, the corresponding node number and some additional parameters. For tasks we have also to specify the name of the task body, because each task body can have more than one task activation differing only by name. Figure 3 shows a task graph with three tasks and two mailboxes mapped on three processor nodes. The mapping file defines the initial connection scheme of the objects.

Let us now discuss the two typical parallelization paradigms for parallel computers. Most of the relevant implementations of parallelized algorithms use either data partitioning or task



```
GLOBAL_BEGIN
 MBOX (box_1 ,..., NODE 1)
 MBOX (box_2 ,..., NODE 2)
 TASK (Task_A INIT send , box_1 NODE 1)
 TASK (Task_B INIT send , box_2 NODE 2)
 TASK (Task_C INIT receive , box_1 , box_2 NODE 3)
GLOBAL_END
```

Figure 3: Example task graph and corresponding mapping file.

partitioning. We will show that the MMK programming model can deal with both of them.

One of the most common methods for using parallel computers efficiently is to distribute the load to the processor nodes by dividing the data among them. This method is called data partitioning and results for many applications in a good load balance between the nodes [FRI88]. The process graph is very simple because we have only one task body representing the algorithm. From this task body we make as many task activations as there are nodes in our computer and then map one task onto one node. The partitioning of the data also leads to an intertask communication but the resulting process graph is regular in most cases: for numerical applications we often use for example ring or torus structures [ROS88].

By using the mapping file we can adapt the process graph of the MMK application to the available number of processor nodes: we replicate the tasks and the mailboxes, so that the number of tasks equals the number of processors.

The second paradigm of using parallel computers is what we call task partitioning. Here the application is divided into tasks being distinct from each other and solving specialized problems. According to the replicated worker scheme we call this the specialized worker scheme. Nowadays, task partitioning is mainly used with distributed systems, e.g. systems consisting of some few coupled workstations, and for the implementation of non-numerical algorithms. With increasing number of processors there are hardly any applications with task partitioning for the simple reason that the number of tasks3 should exeed the number of processors for using the machine efficiently.

With our mapping file, we can arrange the tasks within the machine such that the efficiency is maximum. The multitasking facility on each node allows a grouping of tasks to minimize the load imbalance between the nodes. Second, software fault-tolerance can be easily achieved by replicating tasks of the process graph with the mapping file on different nodes.

As a conclusion we can say that the MMK programming model is suitable for both, application programs with data partitioning and programs with task partitioning. But with MMK we can also implement any mixture of the two parallelization types and use the machine efficiently for a wide spectrum of different algorithms.

4. A New Programming Style for Parallel Computers

In order to make parallel computers more available for general-purpose applications there will have to be a new programming paradigm which allows a problem oriented programming without taking into account the special qualities of the multiprocessor. There are two preconditions for such a new paradigm, which will now be described.

Most of the parallel computers suffer from not having enough communication links between their processing nodes and from the fact that the length of the interprocessor communication depends from the distance between sender and receiver. This means, most multiprocessors offer only inhomogeneous communication networks. This results in a programming style where the mapping of the processes plays an important role: the amount of communication can only be minimized by mapping communicating processes to the same or at least to neighbouring processors.

With the new parallel machines like the Intel iPSC/2 (see also [WHI88]) there is no such limit because they are virtual fully connected. There is no longer such thing like ’topology’ and the programmer is not longer forced to superimpose a logical structure on his algorithm which can be easily mapped to the physical structure of his computer.

With the hardware precondition being fullfilled, the programmer is not interested any more in node numbers and in whether two nodes are physical neighbours or not. As a consequence the MMK hides the node numbers from the programmer by offering only global objects which can be referred to by name from everywhere in the machine.

The MMK together with the communication mechanism of these new parallel machines like the iPSC/2 creates a new programming style which has the following advantages:

- First of all, we can use task and data partitioning together. The mapping file allows a replication of every specialized task so that the number of task activations can be adapted to the amount of data to be computed.
- The multitasking facility reduces the idle time of each processor and even allows multiprogramming to increase the usage of the supercomputer.
- The replication facility of the mapping file allows to control the granularity of the application program, which is important for loadbalancing ([BAI87], [THE86]).
- The remote creation facility for tasks allows a dynamic loadbalancing controlled by the programmer as he can activate new tasks on nodes with only low load.
- Furthermore the global objects allow a dynamic loadbalancing controlled by the operating system which is the greatest advantage of referencing objects only by name and not by location. At the moment, the loadbalancing feature is only implemented for non-active objects and not for tasks.

In summary one can say that a heterogeneous software concept being composed of a mixture of task and data partitioning is well suited for programming more heterogeneous machines, for example machines with special nodes for file-I/O, graphic output or sensor input. These special nodes can be efficiently used by assigning specialized tasks to them.

In the field of fault-tolerance we can easily implement software fault-tolerance by replicating processes and the communications between them to different resources. At our laboratory we are now making first measurements to evaluate the costs of such methods.

For numerical algorithms which are mostly using data partitioning we can easily add specialized tasks for the pre- or post-processing of the data (for example I/O-tasks, tasks for graphic). At the moment we are parallelizing a placement algorithm for cells in gate arrays [KLE88].

5. Performance Results

In order to compare the quality of the MMK implementation to other existing operating systems we precisely investigated the performance of MMK.

– Of great importance for the evaluation of MMK is the knowledge about the time consumption of each MMK system-call. We will discuss this point in detail and make some comparisons to other operating systems.
– Furthermore, a profiling of the system-calls yielded some interesting results concerning the detection of performance losses within MMK.

Our aim was not only to compare MMK to other systems but mainly to assess the implementation quality. As already mentioned, MMK is implemented on top of NX/2, so performance losses resulting from more hierarchical software layers are unavoidable. A thorough investigation of MMK will result in a concept for integrating important features of MMK into the NX/2-system itself, thus multiplying the performance of this redesigned MMK.

For all measurements we used a maximum number of two nodes and two tasks performing system-calls to be measured. All system-calls were investigated with varying combinations for their corresponding parameters. Results will show that there exist different types of interdependencies between parameters and performance. Most important for multiprocessors is the comparison of local and remote system-calls. For some system-calls this distinction depends on the location of a related object: if a receiving mailbox is located on the same node as the sending task, receiving of messages will be a local operation; otherwise receiving is done remotely via remote-procedure-call. System-calls like create_mailbox distinguish by parameter whether the operation is local or remote. Remote-procedure-calls were only performed between directly neighbouring nodes, because the time overhead necessary for multihop communication depends only on the underlying hardware and has no influence in MMK's performance.

Table 1.a shows the times necessary for those system-calls which only depend from being local or remote but not from any other parameter. In Table 1.b we summarize the measurements for system-calls depending on parameters. As can be seen from the table sending and a)

function	local	remote
crembox	0,18 ms	3,38 ms
delmbox	0,33 ms	3,30 ms
cresema	0,16 ms	3,30 ms
delsema	0,33 ms	3,30 ms
relsema	0,12 ms	3,31 ms
reqsema	0,11 ms	3,26 ms
cretask	0,58 ms	4,09 ms
deltask	0,42 ms	3,43 ms
migrate	-	6,75 ms

b)

function	local	remote
*) recmsg	0,05 ms	3,41 - 112,12 ms
*) sndmsg	0,06 - 26,17 ms	3,50 - 112,54 ms

* 0 - 10 KBytes messages

c)

function	time
*) make_global	0,07 - 0,12 ms
scheduler	0,076 ms
RPC	3,06 ms

* 0 - 99 objects

d)

Send message	Bytes	Time (ms)	
		local	remote
NX/2	0	*)	0.290
	512	*)	0.850
	1024	*)	1.040
MMK	0	0.063	3.545
	512	1.405	9.240
	1024	2.727	14.780
PEACE	0	0.080	2.380
	512	0.200	2.540
	1024	0.305	2.660

e)

Receive message	Bytes	Time (ms)	
		local	remote
NX/2	0	*)	0.290
	512	*)	0.850
	1024	*)	1.040
MMK	0	0.051	3.478
	512	0.051	9.295
	1024	0.051	14.852
PEACE	0	0.080	1.570
	512	0.200	1.735
	1024	0.305	1.860

*) no figures known

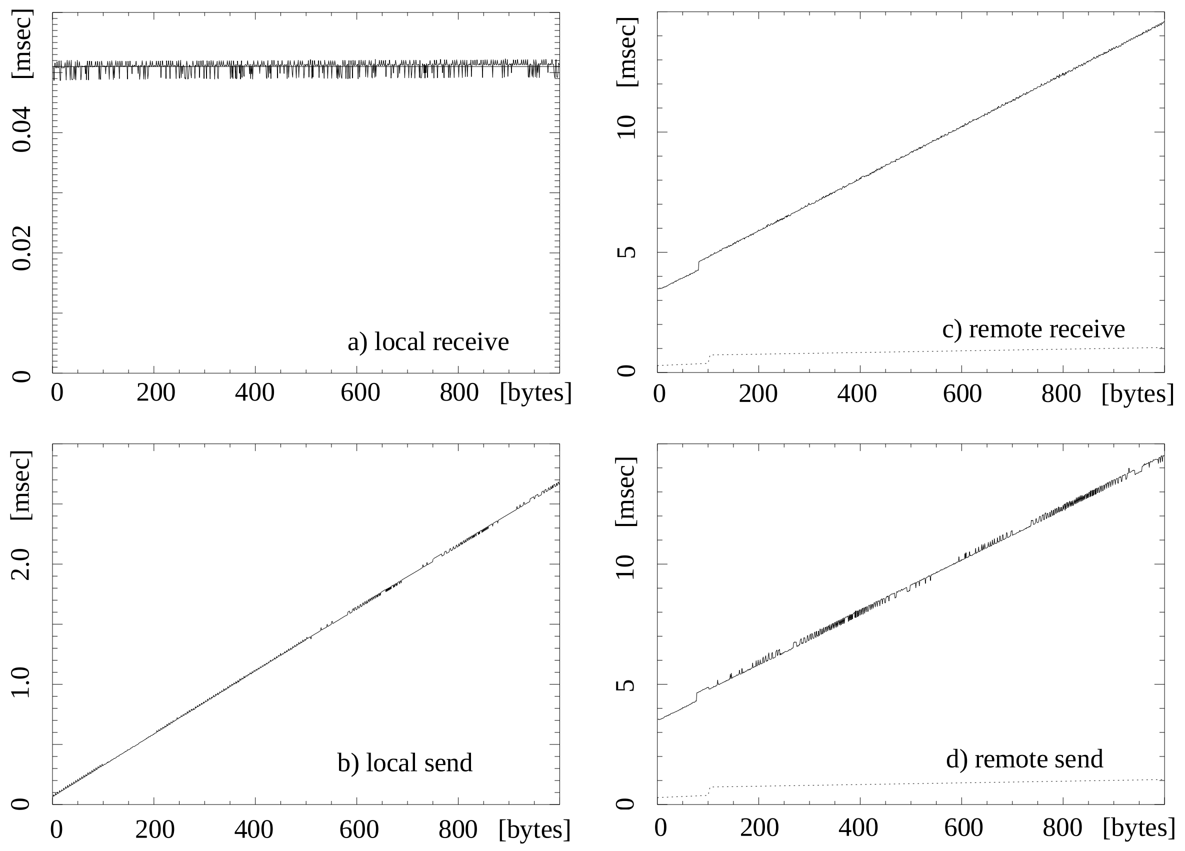
Tables 1.a-e: Time consumption of MMK functions and comparison to other systems.

receiving depend on the message length. Only local receive has a constant time because there is no copying of the message but only an assignment of pointers.

In addition figures 4.a-d illustrate the kind of interdependence between time and message length: time is in direct proportion to the number of bytes sent so that one can project the time necessary for sending or receiving a certain message. The figures for remote actions also show a jump for messages smaller and larger than 80 bytes, resulting from two communication mechanisms within NX/2 for short and long messages. The overhead of the MMK protocol can be estimated by comparing the remote communication performance to that of NX/2. The dotted line shows the time necessary for raw communication via NX/2 system-calls.

Finally in table 1.c we present some figures about the profiling of MMK functions. Most important is the remote procedure call which can only be improved by integrating the mechanisms into NX/2 itself. The make_global procedure is used for reserving a global object identifier for a local object and is called after object creation.

Comparison of performance measurements to other operating systems is very difficult be-cause of three reasons: First of all there are nearly no publications with detailed performance informations. Second, even if measurements exists, there is the unavoidable problem of differ-ing hardware concepts and realizations making performance figures incomparable. Last but not least also equivalent procedure calls may slightly differ in semantics, so that sending mes-sages with MMK can not be compared to sending routines of other systems. So the following



Figures 4.a-d: Time consumption of communication.

shall only give an idea of performance differences between MMK and other systems.

PEACE is the operating system of the SUPRENUM parallel computer. Performance figures are given in [SCH87] and listed in tables 1.d-e together with the times necessary for remote operations with NX/2 [BOM89]. There is a considerable difference between PEACE and MMK whereas PEACE and NX/2 are very similar. Note that the PEACE figures are only given for intercluster communication where at best 16 processors can communicate with each other.

6. Conclusion and Future Works

Our first results show, that the MMK is an adequate compromise between the hardware-oriented operating systems (e.g. NX/2) and programming concepts with high abstraction levels like Linda which often do not provide enough efficiency for numerical applications.

We also were working on an improvement of the MMK implementation to increase the performance. Two issues are of special importance: the global object space for localization independent object manipulation and a migration mechanism for objects. The migration mechanism brings together the advantages of the memory managment unit and of the iPSC/2 network and facilitates a migration of single pages across the interconnection network. Costs for migration are reduced because pages are only migrated on demand. We actually compare different loadbalancing strategies for the control of the migration mechanism.

The integration of both the concept of the global object space and the concept of the dynamic loadbalancing invisible to the programmer results in the programming philosophy of mixed data and task partitioning which makes multiprocessor supercomputers easier to use and more adequate for general purpose applications.

References

[AHU86] Ahuja, S.; Carriero, N.; Gelernter, D.: Linda and Friends, IEEE Computer, 1986

[ANN87] Annaratone, M. et.al.: The Warp Computer: Architecture, Implementation and Performance, IEEE Trans. on Comp., Dez. 1987

[ARL88] Arlanskas, R.: iPSC/2 System: A Second Generation Hypercube, Int. Hypercube Conference, 1988

[BAI87] Baiardi, F.; Pelagatti, S.; Vanneschi, M.: Processor Managment in Highly Concurrent Systems, Esprit 415, Sep. 1987

[BEM87] Bemmerl, T.; Schöder, G.: A portable realtime multitasking kernel for embedded microprocessor systems, Microprocessing and Microprogramming, The Euromicro Journal, Vol:21, 1987

[BEM88] Bemmerl, T.: An Integrated and Portable Tool Environment for Parallel Computers, IEEE Int. Conference on Parallel Processing, St. Charles, August 1988

[BOD85] Bode, A.; Fritsch, G.; Henning, W.; Volkert, J.: High Performance Multiprocessor Systems for Numerical Simulation, IEEE Parallel Processor Conference, St. Charles, 1985

[BOM89] Bomanns, L.; Roose, D.: Communication benchmarks for the iPSC/2, Proceedings of the First European Workshop on Hypercube and Distributed Computers, Rennes, France, Oct. 1989, pp. 93-103, North-Holland, 1989

[CHE88] Chen, M.; DeBenedictis, E.; Fox, G.; Li, J.; Wacker, D.: Hypercubes are General-Purpose Multiprocessors with High Speedup, CALTECH Report, 1988

[FRI88] Fritsch, G.; Ludwig, T.; Volkert, J.: Many-Particle Problems on Distributed Shared Memory Systems, Proceedings of the CONPAR 1988, Manchester, UK, Vol.: A

[GIL88] Giloi, W.K.: The SUPRENUM Architecture, CONPAR 88, Manchester, September 1988

[HAE85] Haendler, W.; Maehle, E.; Wirl, K.: DIRMU Multiprocessor Configurations, IEEE Int. Conference on Parallel Processing, St. Charles, 1985

[HAL87] Halstead, R.H.: Multilisp and Multilisp-oriented Architectures, MIT/ZTI-Symposium, München, Nov. 1987

[KES89] Kessler, R.E.; Livny, M.: An Analysis of Distributed Shared Memory Algorithms, Proceedings of the 9th International Conference on Distributed Systems, pp. 498-505, Newport Beach, California, June 1989, IEEE Computer Society Press, Washington

[KLE88] Kleinhans, J.M.; Sigl, G.; Johannes, F.M.: GORDIAN: A New Global Optimization / Rectangle Dissection Method for Cell Placement, IEEE Int. Conference on Computer-Aided Design ICCAD-88

[PAR88] Parsytec: Megaframe - Supercluster, Parsytec, 1988

[PIE88] Pierce, P.: The NX/2 Operating System, Int. Hypercube Conference, 1988

[RAS88] Rashid, R.; Tevaninan A.; Young, M. et.al.: Machine Independent Virtual Memory Managment for Paged Uniprocessor and Multiprocessor Architectures, IEEE Transactions on Computers, Vol. 37, No. 8, Aug. 1988

[ROS88] Rost, J.; Maehle, E.: Implementation of a Parallel Branch-and-Bound Algorithm for the Travelling Salesman Problem, Proceedings of the CONPAR 1988, Manchester, UK, Vol.: A

[SCH87] Schröder, W.: A Distributed Process Execution and Communication Environment for High-Performance Applicationsystems, Technical Report, Gesellschaft fuer Mathematik und Datenverarbeitung, Germany, 1987

[SCH88] Schröder, W.: The Distributed PEACE Operating System and its Suitability for MIMD Message Passing Architectures, CONPAR 88, Manchester, Sept. 1988

[SEI85] Seitz, C.L.: The Cosmic Cube, Com. of the ACM, Jan. 1985

[SHA86] Shapiro, E.: Concurrent Prolog: A Progress, Report, IEEE Computer, 1986

[TER87] Terry, C.: CASE-Tools Run on an Expanded Range of Computer Systems, EDN, 23.07.87, p.221

[THE86] Theimer, M.M.: Preemtable Remote Execution Facilities for Loosely-Coupled Distributed Systems, Stanford University, Report No. STAN-CS-86-1128 (also CSL-86-302), Stanford, 1986

[TRO86] Trottenberg, U.: SUPRENUM - A MIMD Multiprocessor System for Large Scale Scientific Computing, EUROMICRO ’86, Venice, 1986

[WHI88] Whitby-Strevens, C.: Supernode: transputer and software, Int. Conference on Supercomputing, 1988