MANNA: Prototype of a Distributed Memory Architecture With Maximized Sustained Performance

by

W.K. Giloi*, U. Bruening, W. Schroeder-Preikschat*****

* GMD Research Institute for Computer Architecture and Software Technology
Rudower Chaussee 5, 12489 Berlin, Germany, e-mail: giloi(a)first.gmd.de

** University of Kiel, Germany
e-mail: ulrich@first.gmd.de

*** University of Potsdam, Germany
e-mail: wosch@first.gmd.de

Abstract

The sustained performance of superscalar microprocessors amounts to only a fraction of their peak performance rating. In parallel computers realized with them the discrepancy between the measurable performance and the theoretical peak performance is even more dramatic. Reaching a satisfactory sustained performance for the single processor is primarily a compiler problem. The sustained performance of message-passing machines depends also on other components of the architecture such as the interconnect and the operating system. It is shown in the paper how through a combination of innovative architectural solutions the sustained performance of a distributed memory parallel computer is maximized. The key to effective latency hiding through overlapping communication and computation is the operating system architecture. It is shown that the programmability of such architectures can be enhanced by providing the programmer with parallelizing compilers and/or a global address space provided by a virtual shared memory mechanism. Prototypes of a parallel computer in which all these measures have be incorporated were built and are used in practice; benchmark performance figures are reported in the paper.

1. RATIONALE FOR MANNA

1.1 Use of Stock Processors

The standard architecture of parallel computers is the distributed memory, message-passing machine realized with stock processors. Although a custom-designed node hardware might allow for a more unconventional node architecture, it will hardly lead to a competitive product. The reason is that the cost of design and continuing upgrading of a complex superscalar processor of the highest performance are so exorbitant that they cannot be amortized only by their use in parallel computers. Rather, processor lines such as the *Pentium* and its successors or the *powerPC* must reach a production volume of millions of processors a year to be profitable, a volume that comes only from PCs, workstations, and embedded systems. Thus, stock processors not only are much less expensive but offer the additional advantage of a design that is automatically on the future upgrade path of that device.

The MANNA development began in 1990. At that time, the Intel i860—the first superscalar processor—had just been introduced into the marketplace. Because of its power the i860 was predestined for its use as node processor of parallel supercomputers. Novel features were

- an envisioned peak performance of 50 MIPS and 50 MFLOPS d.p., respectively, for the 50 MHz version;
- fast, on-chip snooping logic for the data cache;
- pipelined memory access in burst mode.

Consequently, the MANNA design was based on the i860XP. The design was begun already a year before the first silicon became available. This was possible because a simulation model of the processor was purchased from Logic Modelling and integrated into the Cadence/Verilog design environment. Based on that model, the complete 2-processor node of MANNA was simulated and verified during the waiting time for the first samples.

1.2 Programming Models Supported

The natural programming paradigm for distributed memory architectures is the message-passing model. However, this model is perceived by most users as difficult, since it requires an application to be partitioned into a number of communicating parallel processes, programming inter process communication in the form of explicit calls of *sends* and *receives*. In this effort the programmer must take the data dependencies among the processes into account. Moreover, he or she is burdened with the task of providing a favorable initial data distribution. This all raises demands the average user may find hard to cope with.

In contrast, the shared-memory programming model avoids—at least in principle—the difficulties of the message-passing paradigm. Since data are globally accessible, no specific data distribution is mandated, and since communication happens through shared variables, no message-passing constructs are needed. However, only distributed memory architectures are scalable.

One can try to obtain the best of both worlds by realizing a global address space on a distributed memory machine. There exist two approaches, viz.:

- the virtual shared memory (VSM) or of
- the distributed (physically) shared memory (DSM).

VSM does not raise the hardware cost; however, it induces software overhead. In contrast, the DSM requires in each node a sufficiently large secondary cache and a directory to maintain cache coherence, thereby driving up the hardware cost. Hence, in both cases must cost-effectiveness be traded for the convenience of the shared memory programming model. Therefore, we ruled out the DSM approach but decided to rather build a pure message-passing machine with the highest possible performance and cost-effectiveness and find better ways of simplifying parallel programming.

For numerical, i.e. data parallel, applications programming of message-passing machines has been significantly facilitated by such tools as Parmacs [LST 92], PVM [Bea 93], or MPI [MPI 93], which allow the user to program in a machine-independent fashion. Moreover, for standard applications the explicitly programming of communication is avoided by making it a library function.

The ultimate solution to the parallel programming problem may be the parallelizing compiler that allows the user to formulate his applications at a high level of abstraction, without having to be concerned with specific parallel computing aspects such as program decomposition, data distribution, data access synchronization, and inter process communication and coordination. The task of an automatically parallelizing compiler is significantly supported by high-level programming models such as High Performance Fortran [HPF 92] or the more flexible and generally applicable PROMOTER model specifically developed for machines like MANNA [GaS 93]. In cases where the data type *pointer* is needed, e.g., in non-numerical applications programmed in AI languages like Lisp, the programmer may in addition be provided with a global address space created through the virtual shared memory mechanism.

1.3 Sustained vs. Peak Performance

The performance of parallel computers is usually rated as peak performance which, in turn, is computed as the peak performance of the node processor times the number of nodes. For the user, peak performance is a useless figure which not even allows for a relative comparison between different parallel machines. This has two reasons. (i) The peak performance rating of the makers of microprocessors is often inflated, i.e., could not even theoretically be reached. (ii) Even with the best compilers the benchmark performance usually reaches only a fraction of the peak. In the peak performance rating, different parallel computers using the same node processor are rated the same, yet they may differ significantly in terms of sustained performence. The purpose of the MANNA design was maximization of sustained performance, by carefully analyzing the reasons for a poor sustained-to-peak performance ratio and devising measures for avoiding performance degradation as much as possible.

1.4 The Week Points of Distributed Memory Architectures

Potential drawbacks of distributed memory architectures are:

- poor utilization of the superscalar node processor by the compiler
- communication latencies
- inadequate interconnection topologies
- inefficient implementation of the VSM mechanism (if needed).

The utilization of the superscalar node processor by the compiler depends on the degree of optimization the compiler can do in order to keep the instruction execution pipeline(s) of the processor filled with useful operations rather than bubbles. At the present state-of-the art, the degree of optimization still leaves much to desire.

Communication latency usually is the largest performance-limiting factor in message-passing machines. The latency T_l of a message transfer such as *send* or *receive* is given by two factors.

T_l [µs] = T_S [µs] + L [bytes] / R_S [Mbytes/sec.]

with T_S being the message start-up time, R_S being the transmission rate through the interconnect, and L being the block length of the message. If message passing is performed by the operating system kernel (the normal case), T_S is a parameter of the operating system.

Communication latency often is (falsely) seen purely as a function of the interconnect transmission rate. This is not the case if any software (e.g., the operating system kernel) is involved in the communication. In that case, the start-up time T_S is typically in the range of some 10 to some 100 microseconds, and the fastest interconnect of the world cannot solve the communication latency problem. On the other hand, if the entire inter process communication is handled strictly by hardware, one can get T_S down to a range of some 100 nanoseconds to some microseconds, so that the message transmission time becomes the dominating factor. However, a machine without any operating system support would be hard to program.

To minimize the communication latency, one can take either of the following approaches (or a combination of both):

- *latency minimization*—making the node hardware, the operating system, and the interconnect as fast as possible;
- *latency hiding*—avoiding idle times caused by waiting for communication.

The remainder of this paper deals with the measures taken in MANNA to maximize the sustained performance of the system by minimizing communication latency by an appropriate operating system (Chapters 2 and 3) and node architecture (Chapter 4), and by devising an innovative interconnection topology with minimal cost and maximum performance (Chapter 5). Chapter 6 gives performance figures. Chapter 7 discusses the mechanisms for an efficient implementation of virtual shared memory. Chapter 8 deals with programming environments provided on MANNA, and Chapter 9 gives a brief overview about the uses of the system.

2 LATENCY MINIMIZATION IN THE OPERATING SYSTEM

2.1 Functionality and Start-up Time

To be universally applicable and programmable, a parallel computer must provide the appropriate operating system (OS) services. As this includes inter process communication, the message start-up latency depends strongly on the functionality of the OS *kernel.*

The simplest case is the single-user, single-tasking operation with one thread of control or several concurrent, unscheduled threads (lightweight processes). Here the *single address space mode* is appropriate, in which the entire node software—application as well as OS functions—runs without protection in one address space. This simple approach of minimal software overhead is called a *dedicated machine.* Communication is usually provided not by an OS process but by runtime library routines.

Yet there may be cases where the operating system should provide a *multi-processing environment* by adding pre-emptive scheduling and process environment separation. In a single-processor node, each communication activity leads now to an OS kernel trap and, consequently, an environment switch. Even when the user needs only a dedicated machine, the overhead for the potential multi-processing operation must be paid. The highest overhead is encountered when the system is designed for *multi-user operation*, in which case *user task isolation* are added. Figure 1 shows how the set of requirements for the OS kernel grows with increasing functionality.

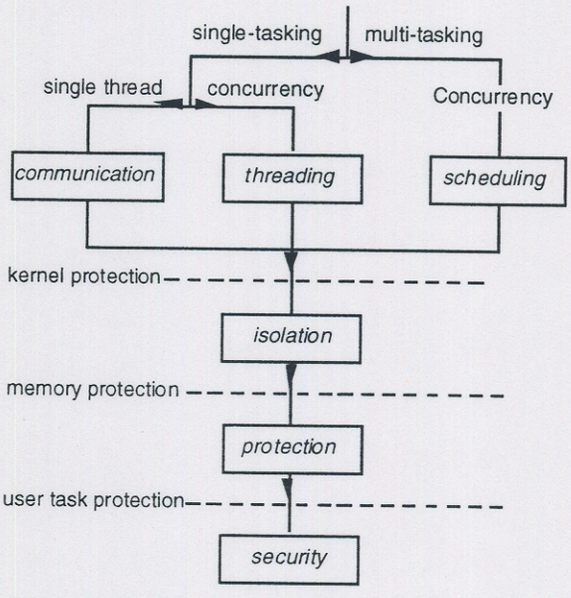

Figure 1 Increasing functionality of kernel services

2.2 The PEACE Family of Kernels

The MANNA operating system PEACE is based on the novel concept of the *OS kernels family* [Sch 91] which allows any desired OS functionality to be provided with minimal overhead by applying the following construction principles [Sch 94].

- The variety of kernels tuned to optimally support specific user demands and operating modes [Sch 91] is obtained by a common communication nucleus augmented for the specific use by minimal function extensions as indicated in Figure 2. The nucleus runs in supervisor mode, while all functional extensions are lightweight processes running in user mode.
- Since the family members typically differ only in a few components that concern the user interface or the run time behavior, object-orientation is the natural approach toward implementing the kernel family. That is, new family members are derived from existing ones through inheritance, and type and function polymorphism is used to tailor OS abstractions to specific user demands.

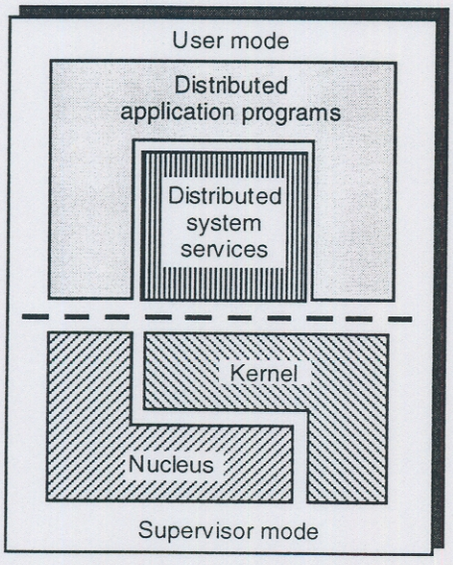

Figure 2 Architecture of PEACE

2.3 Performance Figures

Table 1 lists the latency for a single communication activity (*send* or *receive*) of a number of commercial computers including MANNA [BGS 94]. OSF/1 (Mach 3.0) is a UNIX look-alike. PUMA, NX, and CMOST are typical microkernel operating systems. "Active Messages" is an experimental OS based on the active message concept [Eea 92]. Compared to the microkernel OSs and by virtue of the family concept, PEACE is approximately a factor of 2 faster (23 microseconds on the 50-MIPS i860XP superscalar processor). This stems from the fact that the overhead of any member of the PEACE family of kernels is only what is absolutely needed for the functionality provided, so that the user newer pays for something (s)he doesn't need nor want.

Machine	Operating System	Latency [µs]
Paragon XP/S	OSF/1	240
nCUBE/2	PUMA	110
iPSC/860	NX	100
CM-5	CMOST	65
Paragon	PUMA	50
nCUBE/2	Active Messages	32
MANNA	PEACE *	23

* single-processor operation

Table 1: Message-passing latency of parallel computers in the dedicated machine mode

If the multiprocessing kernel of PEACE is invoked on MANNA, the start-up time becomes about 80 microseconds. Over one third of that time must be attributed to the *kernel* trap that becomes part of each communication.

2.3 Additional Kernel Functionality

As discussed below, there are other optional functionalities of the MANNA architecture that require specific OS support. Therefore, specific kernel family members are provided for:

- *AP-AP Operation* :
  the use of both processors in the MANNA node as *Application Processor* (AP);
- *AP-CP Operation* :
  the use of one of the node processors as dedicated *Communication Processor* (CP);
- *VSM Operation* :
  a variant of the AP-CP operation that provides a global address space through *virtual shared memory* (VSM).

3. LATENCY HIDING IN MANNA

3.1 Latency Hiding by Overlapping Communication With Computation

The effect of communication latency can be drastically reduced by equipping the node with a dedicated *communication processor* (CP) in addition to the *application processor* (AP) [GaS 89]. In this symbiosis the task of the AP is to uninterruptedly *produce megaflops*, while the CP executes the communication tasks of the OS *kernel*. Both processor work in parallel; consequently, the communication start-up time occurs in the CP but is hidden from the AP. The AP sees only the latency of sending a communication request to the CP.

The concept of overlapping communication with computation is about 15 years old. The first system ever built with a dedicated communication processor was UPPER [GaB 81], [Gil 84], an early distributed memory parallel architecture completed in 1981. The UPPER node consisted of two MC68000 processors. One was working as the node CPU proper (AP), while the other one took care of inter process communication (CP). The result was disappointing: the gain in communication speed was only marginal, for the communication latency between AP and CP of 1...2 millisecond used up most of what we had hoped to gain.

3.2 Efficient Latency Hiding

The prerequisite for efficient latency hiding in message-passing machines through the use of two processor, CP and AP, is an appropriate OS architecture. In that respect, all commercial parallel computers that have also a 2-processor node fail to utilize it to its full potential because of the communication overhead of their microkernel operating systems.

The kernel family concept on which PEACE is based leads not only to extremely lightweight OS kernels but also to their modularization—in contrast to the monolithic microkernel. Figure 3 shows the hierarchical decomposition of the PEACE kernel into three modules [Sch 94]: NICE (network independent communication environment), COSY (communication system), and POD (port drivers).

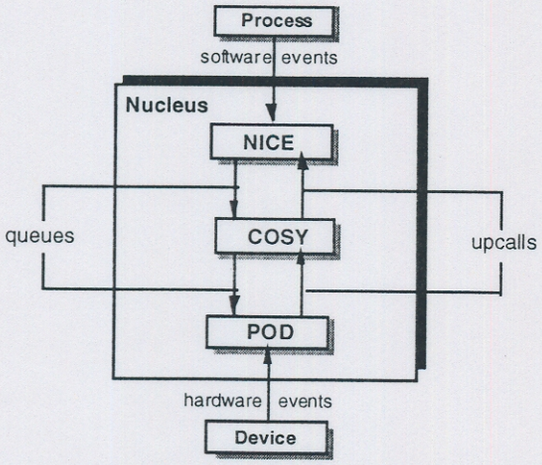

Figure 3 Modularization of the PEACE kernel

NICE takes care of the inter-thread communication policies, COSY handles the communication protocols, and POD is the glue between the OS nucleus and the communication hardware (the network).

The modularization of the PEACE kernel allows for an optimal distribution of OS functions over the two processors of the MANNA node. Figure 4 shows an example [BGS 94] (depending on the application, there may be variations of this). The application processor (AP) executes solely NICE functions, while the complete NICE-COSY-POD suite is executed only on the communication processor (CP). On the CP this may take 23 microseconds, yet the AP "sees" only the couple of microseconds needed to put a communication request to the CP. The dramatic improvement in the effectiveness of the latency hiding scheme of overlapping communication with computation is evidenced by Table 2. With the latency hiding mechanisms provided by the MANNA two-processor node architecture (2 Intel i860XP) and PEACE, the effective latency is at most 4 microseconds.

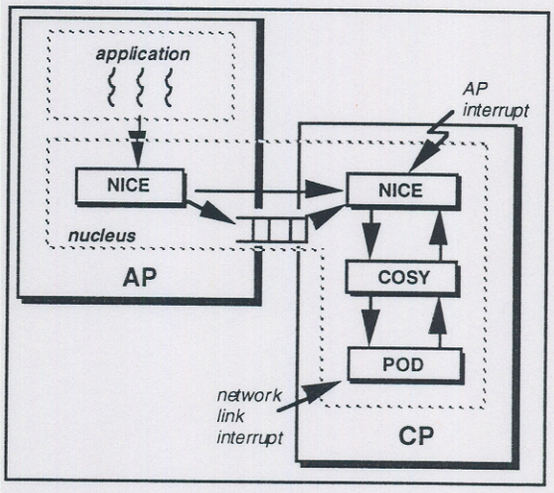

Figure 4 Distribution of kernel functions over AP and CP

Properties	Intel Paragon	MANNA
Operating system	OSF/1	PEACE
Size of kernel	7 Mbytes	0.2 Mbytes
Communication latency	240 µs	23 µs
Effective start-up time	20 µs	4 µs

Table 2 Comparison between OSF/1 and PEACE. Both nodes have 2 processors i860XP.

The overall advantages of the innovative concepts of the PEACE operating system are:

- an order of magnitude lesser code for the kernel implementation
- an order of magnitude lesser communication latency
- extremely efficient latency hiding—the effective start-up time is in the same order of magnitude as if there were only naked hardware, i.e., no OS at all!

4. THE 2-PROCESSOR NODE

Figure 5 depicts the block diagram of the 2-processor MANNA node. Both AP and CP are superscalar processors i860XP; thus, they both have the same memory management and can snoop on each other's caches. The processors communicate through the shared 32-Mbyte DRAM node memory. The elaborate memory design features burst transfer support from 4 interleaved memory banks in page mode via a three-stage pipeline; that is, the memory access latency of 7 clock ticks is overlapped with the previous access cycle. This gives the memory a 7-1-1-1 cycle access pattern and a resulting access bandwidth of 400 Mbytes per second once the pipeline is filled and the accesses are to the same page. Hence, the DRAM node memory has almost the behavior of a secondary cache.

A bi-directional communication link with a data rate of 2x50 Mbytes/second connects the node with the interconnection network, a hierarchy of crossbars (see Chapter 5).

The two-processor MANNA node is completely symmetric; thus, it does not matter which of the two processors is AP and which is CP. For applications that have a relatively low message traffic, the dedicated CP may not be sufficiently utilized. In this case, it is a better approach to use both processors may as APs. PEACE provides kernels for both operations.

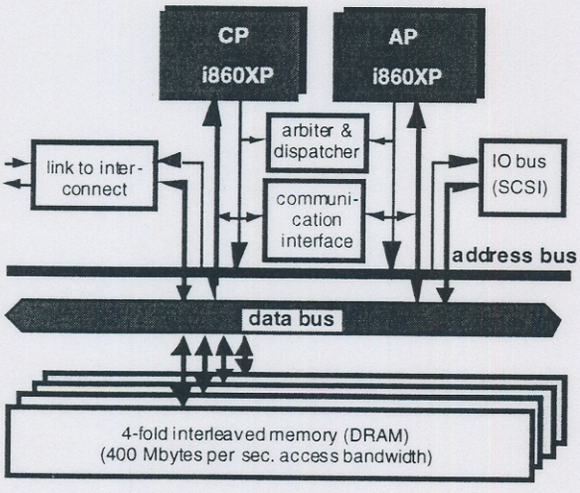

Figure 5 Block diagram of the MANNA node

5. MANNA INTERCONNECTION NETWORKS

The desired general applicability of MANNA would be in conflict with an interconnect that places a strong penalty on non-local communication. This ruled out topologies that work satisfactorily only for strongly local access patterns, e.g., the 2D or 3D mesh. Another MANNA invention, the *hierarchy of crossbar interconnect* [Mon 89], provides "the best of all worlds" by satisfying the requirements of

- high global connectivity (no penalty for non-locality)
- low blocking probability
- minimal hardware cost,

The basic building block of this unique, scalable network topology is a byte-wide, bi-directional 16x16-crossbar switch realized as a single-chip ASIC. The transmission rate is the same as that of the INI, i.e., the bus clock of the processor. Groups of 10 nodes are interconnected via the 16x16 crossbar, to form *clusters*, each cluster thus having 32 processors. This leaves 6 uncommitted links of the cluster crossbar for interconnecting the cluster with other clusters. Up to 4 clusters, i.e., 40 nodes, can be directly interconnected through a single back plane with 4 crossbar chips on it. Larger systems require an additional central crossbar switching facility, to interconnect the clusters. Figures 6 depicts a 40 node configuration. Systems with more than 16 clusters, practically of any arbitrary size, could be built by applying the hierarchical crossbar topology recursively, i.e., have clusters of clusters of clusters, and so on [Mon 91].

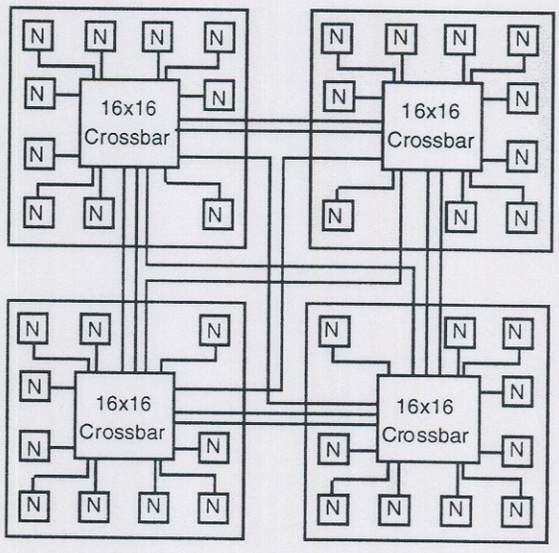

Figure 6 MANNA computer with 40 nodes

Simulations proved that the hierarchy of crossbar topology of MANNA has the same low blocking probability as the hypercube [GaM 91]. However, whereas the hypercube requires for each node (log N) links into the interconnect and, thus, is not affordable in massively parallel systems, the MANNA topology

requires only one link per node. Hence, the hierarchy of crossbars combines excellent connectivity (low blocking probability, minimum number of hops) with utmost economy. In fact, it is significantly less expensive than the 2D-mesh (which has a bad blocking behavior). Figure 7 shows the blocking probability for three types of 1024x1024 interconnects as function of the number of simultaneous connection requests. As the figure demonstrates, the blocking behavior of the crossbar hierarchy is almost as good as that of the hypercube, and both behave much better than the 2D-mesh. The crossbar is a dynamic topology with a high switching complexity of $O(n^2)$. However, that costs only silicon (the byte-wide 16x16 crossbar is a single compacted gate array) and not cables and connectors.

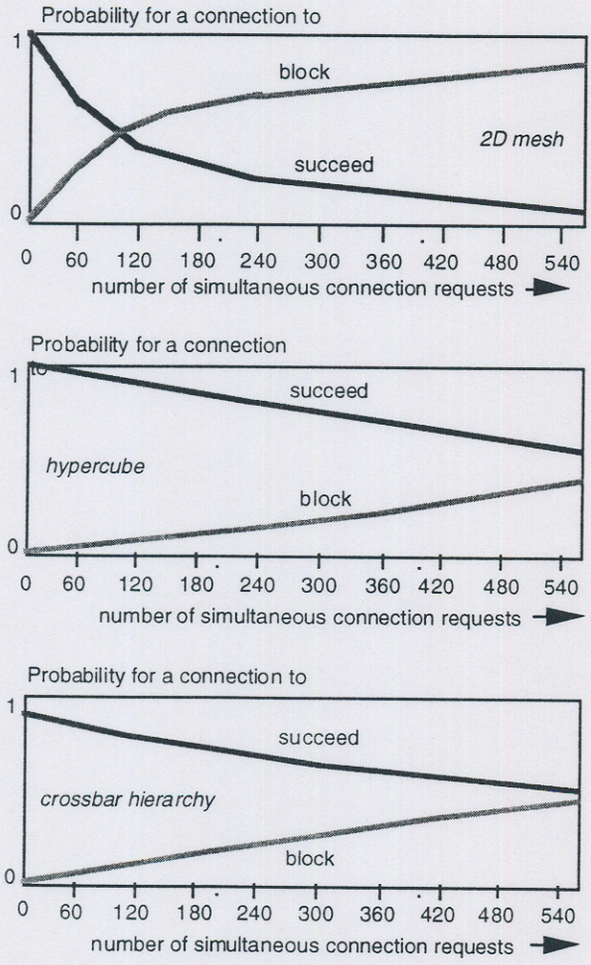

Figure 7 Blocking behavior of mesh, hypercube, and crossbar hierarchy

6. PERFORMANCE FIGURES

It should be expected that the high memory bandwidth of MANNA and the high connectivity and transmission bandwidth of its interconnect results in an excellent numerical processing performance. The 1000x1000 Linpack performance of the 1-processor MANNA node, obtained with the *Portland Group* Fortran compiler and PVM, is 18 MFLOPS. However, this tells only something about the processor and the compiler. A better indicator for the performance of the overall *architecture* is is the speed-up with the number of processors. This measurement is shown in Table 3 in comparison to the Intel iPSC860 and the Meiko CS-2.

# of Nodes	MANNA	iPSC860	CS-2
2	1.96	1.68	1.88
4	3.70	2.42	3.20
8	6.72	3.31	4.79
16	11.35	4.22	6.12

Table 3 Linpack speed-up (1000x1000 elements)

For the inner product of two matrices with 100x100 elements the sustained performance of the 2-processor node is 68 MFLOPS d.p., at a theoretical peak performance of 100 MFLOPS, respectively. The 2-processor node performance in ray tracing amounts to 600 000 polygons per second.

7. VIRTUAL SHARED MEMORY

7.1 Adaptive Consistency

Another major goal of the project concerned the development of a highly efficient virtual shared memory (VSM) realization. The goal was to achieve acceptable performance by appropriate hardware and OS support, to make VSM practically viable. MANNA VSM implements the usual mechanism where pages of a virtual memory migrate to those nodes where they are needed [Li 86].

The following concepts enhance the efficiency of VSM on MANNA:

- the adaptive consistency model [Gea 91], allowing for multiple writes during the computation phases of data parallel algorithms without the danger of semantics violations;
- the support of the VSM mechanism by a specific VSM kernel of PEACE;
- the strong support of the VSM-related OS functions by the 2-processor node architecture.

Adaptive consistency is a variant of *realease consistency* [Gea 90] which works particularly well in connection with the common SPMD (single program—multiple data) mode of execution. During

the computation phases, the nodes work without observing states of objects in other nodes. Therefore, the system may work in the MRMW (multiple readers—multiple writers) mode, as long the system merges the diverging copies into one consistent object before entering a communication phase. During a communication phase the system works in the MRSW (multiple reader—single writer) mode.

The programmer can make the following use of the adaptive consistency mechanism. By executing a *define_local* system function a thread receives the unconditioned write capability for its local copy. The write access to the copy is confined to the node, i.e., does not affect the copies of the same object in other nodes. Thus, threads in different nodes may write into the same page, each one into its local copy. The only restriction is that the threads are not allowed to write into the same location in the page, i.e., the write addresses must be mutually exclusive. For example, in array processing each writer may contribute a row or a column to a result matrix. This mechanism avoids the page thrashing problem [Li 86].

At the end of the write phase, one of the writers—it does not matter which one—executes the function *define_global*. This has two effects: (1) the system will automaticaly unify the local copies into a single global object whose content is the union of the local changes, while all local copies are invalidated and (2) the thread that executes the *define_global* function becomes the new owner of the unified copy. The unification is performed by a fast hardware algorithm [Gea 91]. The order in which the copies are merged is arbitrary. The merging procedure can be pipelined or executed in parallel. This mechanism is provided by OS functions executed on the CP [Cor 93]. Consequently, its overhead is hidden from the application.

7.2 Consistency Manager

In VSM the nodes must know the owners of the shared objects; and the owner of an object (the writer) must know who else has a copy of it. This information is furnished by a compiler-generated *access list* containing the identifiers of all the sharers. In the VSM kernel of PEACE, access lists are maintained by a service called *consistency manager* [Cor 93]. Under normal conditions a consistency manager never changes its location; therefore, the page tables need not be updated after a change of ownership. Since the consistency manager has a copy of the object, it can directly satisfy requests for copies, without having first to go to the owner. This saves a significant amount of message traffic, and it allows the system to destroy obsolete processes without destroying the shared objects they own.

8. PROGRAMMING MANNA

8.1 Conventional Programming Environment

A programming enviroment for application software production is provided on MANNA by the *Portland Group* Compilers for C and Fortran, enhanced by parallelization tools such as PVM and MPI. The elaborate environmental control simulation systems running on MANNA were programmed in that environment. In addition, there exist several research-grade programming systems for automatic or interactive parallelization by compiler. These are:

- SNAP: an experimental, automatically parallelizing Fortran compiler:
- parLisp: an interactive commonLisp parallelizer;
- PROMOTER: compiler and runtime system for the PROMMOTER programming model.

8.2 The SNAP Compiler

The automatically parallelizing Fortran compiler *SNAP* has been developed to test the feasibility of such a tool [Hae 93]. SNAP has front ends for Fortran77 and Fortran90. Hence, it allows the user to write sequential programs that are automatically translated into an optimized parallel version. Program partitioning, data distribution over the nodes, and communication between the parallel executing program units all is optimally performed by the compiler. SNAP works for data parallel applications, reducing the optimization problem to finding a data distribution that minimizes the cost of communication. Optimization is performed by genetic algorithms.

8.3 ParLisp

parLisp [Sod 95] is an interactive parallelizer for programming in commonLisp symbolic applications, e.g., computer algebra, expert systems, theorem provers. Symbolic applications typically do not exhibit much data parallelism; however, there may be program parallelism (e.g., in tree searches) that are worth exploiting. In such applications there exist typically a number of problem solving paths with varying depth. Path creation may depend on the actual data, i.e., occur at run time; thus, the compiler cannot create them. Another difference to numerical problems is that the underlying data structures usually are pointer lists that are created also at run time. Despite these problems, it is possible to recognize and exploit parallelism interactively. Any number of parallel processing paths can be created, and the results of

their execution can be reported at any time. parLisp utilizes MANNA´s VSM architecture which enables it to securely handle pointer structures. In all cases where the uniqueness of the data objects is of no concern, explicit copies may be exchanged for which unrestricted read and write capabilities are granted.

The probably biggest challenge in symbolic parallelization is to find solution steps that are large enough to balance the cost of communication of the distributed memory architecture. To this end, parLisp provides rules and tools for granularity control. The user is assisted in program partitioning by *profiling* tools by which time estimates are obtained for the individual steps, as well as information about the number of loop iterations or the depth of recursion, respectively.

8.4 PROMOTER

PROMOTER (programming model to enable real-world computing) [GKS 95] allows the programmer to explicitly express his or her knowledge about the logical spatial structures of parallel applications. This enables the compiler and runtime system to automatically perform mappings and aggregations, thus sparing the user this tedious and error-prone task. Application universality is achieved through graph structures as interface between application and implementation. Graphs constitute a flexible and convenient, architecture-independent framework for parallel applications. Consequently, domains may be arbitrarily defined; hey may be regular or irregular, static or dynamic. Graphs in PROMOTER also furnishing explicitly the information needed for optimized mapping.

9. CONCLUSION

Through a host of innovative architectural concepts, the scalable distributed memory MANNA architecture excels other, comparable machines significantly in performance. Easiness of programming is not paid for with sacrifices in cost-effectiveness and/or performance but provided through appropriate parallel programming tools and parallelizing compilers. An exception is the global address space programming paradigm provided through VSM which, of course, carries some (minimized) performance penalty.

A number of MANNA computers with a total of 160 nodes were built. MANNA has become the main "work horse" at GMD FIRST for "grand challenge" applications such as the simulation of summer smog or other air pollution control tasks. MANNA is also the development platform for the high-level PROMOTER programming system, a project of the Real World Computer Program. Last but not least, a couple of MANNA computers have been installed at the site of major cooperation partners of FIRST, e.g., at the Fraunhofer Institute for Graphical Information Processing at Darmstadt, Germany (where it has become their most powerful tool for virtual reality applications). Another MANNA computer has been installed at McGill University at Montreal, Canada (where it is used as the development platform for multi-threaded architecture concepts and software).

A major step towards the further improvement of the MANNA is the development of *powerMANNA* currently conducted at FIRST. The powerMANNA architecture is the one described in this paper, with the Intel processor i860 being replaced by the Motorola/IBM powerPC 620. The bandwidth of the node memory and the interconnect has been adequately increased.

REFERENCES

[Bea 93] Beguelin A., Dongerra J., Geist A., Manchek R., Sunderam V.: A User's Guide to PVM — Parallel Virtual Machine, available from netlib@ornl.gov by the message: send pvm_shar from pvm

[BGS 94] Bruening U., Giloi W.K., Schroeder-Preikschat W.: Latency Hiding in Message-Passing Architectures, *Proc. IPPS '94*, IEEE-CS Press 1994

[Cor 94] Cordsen J.: Basing Virtually Shared Memory on a Family of Consistency Models, Proc. 1st Internat. Workshop on Scalable Shared Memory Systems, 1994

[Eea 92] von Eicken T., Culler D.E., Goldstein S.C., Schuster K.E.: Active Messages : A Mechanism for Integrated Communication and Computation, Tech. Report UCB/CSD 92/675, University of California at Berkeley 1992

[GaB 81] Giloi W.K., Behr P.: An IPC Protocol and Its Hardware Realization For High-Speed Distributed Multi-computer System, *Proc. 8th Internat. Sympos. on Computer Architecture*, IEEE Catalog No. 81CH1593-3, 481-494

[GaM 91] Giloi. W.K., Montenegro S.: Choosing the Interconnect of Distributed Memory Systems by Cost and Blocking Behavior, Proc. 5th Internat. Parallel Processing Symposium, IEEE Catalog no. 91TH0363-02 (1991), 438-444

[GaS 89] Giloi W.K., Schroeder W.: Very High-Speed Communication in Large MIMD Supercomputers, *Proc. ICS '89*, ACM Order No. 415891, 313-321

[GaS 93] Giloi W.K., Schramm A.: PROMOTER — An Application-oriented Programming Model for Massive Parallelism, in Giloi W.K.,

Jaehnichen S., Shriver B.(eds.): *Massively Parallel Programming Models*, Proc. Internat. MPPM Conference 1993, IEEE-CS Press 1993, order no. 4900-02, 198-205

[Gea 90] Gharachorloo K., Lenoski D., Laudon J., Gibbons P., Gupta A., Hennessey J.: Memory Consistency and Event Ordering in Scalable Shared-Memory Mulriprocessors, *Proc. 17th Annual Sympos. on Computer Architecture*, IEEE catalog no. CH2887-90, 15-26

[Gea 91] Giloi W.K., Hastedt C., Schoen F., Schroeder-Preikschat W.: A Distributed Implementation of Shared Virtual Memory with Strong and Weak Coherence, in Bode A.(ed.): *Distributed Memory Computing*, Proc. EDMCC2, LNCS 487, Springer-Verlag 1991, 23-31

[Gil 84] Giloi W.K.: Obtaining a Secure, Fault-Tolerant, Distributed System With Maximized Performance, in Reijns G.(ed.): *Hardware Supported Implementation of Concurrent Languages in Distributed Systems*, North Holland, Amsterdam 1984

[GKS 95] Giloi W.K., Kessler M., Schramm A.: PROMOTER: A Programming System for High-Level Massively Parallel Programming With Arbitrary Appliccation Topologies, Proc. RWC Symposium 1995, RWC Partnership, Tokyo, June 1995

[Hae 93] Haenisch R.: SNAP! Prototyping a Sequential and Numerical Application Parallelizer, *Proc. Internat. Workshop on Automatic Distributed Memory Parallelization, Automatic Data Distribution, and Automatic Parallel Performance Prediction* (March 1993), Springer WICS

[HPF 92] High Performance Fortran Forum: High Performance Fortran — Language Specification (DRAFT), Version 0.4, November 1992

[Li 86] Li K.: Shared Virtual Memory on Loosely Coupled Multiprocessors, PhD thesis, Yale University 1986

[LST 92] Linden J., Schüller A., Trottenberg U.: Methodological Aspects of High Performance Scientific Computing, in Sydow A.(ed.): *Computational Systems Analysis 1992*, ELSEVIER, Amsterdam 1992, 1-10

[Mon 89] Montenegro S.: Kommunikationsstrukturen für verteilte Rechnersysteme, PhD thesis, Technical University of Berlin 1989

[MPI 93] Message Passing Interface Forum: Document for a Standard Messdsage-Passing Interface (DRAFT) (Sept. 1993)

[Sch 91] Schroeder-Preikschat W.: Overcoming the Startup Time Problem in Distributed Memory Architectures, in Milutiniovic V., Shriver B.(eds.): *Proc. 24th Hawaii Internat. Conf. on System Sciences, vol.1*, IEEE Society Press 1991, IEEE catalog no. 91TH0350-9, 551-559

[Sch 94] Schroeder-Preikschat W.: *Logic Design of Parallel Operating Systems* , Prentice-Hall, Englewood Cliffs NJ 1994

[Sod 95] Sodan A.: Mapping Symbolic Problems with Dynamic Tree Structures to Parallel Machines, Ph.D. thesis, Technical University of Berlin 1995