Scientific Programming on SUPRENUM

B. Thomas
SUPRENUM GmbH
Bonn, Germany

Summary: SUPRENUM software (distributed operating system services, compiler, libraries, tools) support a convenient, communicating-processes based programming model for distributed memory MIMD systems. The concepts will be presented along with examples and additional tools that facilitate development and understanding the behaviour of a parallel application.

1 Introduction: User Requirements in Scientific Programming

Various scientific applications quoted at Gigaflops performance have recently demonstrated the supercomputer potential of medium to large scale distributed memory parallel computers (Addison 1990). It is also widely accepted on theoretical grounds that future top level performance will necessitate distributed memory (DM) systems. Nevertheless, general migration from conventional supercomputers to parallel architectures appears rather slow in the scientific community.

A major reason for that is clearly that with parallel architectures optimal performance of an application relies on a tuning to a multitude of factors, such as shared vs distributed memory, interconnect system and topology, scale of parallelism, processor architecture and performance, and, even worse, on programming style and languages. Optimal choice is required, or optimal utilisation of a given choice, to be made by the application programmer, whereas on sequential systems the user is hardly concerned with anything beyond fast and stable algorithms.

Arising from the conventional view are several key demands to parallel scientific programming:

(i) The application must have sufficient parallelism to allow for an efficient exploitation of a multi-processor in principle. This has in fact been shown for many appliciations, including weather models, CFD and discretised simulations in general (see e.g. Hoffmann, Maretis, 1990).

(ii) Exploitation of both the algorithmic parallelism and of the performance potential should be as easy as possible. This bears over to the programming and to the use of a parallel system.

(iii) If re-working on an application is inevitable then the new software should not again be machine dependent. Thus, portablility within and between classes of machines is the goal and has to be supported.

(iv) After all, supercomputer performance is the main quest, and sacrifices must be kept as small as possible.

Fig.1 indicates how, roughly, these user demands map onto requirements to the layers of a full parallel application environment. Exploitation of parallel potential requires the numerical model to be redesigned adequately, which is not in the responsibility of the parallel computing environment. It is, however, with respect to providing an adequate and convenient high level programming model, and supporting it by appropriate means to express it this way, i.e. by an appropriate programming language. The demand for portability of application programs is mainly answered by ensuring that the language is widely spread and in use, and by providing a system interface which hides system software and hardware idiosyncrasies from the application programmer. Nevertheless, performance requires fast system software to make use of the hardware features in optimal way with little overhead. For complete user satisfaction (see above) a system development has to attack all these requests simultaneously - at least as long as optimal solutions to parts of it have not been identified in general, and standards based upon these have not yet emerged.

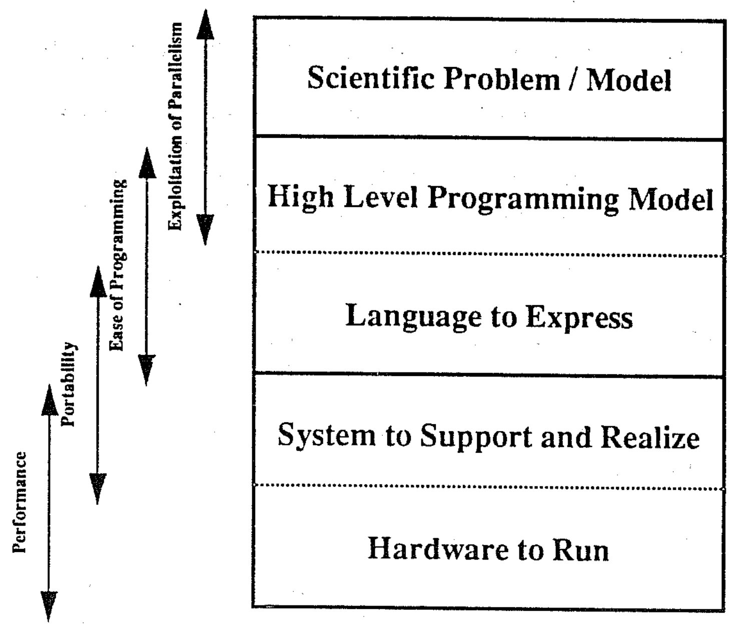
Exploitation of Parallelism
Ease of Programming
Portability
Performance
Scientific Problem / Model
High Level Programming Model
Language to Express
System to Support and Realize
Hardware to Run

Fig. 1 Mapping User Demands to System Requirements

Within the SUPRENUM project, from which SUPRENUM-1 emerged as a first product, all levels of the above have been included. A range of applications have been selected and parallelised to demonstrate that there is enough parallelism in many application areas to make a parallel version worth while. This has been described on various earlier occasions (e.g. Solchenbach, Thole, 1990).

The SUPRENUM programming model has been defined in terms of an abstract machine concept, and Fortran - with comfortable MIMD extensions - has been chosen as primary application programming language to facilitate writing and porting of parallel applications. With a set of tools and libraries, and a UNIX^{TM} user environment SUPRENUM addresses the needs for ease of programming and portablility. These components will be described in more detail in this paper.

Performance requirements are met by the scalable hardware, with up to 256 vector processing nodes at 20 MFlops peak (IEEE double precision, chained mode) accommodated in clusters of 16 nodes and interconnected by a two-level bus system. A low-level operating system kernel (PEACE) provides for optimised services to be requested by an application program w.r.t. process execution and communication. Details have been presented elsewhere (e.g. Giloi 1988 , see also contribution of O. McBryan, this volume) and will not be repeated here.

2 SUPRENUM Programming Model

Programming for MIMD-parallel systems requires a suitable programming model to design an application in and languages to formulate it. Usually the two are closely linked, and there have emerged several styles in parallel programming varying typically in the granularity of parallelism and the degree of explicity of parallel threads and communication.

With respect to these features, the SUPRENUM programming model is clearly originated in scientific computing:

- A parallel program consists of autonomous *tasks*, i.e. program units
- Parallel *processes* are created from tasks dynamically:
    - An application is started by an *initial process*
    - Processes can create new processes at any time from tasks; process termination is implied by the end of a task
    - Termination of the initial process ends the whole application
- Processes own a private data space. Inter-process communciation of data objects is done by message passing
- Arbitrary process structures are allowed independent of the underlying hardware
- Vector processing can be used within tasks.

This programming model is explicitly supported by SUPRENUM's system software and, from the programmers point of view, by SUPRENUM Fortran as primary programming language.

3 SUPRENUM Fortran

The main new features in SUPRENUM Fortran are indicated in Fig.2. A parallel program consists of a set of independent, regular main programs which are declared *task programs.* They are the textual units from which processes will be generated at runtime. There is only one *initial task* program which gives rise to the one initial process, and there may be one or more task programs. Process creation can occur wherever needed and is programmed for by using the NEWTASK function, which takes the name of a task program and returns the identification for the newly created process (data type TASKID). Multitudes of processes can of course be created running the same task program.

Communication between processes is programmed in terms of SEND and RECEIVE constructs, which imply asynchronous message passing semantics. Messages are specified as usual I/O-lists, can be identified by TAGs, and address the communicating processes by their TASKID. There are several other features within SEND and RECEIVE for optional use that make them very flexible language extensions. The WAIT statement is used to formulate alternative branches depending on whether a specified message (or set of) has arrived. A few additional functions are included to inquire about messages and processes.

A subset of the F90 extensions for vector and array processing has been included in SUPRENUM Fortran. It includes definition and manipulation of array objects of all sorts, array properties, array subscripts, dynamical allocation, conditional operations and intrinsic vector/array functions. Using the new array notations makes programming more object-oriented, avoids extensive loop programming and, after all, is a convenient way to directly influence vector processing in SUPRENUM, as an alternative to relying on the *vectorizer* which is included in the compiler.

Program structure	INITIAL TASK PROGRAM, TASK PROGRAM
New data type	TASKID TASK EXTERNAL
Process initialization	TASKID-**Function** NEWTASK**(tprogname, cpu)**
Communication	SEND **(tag, address_id, ...) i/o-list** RECEIVE **(tag, sender_id, ...) i/o-list** WAIT **(tag, sender_id, label, ...)** ... CONTINUE
Vector processing	**FORTRAN 90 constructs**

Fig. 2 SUPRENUM Fortran: New Features

The sample task program gives a flavour of SUPRENUM Fortran programming. For a full specification see (SUPRENUM, 1989).

Comparing with other ways that exist today to formulate parallel applications, SUPRENUM Fortran has the following main characteristics:

* All definition and creation of parallel processes and communication is explicit within the language syntax. There is no "mixed-level" (e.g. OS/language/hardware) thinking required in writing a parallel application.
* Processes correspond to program units. Processes are created dynamically and are explicitly identifiable by (logical) process-id's, as opposed to e.g. the Occam model.
* Communication is explicit, addressing processes directly (by their process-id), rather than a communication path (e.g. a unidirectional channel).
* Communication is asynchronous throughout (non-blocking SEND). Efficient synchronous communication is provided by library calls.
* Messages are identifiable, which can be important for asynchronous communication programming.
* Communication may comprise structured data objects of various types. Copying into sequential buffers or byte streams is not required.
* There is no restriction to process topology.
* The syntax contains no references to hardware or system software configuration.

```
* Sample Task Program Distributed Matrix Multiplication
* Method: Distribution of block of rows and columns

        task program multiply
        taskid host, pred, suc                   ! Ring topology

        real, allocatable, array (:,:):: a, b, c

        host=master()                            ! Initial task ID
        receive (1,host) myind, pred, suc, m, nb, nw

        allocate (a(1:nw, 1:m),                  ! Dynamic Allocation
     +            b(1:m, 1:nw),
     +            c(1:m, 1:nw))

        receive (2,host) a,b                    ! Receive array sections
        k=myind

        do i=1,nb-1
              send (i+5,pred) a                  ! Send array to pred
              j1=k*nw
              j0=j1-nw+1
              c(j0:j1,1:nw)=matmul(a,b)
              receive (i+5,suc) a                ! Receive data into a
              k=mod(k,nb)+1
        end do

        j1=k*nw
        j0=j1-nw
        c(j0:j1,1:nw)=matmul(a,b)

        send(3,host) c
        deallocate (a,b,c)                      ! Release arrays

        end
```

4 Tools and Programming Aids

In response to the new challenges that parallel programming imposes on main stream scientific programming, SUPRENUM projected and developed a set of new program development aids beyond the classical ones. They either come as dedicated tools or as libraries of routines.

4.1 Libraries

Several libraries are provided to support portability, or transparency of the underlying parallel system.

Linear Algebra Library (SLAP). SLAP contains distributed memory versions of routines from the classical packages LAPACK, EISPACK and of BLAS-2 and -3. The routines are numerically equivalent to the classical ones. They follow the standard calling convention and make use of the ANL/GMD Macros to ensure machine independency to a high degree. The library is supplemented by routines that take care of appropriate data distribution and minimized communication overhead.

ANL/GMD Macro Library (PARMACS). Portable programs for parallel systems require a straightforward, transparent programming model. SUPRENUM Fortran supports such a programming model including features that are not generally available on other systems which could in principle be programmed under a very similar model. The Macro Library interfaces application programming between a variety of parallel computers. Already implemented on a selection of both shared and distributed memory systems it provides the SUPRENUM programming model even slighlty extended by including synchronous communication and global synchronisation "primitives". A detailed discussion can be found in Bomans, Hempel, 1990.

Communication Library (SCOMLIB). An even higher degree of abstraction from the parallel computing environment is supported by the Communication Library. Fortran programs can be written completely without MIMD extensions, as introduced e.g. in SUPRENUM Fortran. All process and communication management is hidden in subprograms. SCOMLIB is currently dedicated to a wide class of 2D and 3D grid-based applications and typically comprises routines for generation of process grids, data exchange across inner boundaries and global operations and data communication (Hempel 1987, also Hempel, this volume). Using Scomlib makes parallel programming easy and safe; applications will run on any machine that has the communication library implemented onto it.

Mapping Library. Generating a class of standard process structures like rings, grids, trees etc. and placing them onto processing elements is facilitated by calls to a Mapping Library. General topologies (graph structures) can be defined by an appropriate adjacency matrix. Usually the actual placement of processes onto processors is automatic; however, there are means to control this if the programmer wishes to do so.

4.2 Tools

Simulator. Since the early days of the project, application programming for SUPRENUM was advancing very fast even without the system already available. Instead, a SUPRENUM Simulator (SUSI) was used, which implemented the SUPRENUM programming model on remote Unix workstations, based on a Fortran with the SUPRENUM MIMD extensions. A parallel application, usually in test model size rather than in full problem size, could be run in the Simulator environment as if on the real hardware, giving comprehensive traces of the behaviour of system components (processes, processors, busses etc.). SUSI comprises a graphical "hardware" configuration tool which allows for specification of a parallel system to run (simulate) the application on. It also provides a comfortable user interface to interactively control the run of the simulation. (Tietz, 1987)

Whereas SUSI is a suitable tool to take first steps, particularly towards the MIMD aspect, in parallel application software development for the given programming model, remote software development that intends to take full account e.g. of SUPRENUM Fortran features such as the F90 extensions preferably uses the SUPRENUM Emulation Environment (SUPREME). This includes the central system components such as PEACE or the VFPU as local emulation on Sun^{TM} workstations, has the SUPRENUM Fortran compiler and vectorizer intergrated, and is operated under the usual SK User Environment. Thus, there is no difference in working locally under SUPREME or being actually connected to a SUPRENUM system, except for execution time, of course (SUPRENUM, 1990).

Where SUSI generates its own process traces upon request, the SK User Environment provides a State Server, to be loaded together with the application by a simple request. The State Server automatically reports relevant events such as creation and loading of a process, sending and receiving a message. No provisions have to be made for that in the program, however. As is unavoidable in principle, such "measurements" slightly affect the behaviour of the application process system proper. Also, this effect is different for a real system or for the emulation. As a consequence, the event times reported particularly under the emulation constitute a logical picture rather than sharp measurements. This is nevertheless found very valuable for analysing and optimising a parallel application.

Graphical Performance Analysis Tools. A set of communicating processes as it is, a parallel application deserves some consideration of the internal interactions. This is definitely a new feature to parallel programming as opposed to the sequential style, where the programmer has to understand the optimised numerical algorithm in first place and otherwise rely on the speed of a given processing device.

In parallel supercomputing an optimal scheduling, or overlapping, of communication and computation can result in considerable performance gains. Thus, if an application programmer is willing to, he or she can analyse performance of a parallel code by inspecting the behaviour of the process system. SUPRENUM provides a set of tools which generate comprehensive graphical presentations of process events to facilitate such an inspection (Thomas et al., 1988).

As input these tools require a stream of events in a uniform format which come from e.g. the State Server or the trace information of SUSI automatically, or any other suitably implemented facility. Relevant events comprise creation and termination of a process, sending or casting a message, message arriving in a process' mailbox, receiving a message (from mailbox), blocking and re-activating a process. Events may typically contain information about time (wall clock), type of event, processes and processors involved, and message id. Global time is of course a problem, since different processes on different processors can provide local time only. In the current SUPRENUM software environment the Sate Server automatically generates the information required to obtain a best guess for mapping local to global time. A verbose account of the behaviour of the process system can be obtained from an event stream on ASCII terminals.

On colour graphic terminals, the tool called DynamicMap produces an animated view of the activities of processes, processors, mailboxes, and of the communication channels (interconnect system) along with information on sent-out or incoming messages. At any time the presentation gives a full picture of the state of the whole process system in user relevant terms. The animation can be slowed down to follow through the actions of the system, e.g. right before a critical situation.

The TimeMap produces a comprehensive temporal overview of the behavour of the complete application. It displays a recording of states of processes and associated mailboxes as well as of process creation and inter-process communication over the full time interval. The graphics provide a first glance of the communication schedule and active-to-inactive time ratio of processes. Analyzing the display in more detail and comparing different runs can provide useful hints to inefficient organization of the communication and lead to improved tuning of the application. Critical sections can be identified and subjected to more intense debugging or user supplied exploration.

The StatisticMap displays statistics obtained from the reported data as e.g., load of processors, frequency of communication events total and between process pairs, occupation of mailboxes and time spent on message passing activities.

4.3 Semi-automatic Parallelizer

Still a challenging research topic, automatic transformation of sequential programs for MIMD execution on distributed memory architectures was attacked in the SUPRENUM project with respect to parallelization of grid-oriented applications. A prototype of the interactive system SUPERB has been developed at the University of Bonn (Zima et al., 1988) for semi-automatic transformation of Fortran 77 programs into parallel SUPRENUM Fortran programs. SUPERB comprises a powerful analysis component, a set of MIMD and SIMD transformations and a flexible dialog system.

In principle, there are no restrictions to the type of programs SUPERB can handle, since it requests help from the user and offers dependence information and transformation methods interactively. Currently, however efficient MIMD-parallelization works best on grid-like data domains with computation at grid points imposing local data dependency. Work on this type of approach for making sequential programs run on parallel machines continues, in part under the Esprit project GENESIS.

5 Conclusion: Performance and Comfort - a Trade-off

Today, advancements in software technology for sequential and vector computers make it safe to expect that high level programming and portable software development will end up in optimal code for those types of machines. It is even expected that compiler technology will soon be able to make the power of advanced processors such as the Intel or IBM superscalar processors fully available to application programmers in high level languages. The reason is that understanding of sequential and vector processing and also of fine grain concurrent processing on a common address space is quite developed today, and has improved software technology over years.

Large scale parallel computing brings in new issues, viz. distributed data and code, and the need to handle communication. These appearently are much harder to be integrated into general, automated software support. Currently, there is no better way to real optimized efficiency than to get involved in the aspects of parallelism at a rather low level with the individual application problem and the individual machine. Support aids do exist, as SUPRENUM demonstrates. They usually address certain key aspects such as communication programming, portability across parallel machines, parallelization support for a class of problems etc. They clearly add to the comfort of parallel programming, assist in porting existing codes and, after all, increase acceptability of parallel supercomputing in the scientific communitiy.

On the other hand, transparency of the underlying parallel hardware/software system to the programmer has to be paid for by a loss in performance. For example, communicating data structures between processes in a DM system by means of a comfortable SEND statement as in SUPRENUM Fortran induces considerably higher message start-up as compared to sending a stream of bytes by low level functions. Today, a good strategy to utilisation of parallel systems is first to get an application running with moderate to good performance (compared to peak expectations) under comfortable conditions and with moderate to low effort. Then, getting to understand the behaviour of the parallel application in conjunction with the key features of the system the programmer might choose to gradually implement optimisations. As an example, if, at certain points, synchronous communication is required only, the programmer might introduce calls to low level synchronous communication routines and thus avoid typical overhead required for asynchronous message passing (e.g. buffering, mailbox management). With respect to the software environment of a parallel supercomputer this means to offer "levels of transparency" to the programmer, which may reach from global automatic parallelization, via the use of high-level libraries, language extensions, library interface to the operating system, down to accessing OS or even hardware functionalities directly.

References

Addison, C., 1990: Integrated benchmark and evaluation report: Overview document. Internal Report A2xD2/1, Esprit Project 2702 (GENESIS).

Bomans, L., Hempel, R., 1990: The Argonne/GMD macros in Fortran for portable parallel programming and their implementation on the Intel iPSC/2. Parallel Computing 15.

Giloi, W., 1988: Suprenum, a trendsetter in modern supercomputer development. Parallel Computing 7/3.

Hempel, R., 1987: The SUPRENUM communications subroutine library for grid oriented problems. Report ANL-87-23, Argonne National Laboratory.

Hempel, R., 1991: Argonne/GMD macros and SUPRENUM communication library: tools for portable parallel Fortran programming.

Hoffmann, G.-R., Maretis, D.K., (ed.) 1990: The dawn of massively parallel processing in meteorology. Springer, Berlin

McBryan, O., 1991: Effort and reward: comparison of parallel implementations of a weather model. This volume.

SUPRENUM, 1989: SUPRENUM-Fortran Reference Manual. SUPRENUM GmbH, Bonn.

SUPRENUM, 1990: SUPRENUM - Getting started. SUPRENUM GmbH, Bonn.

Solchenbach, K., Thole, C.-A., 1990: The SUPRENUM architecture and its application to computational fluid dynamics. In Hoffmann, Maretis, 1990 (ed.)

Tietz, Ch., 1987: Das Benutzer-Interface des SUPRENUM Simulationssystems. GMD Report, St. Augustin.

Thomas, B., Thomas, E., Truchet, E., 1988: SUPRENUM visualisation tools for distributed applications - Users guide. SUPRENUM Report 12, Bonn.

Zima, H., Bast, H.-J., Gerndt, M., 1988: SUPERB: a tool for semi-automatic MIMD/SIMD parallelization. Parallel Computing 6.