Software Environments for Scientific Modelling on Parallel Computers: Summary of the Discussion

TUOMO KAURANNE

European Centre for Medium-range Weather Forecasts, Shinfield Park, Reading, Berkshire RG2 9AX, United Kingdom

1 Introduction

The previous three ECMWF Workshops on the Use of Multiprocessors in Meteorology have discussed the problems presented by implementing atmospheric models on parallel computers. One of the principal conclusions has been that most of the problems posed by the hardware and by the parallelization of algorithms seem surmountable, whereas the status of software environments for parallel computers still leaves much to be desired. Programming tools for distributed memory parallel computers are particularly inadequate.

It was therefore deemed appropriate to lay emphasis in the fourth workshop on programming aspects of massively parallel, distributed memory computers. It seems that this particular computer type is the most likely candidate to become the paradigmatic supercomputer of the late 1990's, the same way as the vector processor became the first specialized supercomputer in the late 1970's.

The public discussion that concluded the workshop dwelt also on hardware and operability issues, in addition to various aspects of software problems on massively parallel computers. The participants represented a very wide spectrum of interests, including operational weather forecasting, climatological research, particle physics research, research in parallel algorithms, research

in parallel languages and parallel computer manufacturing.

2 User requirements

The meteorological users are facing a difficult dilemma. On the one hand, there is a strong need for substantially more computing power than is available on present supercomputers. An estimate of performance requirements in weather forecasting and climatology was presented. According to this estimate, meteorological and climatological simulations would need a sustained performance of 700 Gflop/s by 1994-1995. It represents a factor of one hundred over fastest existing supercomputers.

On the other hand, existing model codes represent considerable capital investment for the forecasting and research centres. Model codes will generally outlive supercomputers. Therefore, radical recoding to implement a model on a particular supercomputer is very undesirable. An estimate put forward in the discussion was that there must be convincing evidence of a ten-fold gain in sustained performance from massively parallel supercomputers over vector supercomputers before forecasting centres get interested in starting to convert their operational codes onto novel computers.

The inevitable capitalization on forecasting systems goes beyond mere programming concerns. ECMWF disseminates almost 3000 products a day, straining heavily the operational and communication capacities of supercomputer systems. It is also difficult to predict the nature of forecasting systems - as well as the state of the supercomputer industry - more than 10 years ahead, making very long term plans meaningless. Radical changes in the computing environment also imply losses of economy in system maintenance and staff allocation. The centres would ideally want to concentrate exclusively on weather forecasting and on research, with a minimum of involvement in computer wizardry. Since climatological models require a finer granularity of parallelism than weather models, because of a lower spatial resolution, code level problems posed by massively parallel computing in climatology appear to be even more pronounced than those in forecasting.

A necessary prerequisite to bring at least some continuity into the process of adopting massively parallel computers into atmospheric modelling would be the executability - albeit with a low efficiency - of existing model codes on novel computers. This would at least provide a bottom line from which to

start working upwards, instead of a total jump into the dark when starting the migration effort.

3 Performance expectations

Substantial improvements in supercomputer performance are expected to be achieved at a brisk pace. It is believed that supercomputers with 20 Gflop/s peak performance will be available in 1991-1992, whereas the years 1993-1995 should see the emergence of supercomputers with a peak performance in the 100-200 Gflop/s range. But this kind of performance will not be achieved on all applications. Whenever the cost of performance is a critical issue, low communication algorithms become mandatory, since global communication bandwidth is very expensive on ultrafast supercomputers. Employing low communication algorithms requires a "model engine" approach to supercomputing. In this approach, massively parallel computers are dedicated to running a single or at most a few model codes.

It seems therefore that massively parallel supercomputers are initially more likely to be employed as back end computing servers than in standalone configurations. On the other hand, a slow front-end is a poor companion to a powerful supercomputer; the front-end will also have to be reasonably powerful, possibly in the same class as present vector supercomputers.

It was also suggested that general purpose massively parallel computers and, in particular, very competent general purpose parallel compilers would be feasible in the long term. This prospect was not unanimously accepted. There was general agreement, though, that 700 Gflop/s by 1995 would not be feasible without massively parallel computing.

4 Benchmarks

Some other fields in computational fluid dynamics - such as aerospace engineering - have some widely recognized benchmark codes. This is not the case in meteorological supercomputing, despite a definite need for such.

Massively parallel computer manufacturers find it unfair that the very next step from Linpack-type simple benchmarks up is often a full model code presented as a mandatory requirement in supercomputer acquisitions.

On the other hand, accepting supercomputers on the basis of any single simplified meteorological benchmark would imply many problems for operational forecasting centres. Among these problems are the differing opinions among the forecasting centres about what kernels are important. The representativeness of simple codes is heavily dependent on architecture: novel architectures create novel bottlenecks. Benchmarking methodologies vary. And, finally, it is virtually impossible to base supercomputer purchase in an operational environment on simplified benchmarks alone; there has to be guaranteed performance levels for the operational model.

It is difficult for operational users to invest in novel hardware. Some centres, especially those with a research mission, can upgrade their computing facilities in a piecemeal fashion. The very big centres like CERN even integrate hardware components to construct their own supercomputers, as does one meteorological research centre (Wisconsin). For most meteorological research institutions, however, constructing their own computers would move a great deal of limited staff resources out of focus and consequently such endeavours are difficult to finance. Big research centres can have several teams out simultaneously with a mission to plan for future local computing environments. The time scopes of these teams may vary between two and twenty years. Forecasting centres can afford to have at most two teams, both with a temporal focus not exceeding five years.

Nevertheless, it was felt that a basis for collaboration between users and vendors could be found in the benchmarking field. If experimental hardware procurement is a problem, computer manufacturers can provide important users with access to hardware. Users, on the other hand, would contribute a man-power investment towards making a set of relevant benchmarks run on novel computers. These benchmarks would be agreed between major meteorological research centres and would most likely be at the shallow water equations level. This still means quite a lot of work for somebody, but it seems that there may be some funds available to commence work on this.

5 Interconnection topologies

Recently, some prominent massively parallel computer manufacturers have moved away from hypercube-like rich interconnection topologies, adopting simple mesh-like interconnection topologies and relying upon efficient hard-

ware routing devices and novel routing algorithms to provide communication speed and flexibility. It is sometimes difficult to map physical low-dimensional structures onto hypercubes. The large number of links needed is also expensive to realize in high-dimensional hypercubes.

From the user's point of view, it would be best not having to be concerned about the underlying physical interconnection structure at all. The big question is, whether this can be done without risking excessive communication delays in some applications. If the interconnection network does offer a sufficiently high global communication bandwidth - say, one like the memory access bandwidth on vector supercomputers - the underlying physical topology can be hidden from the user. This is, in principle, the case with hypercubes. If not, the user can often reduce communication overheads significantly by mapping his algorithmic structure onto the underlying interconnection topology in a way that maximizes locality of communication. If the underlying data flow graph is not a mesh, however, there is no way it can be mapped onto a mesh-like topology preserving locality of communication. Hence, every message has to cross several links and consume a corresponding multiple of communication bandwidth. Providing this on a mesh is essentially as expensive as on a hypercube.

6 What can be hidden from the user?

There are a number of things that cannot be hidden from the user. These include SIMD processing. The most that one would want to hide is the parallel nature of computing; in particular, the distributed storage of data structures. This amounts to implementing a virtual shared memory. In the minimum, it would be desirable to hide the access latency to a remote data object under computation.

Implementing a complete virtual shared memory is more or less the same as implementing a real shared memory on a large, distributed set of memory chips. The problems encountered in this would be very similar to those encountered in present highly banked memory architectures. It is relatively easy to implement the functionality of a shared memory on a physically distributed memory, but making it efficient seems difficult. The overhead associated with virtual shared memory may be just a factor of two to four, as was the case with the Myrias parallel computer, but it may just as well be

a factor of one thousand. On the other hand, users may just refuse to accept computers without at least a shared memory emulation.

An acceptable trade-off might be to limit the scope of a shared memory to a few large data structures. This would greatly reduce the combinatorial complexity of managing memory references. Page traffic could then be conducted at a higher level with longer messages, making more efficient use of the communication bandwidth available. This would call for standard primitives to be used in referring to distributed data structures, like imposing their storage to follow a given pattern; or accessing a remote part of a data structure while overlapping the access latency with computation. The golden rule would be to allow the largest possible degree of generality that can be efficiently implemented.

It is important to realize that implementations of message passing communications exist today on almost all parallel computers with a distributed memory. There are also some emerging standards in how to code message passing primitives into Fortran programs, whereas the users still have to wait some years before most major parallel supercomputers implement a virtual shared memory.

At a lower level, Unix is the operating system adopted as the user interface on all parallel computers. However, it is not possible to use Unix as such to code the run-time environment of a parallel computer. Most parallel computers therefore implement proprietory run-time environments that aim at being as small and light-weight as possible. This lack of standardization frequently causes problems for the users. It would be desirable to adopt some standard system calls also at the level of the run-time environment.

7 Programming environments

Programming environments on parallel computers fall into a hierarchy of increasingly ambitious and novel approaches. At the lower end, there are subroutine libraries and macro preprocessors for languages like Fortran. At their simplest, these provide only standard macro calls to communicate between processors. Sometimes macro packages come equipped with more ambitious functionality, like representing all the communication patterns typical of certain algorithm types.

Macro packages and associated Fortran preprocessors are available today,

but there is no real portability between systems yet. Therefore, one might argue, one should proceed directly to the second level in the programming hierarchy; that of Fortran extensions. This is the way adopted by most parallel computer manufacturers, since it allows to incorporate the parallelism into the compiler; the place where it properly belongs from the computer point of view. Moreover, compiler technology is gradually becoming independent of hardware development, rendering more portability to extended programming languages. From the user's point of view, however, nonstandard language extensions may even be dangerous, if similar constructs are used with different semantics.

On the highest level we find novel parallel languages. In Wisconsin, all meteorological models are maintained at a level of a Model Description Language. Novel languages are often very compact, allowing an abstract description of numerical algorithms. Their maintenance is made easier by writing the compilers themselves in high-level, object-oriented programming languages like C++, and by the use of powerful editing tools on workstations. Since most computer science graduates nowadays are educated in the object oriented paradigm, there is an adequate supply of programmers as well.

Apart from Wisconsin, however, there was a common feeling among the participants that developing and maintaining special-purpose meteorological programming languages is not feasible for most operational forecasting or research centres. There are also a number of parallel programming languages available, that are maintained by commercial software houses. But even these imply a radical change to the present programming practice. The difficulty of accepting non-standard programming tools is even more pronounced in the frequent case where several programmers and even several programming teams have to collaborate on a single, massive code.

8 Conclusions

In conclusion, it seems that there is still some discrepancy between the users' requirements and what can be provided on massively parallel computers in the field of software environments. Moreover, it is not known what kind of programming support is feasible and what is not, both from the manufacturer's and from the user's points of view. And, finally, there seems to be some lack of understanding between computer manufacturers and users,

and between different user communities, concerning the most important constraints under which each of the groups operates. These appear to be very different even from user group to a user group.

Therefore, workshops like the present one that bring together representatives from all relevant communities and keep them together in close collaboration for several days provide an important service in promoting wider acceptance of parallel computers. We hope a concensus on how the users' needs could satisfactorily be met without sacrifacing too much performance begins to emerge from the joint efforts to be put into benchmarking parallel computers by both meteorological users and massively parallel computer manufacturers. Such a collaboration will be initiated as a result of the present discussions.