Parallel Processing: A View From ECMWF

Tuomo Kauranne[1], Geerd-R. Hoffmann

European Centre for Medium-Range Weather Forecasts
Shinfield Park
Reading, Berks RG2 9AX
England

Abstract

Operational numerical weather prediction requires that all the computing power available is harnessed to the one task. Therefore, ECMWF became involved in parallel processing as soon as it became available on large computers. The main operational programs were restructured to use multitasking on a limited number of processors, currently 8. The advent of massively parallel machines, expected in the near future, raises quite a number of major problems. These areas have been discussed in the biennial workshops held at ECMWF since 1984. Besides the user environment, the inter-dependency of hardware architecture and algorithms was the area which needed most development. Recently, ECMWF became involved in the Genesis project, sponsored by the European Community. In this framework an expanding series of benchmarks is being executed on a number of massively parallel computers. These benchmarks are used as a basis for extrapolating the performance of the full model on these machines. First results from both benchmarking activities and performance predictions are reported.

1. Introduction

The European Centre for Medium-Range Weather Forecasts (ECMWF) was established in 1975 as an international organization, at present funded by 18 European countries. Its headquarters are situated in Reading, England, and its main functions are (see [Convention]):

a) to develop dynamic models of the atmosphere with a view to preparing medium-range weather forecasts by means of numerical methods;

b) to prepare, on a regular basis, the data necessary for the preparation of medium-range weather forecasts and to make these data available to the meteorological offices of the Member States;

c) to carry out scientific and technical research directed towards improving the quality of these forecasts;

d) to collect and store appropriate meteorological data;

e) to make available to the meteorological offices of the Member States for their research, priority being given to the field of numerical weather forecasting, a sufficient proportion of its computing capacity.

ECMWF runs a ten day global forecast every night. The operational nature of forecasting justifies a significant effort in achieving maximum performance from the forecast model. As a result, ECMWF has used multiprocessing ever since it became available on supercomputers. Running the operational forecast model on multiple processors started on a Cray XMP/22 in 1984 and an increasing number of processors have been employed on its successors - a Cray XMP/48 since 1986 and a Cray YMP/864 since 1990. The speedups on these machines have been 1.9, 3.7 and 7.3, respectively, as reported in [Dent].

The accuracy of a weather forecast is not dependent only on the resolution used in the numerical solution; the other two major factors are the accuracy and suffiency of observational data and the validity of the set of equations and parametrizations used in the forecast model.

All three factors contribute essentially to the accuracy of a forecast. All of them are also continually being worked on, which will push current limits of accuracy further. Hence, on the basis of our experience so far, there is every reason to keep planning for increases in numerical resolution and complexity of parametrization schemes, should we have more powerful computers at our disposal. Other approaches contributing to the requirement for more computational power are the increasing use of Monte Carlo forecasting and the adoption of three and four dimensional variational analysis schemes.

[Bengtsson] gives some estimates of resource requirements for short and medium range forecasting and climate simulation in the future. For medium-range global forecasting, in the order of 2. 10^{14} floating point operations, i. e. 200 Tflops, are needed. Since we should like to compute a forecast in less than two hours, the sustained speed of the computer running the model must be about 30 Gflop/s. This is more than an order of magnitude more than can be realized at the moment. A T400 L60 global model to be used in medium-range forecasts in the mid-1990's would, under similar assumptions, require about 1 Gword of real memory and a sustained speed of 100 Gflop/s. As many recent supercomputer procurements for climatological simulation demonstrate, the above requirements are by no means an upper limit. To attack global warming and other great climatological challenges, a sustained speed of 1 Tflop/s would be required by the end of the decade.

Even though vector supercomputers have improved steadily, the speed of individual processors does not seem to improve much any more. A state-of-the-art Cray YMP/8 is about 25 times faster than a Cray-1 in 1977; of this only a factor of 2 comes from speeding up processors. The rest is due to parallellism and increased memory bandwidth. Simultaneously, the speed of massively parallel supercomputers has improved quite dramatically, and is presently exceeding the speed of vector supercomputers in some applications. It therefore seems likely that the supercomputers first to break into the 100 Gflop/s to Tflop/s terrain in sustained performance will be massively parallel.

Since 1984 ECMWF has followed developments in massively parallel computing by arranging biennial workshops on the Use of Parallel Processors in Meteorology. The fourth such workshop will take place in 26-30 November 1990 and focuses on software environments for parallel computers. Since 1989 we have also participated in the ESPRIT project Genesis, which aims at producing a European massively parallel supercomputer by the mid-1990's.

2. Use of parallel processing

So far, massively parallel computers have only been used experimentally by meteorologists. Part of the reason has been lack of reliable vendors and guaranteed upgrade paths on the market: a lack of continuity that is necessary for all production work. Another reason has been the high programming threshold to start writing code on such machines. The main reason, however, has been uncertainty about whether massively parallel computers can really deliver the superior performance expected from them on production codes: there haven't been any true massively parallel supercomputers on the market until very recently.

The algorithmic side of this last factor has been addressed by a number of meteorological centres, including National Center for Atmospheric Research (NCAR), United Kingdom Meteorological Office, National Meteorological Center in Washington, Royal Netherlands Meteorological Institute, ECMWF and many universities, by porting kernels and even complete models onto massively parallel machines during the last five years or so. The results have invariably been encouraging: good efficiencies have been reached. Also negative factors have emerged. Programming is still quite tedious and has to be done again for every new machine. Also, vector supercomputers have been able to keep ahead of massively parallel supercomputers in sustainable peak performance, contrary to common expectations. Hence, despite good efficiency, in absolute terms, we still have to see a full operational weather model running faster on a massively parallel supercomputer than on a vector supercomputer.

We shall look very briefly into two of the problems listed above: the programming aspect and the attainability of ever faster sustained speeds on weather models. The latter analysis is carried out with the asymptotic behaviour of large resolution models on large massively parallel computers in mind. Practical judgement on the suitability of a particular computer to a particular model will still have to be done case by case.

2.1 User interface

So far, the most prominent division in programming parallel computers has been that between shared memory and distributed memory programming models. Having to write code where references to different parts of a single data structure have to be made by explicit message passing, with the associated buffering and other complications, is very tedious.

Three principal ways to approach this problem have been suggested. First, there are a number of novel parallel languages that explicitly include parallelization constructs. Examples are Occam [Occam], Force [Jordan], Strand [Foster] and Booster [Paalvast]. Many of these are very elegant and often have sound parallel semantics, which helps crucially in writing parallelizing compilers for them. Some of the languages assume a shared memory, leaving it to the hardware vendor to do the physical mapping.

The main objection to novel languages is the pre-eminent status of Fortran in scientific computing. This seems to be one of the most stable patterns in scientific modelling as a whole, even to a degree of rejecting parallel extensions to Fortran, as demonstrated by the Fortran 90 effort. It would therefore be mandatory for a novel parallel language to provide the ability to link Fortran modules. Often novel languages would be used essentially as communication harnesses only.

The second avenue is provided by subroutine libraries and higher level software packages. The use of the latter is greatly on the increase and it seems quite possible that scientific computing will follow administrative computing in rejecting an old programming language - Fortran in scientific computing, COBOL in administrative computing - in favour of high level tools - scientific software in scientific computing, data base languages and system generators in administrative computing. High level tools are, however, most often written using Fortran and the task of parallelization is transferred to software vendors.

Some portability can be provided by communication libraries like COMLIB [Hempel], and the Argonne/GMD Communication Macros [Bomans]. Also, there are libraries to help in some aspects of parallelization such as automatic distribution of irregular grid structures and dynamic load balancing. Even these tasks, however, would be much easier with some form of shared memory. The third solution proposed to the programming problem is a real or virtual shared memory.

As SIMD computers demonstrate, a physically distributed memory does not necessarily imply distributed memory programming. The problem is, rather, how to manage the logical complexity of efficiently mapping a shared data structure to a distributed memory with limited long-range communication capacity. Reducing the combinatorial complexity of keeping track of every single memory reference, as well as avoiding thrashing due to excessive page traffic, are the key issues in designing a (partial) virtual shared memory.

Thinking Machines have approached this by parallelizing only Fortran 90 array constructs. This could be extended to incorporate references to a limited number of data structures that could be explicitly declared shared. If their number was small, the associated combinatorial complexity would be reduced greatly. Page traffic could be conducted at object, rather than page, level, allowing for more efficient high volume data transfer. Allowing the user to request access to a segment of a shared data structure in advance, using e. g. Fortran array syntax, and imposing the location of any such segment by hand, if necessary, would be important ways of boosting efficiency while retaining the convenience of some kind of shared memory.

I/O may also have to be different on a massively parallel system compared to a vector supercomputer. The host computer is a severe bottleneck in centralized file systems. The most economical way to provide a lot of I/O capacity on a massively parallel computer is using distributed SCSI disk systems attached to processing nodes. These create a problem analogous to a distributed memory. It would therefore seem natural that they would be used for memory mapped I/O alone: to store data structures rather than files. A conventional file system could be provided at the host for low bandwidth I/O. Output requested from a supercomputer will more and more be in a graphical format. This can be more efficiently handled by a separate video output channel. Therefore, it may be that the role of explicit I/O will diminish in the future, which makes programming simpler.

2.2 Impact of architecture

Apart from programming differences, the most significant difference between shared memory supercomputers and massively parallel computers is probably data access. Massively parallel computers are mainly advocated because of superior capacity and cost effectiveness in computation. Algorithms are mainly studied from the point of view of parallelizability of computation. In the case of meteorological models it seems that there is a lot of potential as far as parallelization of computation goes - however, there are serious potential problems in data access.

2.2.1. Global data access characteristics of meteorological algorithms

The spherical shape of the earth is the dominant geometric anomaly of global weather models, when compared with the conventional model problems usually used for demonstrating the efficiency of parallel computers. This causes grid lines to converge near the poles, see Figure 1. This singularity is not physical but results from mapping the sphere onto a rectangle. It is unavoidable in any regular latitude-longitude grid. For the sake of programming and algorithmic convenience, these are the preferred choice. If not accounted for, grid line convergence imposes a severe stability constraint on time stepping, making model runs very expensive.

In local discretizations like finite differences or finite elements, grid line convergence can be compensated for by making the equations implicit along the latitudes in a neighbourhood of the poles. This can be accomplished by Fourier filtering or line-implicit time stepping schemes. Skipping grid lines near the poles has the same effect.

In spectral methods with spherical harmonics and triangular truncation, the discrete equations are rendered isotropic everywhere. The spherical harmonic basis thus automatically filters the undesired rapidly oscillating modes near the poles that constrain the stability of the time stepping. Since physics and the nonlinear terms in dynamics have to be computed in any case on the associated physical grid, it is necessary to transform fields between spectral and grid representations. This is accomplished by a longitudinal Fast Fourier Transform and a latitudinal Legendre Transform.

Using the next operational resolution, T213 L31, of the ECMWF spectral model as an example, in grid point space we have 320 x 640 vertical columns, of height 31 each. It will be assumed that each computational cell in a column hosts 4 prognostic and up to 6 diagnostic variables, except the one in the bottom layer which has 5 prognostic and 8 diagnostic variables. The total number of variables in each vertical column is consequently estimated to be 313. The total volume of data at one time step is 64 Mwords.

In addition to data allocation requirements imposed by the geometry of the earth, each numerical algorithm has its own characteristic computational geometry. A rectangular two dimensional grid with nearest neighbour connectivity is the natural data dependency graph of explicit grid point methods without Fourier filtering. This assumes that vertical columns are stored within individual processing nodes. This is justified by the intense vertical data dependencies in the parametrization. Fourier filtering introduces data dependency along every full latitude in a neighbourhood of the poles, see Figure 2.

Spectral methods are global and the dimension of their data dependency graphs grows logarithmically with resolution. They display strong data dependency along both latitudinal and longitudinal directions, requiring transposition in between if all individual Fourier and Legendre transforms are to be performed within individual processing nodes. Due to the necessity of tensor-product grids in spectral methods, this dependency is orthotropical and does not extend beyond the latitude and the longitude each grid point is on, see Figure 3. The internal data dependency structure of the Fast Fourier Transform is a hypercube, which is a compressed butterfly, see Figure 4.

In other branches of fluid dynamics, multigrid methods have gained in popularity. It is possible that they will play a rôle also in weather modelling. They are essentially grid point methods, more often used in connection with finite differences than finite elements. They share the basic grid-like nearest neighbour data dependency structure of grid point methods. In addition to this, multigrid methods also display a pyramidal dependency structure due to the coarse grid correction procedure employed by them, see Figure 5. They work on a sequence of grids with geometrically varying resolution. The data transfers between these grids impose the pyramidal data dependencies. The spherical geometry introduces increasingly strong longitudinal data dependencies near the poles in multigrid methods, too.

It is important to realize that data dependency graphs are weighted. The weight along each edge is the volume of communication along it. The basic structures described above have an essentially uniform weight distribution. This means that the potential communication volume required at each processing node grows logarithmically with the number of nodes when implementing spectral methods, whereas it remains constant with explicit grid point and multigrid methods. The maximal length of wires needed to connect nodes remains constant with explicit grid point methods without Fourier filtering, whereas it grows proportionately to the number of processors with the spectral and multigrid methods.

As a whole, global weather models will be increasingly dominated computationally by physics and nonlinear dynamics, both of which involve almost purely local computations. Therefore, the effective parallelization of the linear part of the dynamics may not be crucial to performance in the end.

A particular numerical technique that will most certainly be employed in spectral and grid point models alike is semi-Lagrangian advection. Semi-Lagrangian advection is, in principle, similar to a grid point method in its data dependency pattern. The set of nearest neighbours extends up to five grid lines away, similar to a high order grid point method. This strains both communication links and local memory.

2.2.2. Mapping weighted dependency graphs onto parallel topologies

In an ideal situation a massively parallel computer should have so rich a topology that any edge in the dependency graph of any relevant algorithm can be mapped onto a single physical link - even when the communication volume is taken into account. In practice, this is seldom the case. A relevant measure of the success of this mapping is the average or maximal expansion any edge in the dependency graph is subjected to in the mapping, i. e. over how many consecutive links edges of the dependency graph have to be stretched.

Communication over any channel is estimated by an affine function of the message length:

$$T(l)=s+b*l \qquad (2.1)$$

where T(l) is the time it takes to communicate a message of length l. l is expressed in units of 64-bit words. s is the start-up overhead and b is the inverse bandwidth of the communication channel, i.e. the time to communicate a 64-bit word. The bandwidth is divided by the average expansion whenever the relevant communication channels are saturated.

The communication times (2.1) over edges in the dependency graph mapped onto a computer can be thought of as their lengths; communication start-up time is a lower bound on the length of any edge. The parallel communication delay is then the length of the longest path in the expanded dependency graph. Computation can be incorporated into this framework analogously, with vector start-ups replacing communication start-ups and speed of computation replacing bandwidth of communication.

A high start-up means that the edges have a long minimal length. In meteorological algorithms communication patterns are typically pre-determined and start-up can normally be accounted for if computation and communication can be overlapped. It does make a machine less robust, though, and can be a nuisance in algorithms with lots of serially synchronized stages with a small volume of computation and communication like Full Multigrid.

Obviously, dense dependency graphs are more demanding of parallel interconnection topology. An entirely global data access, a cross-bar, can be geometrically represented as a simplex (a triangle in two dimensions, a tetrahedron in three and so forth, see Figure 6). This is the structure of a shared memory computer with uniform access latency. A hypercube represents the structure of FFT and other global algorithms with a logarithmic parallel complexity, as seen from Figure 4, or a shared memory parallel computer with logarithmic memory access latency like BBN. Both have a valence that grows with the number of processors. The growth is linear in a crossbar and logarithmic in a hypercube. Both are capable of executing an arbitrary parallel program without data access constraints, the latter with a logarithmic delay. On a grid, on the other hand, a block of data accessed globally will have to travel across half the diameter of the machine, hence consuming a corresponding multiple of communication bandwidth.

Unfortunately, with current technologies (i. e. without optics) we are likely to see very little increase in valence. The relevant "dimensionless" quantity is, in fact, (b stands for bandwidth):

$$f_{1/2} = b_{comp} / b_{comm}, \tag{2.2}$$

which is more likely to increase in the future with faster individual processors, rather than decrease as required by increasing valence. Hence, data access will be a most serious hardware constraint on the use of massively parallel computers in meteorology.

[Hockney] has termed the above quantity $f_{1/2}$ computational intensity: a property of an algorithm and a computer alike. For an algorithm, high computational intensity means there is a lot of computation to hide communication delays. For a computer, low computational intensity means the computer does not require high computational intensity from the algorithm in order to attain good efficiency: communication can keep up with computation. The larger the computational intensity of an algorithm and the smaller that of the computer, the more effectively parallelizable the algorithm will be on the computer. The total parallel speedup can be approximated by:

$$s_{tot} = s_{comp} \left(1 - \left(1 + \frac{f^{alg}_{1/2}}{f^{mach}_{1/2}}\right)^{-1}\right) \tag{2.3}$$

where

s_{tot} = total speedup

s_{comp} = computational speedup

$f^{alg}_{1/2}$ = computational intensity of the algorithm

$f^{mach}_{1/2}$ = computational intensity of the computer

In the case of meteorological algorithms - and many others - the relevant quantity is rather

$$f^{exp}_{1/2} = b_{comp} \times exp_{av} / b_{comm}, \qquad (2.4)$$

where exp_{av} is the average expansion of edges in the dependency graph when mapped onto the parallel interconnection topology. This makes the data access problem even more pronounced on massively parallel computers with low valence.

The situation can be contrasted to that of a Cray YMP, where three memory references can be carried out every clock cycle. Hence, computational intensity is approximately 0.3 - a very good number. In the case of massively parallel computers we often have computational intensities in the order of a few hundred, unless the processors are extremely slow. This makes it of paramount importance to search for algoritms with high computational intensities.

Since grid point methods have sparse interconnection topologies, they would seem more readily suitable for massively parallel computers. However, even they suffer from Fourier filtering and in practice the suitability of any particular parallel computer to any particular algorithm will have to be studied individually.

When the topology is rich enough, spectral methods can be very competitive, as is demonstrated by the benchmarks reported in section 3.1. Semi-Lagrangian advection is likely to be used in connection with both spectral and grid point methods. Despite almost local access patterns, it is heavily dominated by the volume of communication, requiring a lot of weighted connectivity.

2.2.3. Local memory access

There is yet another constraint on the performance of massively parallel computers caused by data access requirements. The relevant dimensionless quantity is local computational intensity

$$f^{mem}_{1/2} = b_{comp} / b_{mem}, \qquad (2.5)$$

i. e. the rate at which the processor needs to access its local memory. Again, this is a common quantity for both algorithms and machines. In the case of meteorological algorithms, local computational intensity is fairly small, i. e. individual numbers are not used in many floating point operations.

Here, too, a high intensity algorithm and a low intensity processor are a good match. Unfortunately, massively parallel computers are normally crucially dependent on slow DRAM memories. Hence, no matter how fast individual processors might be, the benefit will be lost without a high degree of memory interleaving, requiring expensive memory reference logic, or the use of expensive SRAM memories. The latter is a less dense technology and reduces the maximal size of memories, too.

In a sense, we are here facing the current limits of performance achievable by massively parallel computers on meteorological algorithms. The speed of individual processors will be limited by memory bandwidth, and the number of processors by packaging and our desire to fit the computer into a normal computer hall. Clearly, careful optimization of system parameters is needed to attain this limit.

3. Initial results

The following benchmarks have been executed on an Intel iPSC/2 hypercube and a Suprenum parallel computer [Trottenberg] in collaboration with Saulo Barros, Gesellschaft für Mathematik und Datenverarbeitung in Bonn (GMD) and University of Sao Paulo. We compare two codes for solving the Helmholtz equation on the sphere. Several 2-D Helmholtz equations are solved every time step in all totally or partially implicit global, multilayer weather models. The combination of having to solve an elliptic equation and the spherical geometry have been identified as the most prominent algorithmic problems in implementing operational global weather models on massively parallel computers.

The first code uses a spherical full multigrid algorithm developed by Saulo Barros at GMD. To eliminate the geometric singularity at the poles due to grid line coalescence it uses a latitudinally implicit line relaxation in a neighbourhood of both poles. The Full Multigrid algorithm uses an F-cycle with a single relaxation sweep in both the injection and interpolation stages. At every cycle, the solution is gradually compressed into a small grid by alternating relaxation and projection onto a coarser grid. On the smallest grid, the equation is solved exactly, after which it is gradually expanded into the full grid by alternating interpolation and relaxation.

The second code is a spectral code using spherical harmonics as basis functions, written by the first author and Saulo Barros. It accomplishes the spherical transformation by a longitudinal Fast Fourier Transformation, using a vectorized FFT code by Clive Temperton of ECMWF, and a latitudinal Legendre Transformation calculated by Gaussian quadrature. The auxiliary routines to compute the Gaussian weights and the coefficients of the Associated Legendre Functions needed come from the package SPHEREPACK by Paul Swarztrauber and John Adams of NCAR. The code performs a symmetric-asymmetric decomposition and uses rhomboidal truncation.

Because of the strong longitudinal dependencies near the poles, both codes use a longitudinal stripwise data allocation to processors. In addition, the spectral code performs a transposition to latitudinal storage, in order to be able to execute Legendre transforms within individual nodes.

Both codes have single processor and parallel versions. Parallelization is accomplished using Argonne/GMD Macros and COMLIB calls to set up and communicate data structures.

3.1 Intel iPSC2 results

Both codes run with reasonably good parallel and vector efficiency on the Intel. The multigrid code has a better asymptotic complexity but due to its structure is less efficient to vectorize and parallelize. When solving a Helmholtz equation on a 128 x 256 grid, the speedup when going from one processor to 32 processors is 15 for scalar and 9 for vectorized code. The speedup due to vectorization is 2.3 on a single processor but only 1.4 on 32 processors. These figures improve significantly on larger problems.

The execution time of the spectral code is dominated by the Legendre Transforms. Their structure is simple and easy to vectorize. Parallelization overhead consists of the two transpositions. The parallel speedup when going from one to 32 processors is roughly 26 for scalar and 25 for vectorized code. Vector speedup is 3.3 for one processor and 3.1 for 32 processor implementations. The execution times on the single node case had to be extrapolated from smaller problems, since there is not enough memory to store the coefficients of the Associated Legendre Functions for a problem of this size on a single processor.

Despite inferior speedups, the multigrid code is faster than the spectral method in three cases out of the four compared here. However, the edge it has over the spectral code decreases dramatically when the code is parallelized and vectorized. In the scalar single processor case, multigrid is 3.2 times faster than the spectral method, which is to be expected on the basis of the suboptimal complexity of the Legendre Transform. When both codes are parallelized and vectorized, however, the spectral method is 1.2 times faster than multigrid.

The reason for not running comparisons with larger problems is that in reality we would be solving a large number of independent Helmholtz equations of roughly the size used in these benchmarks, one for each vertical layer. Besides, the spectral solver has to store the coefficients of the Associated Legendre Functions, which takes up all available memory already at this resolution. Both codes would benefit from longer vectors in the three dimensional case.

3.2 Suprenum results

On the Suprenum the work has barely begun. The only results so far are for the scalar version of the spectral code. On a 32 x 64 grid the parallel speedup when going from one processor to four is 3.1. The effect of vectorization is expected to be even more dramatic on the Suprenum than on the Intel, due to a larger speed difference between the vector unit and the scalar processor.

A remarkable effect in global meteorological models is that we observe a superlinear growth in speedup, measured in Mflop/s, when model resolution is scaled up with the number of processors to match the available memory. This is due to increasing vector lengths, facilitated by the necessity of storing every latitude and longitude in a single processor at the appropriate stage of the solution. When the number of processors is increased and the problem simultaneously scaled up, we store fewer but longer vectors in every processor.

4. Performance predictions

Along with benchmarking kernels, an effort has been made to estimate the performance of the full operational model on some massively parallel computers. A fairly detailed estimate was computed for the Suprenum on the basis of its hardware characteristics. These characteristics ignore all system software overheads. The aim is to evaluate the potential of Suprenum in running global operational weather models in the case in which both the code and the run-time environment are optimally tuned.

The figures calculated are no substitutes for actual performance measurements, but since the task of porting the present ECMWF operational model or any of its algorithmic alternatives to an architecturally very different computer is formidable, they try to provide a somewhat more realistic basis for performance comparison with existing vector computers than mere peak performance claims or dusty deck performance measurements on each. The former tend to overestimate the power of massively parallel computers, whereas the latter give somewhat undue credit to vector computers because of their wider applicability and better performance robustness on general Fortran programs.

The estimates are based on a still somewhat superficial analysis of the algorithms and the Suprenum and are, therefore, liable to change in the course of a more detailed investigation. We also hope to be able to incorporate estimates for some alternative formulations of the model equations later, in order to be able to assess the effect of machine architecture on the choice of algorithm, assuming that they all produce meteorologically equivalent forecasts.

Suprenum has 256 processors, arranged in 16 clusters of 16 processors each. The processors within each cluster are connected by a fast cluster bus, whereas the clusters are connected by 16 slower Suprenum-buses, being logically arranged in a four by four square. Each node has about 1 Megaword (64-bit) of local memory. The total memory is 256 Mwords.

In general, the estimates are based on the following simplifying assumptions:

1) Communication over any channel is estimated by the affine function of the message length (2.1).

2) Computation is always performed at a nearly optimal rate of

8 Mflops per processor. The peak performance of the Weitek processors used on the Suprenum is 20 Mflops, but in the case when every triad-type operation requires two operands from the main memory for each result, the peak is only 10 Mflops. The peak performance of Suprenum is consequently 5 Gflop/s or 2.5 Gflop/s, respectively. 8 Mflops per node corresponds to a typical maximum speed measured, so far, in linear algebraic single processor benchmarks of the appropriate type. To attain this speed, the code must be vectorizable and the operands must be arranged to have small stride in the memory, since, otherwise, page traffic on a virtual memory machine will adversely affect the performance. In matrix multiplication, i. e. Legendre Transform, however, the speed is more likely to be 12 to 13 Mflop/s per processor.

In most parts of the algorithm, the critical resource is either the bandwidth of the global Suprenum-buses, or the start-up and bandwidth of communicating any individual message between two processors. The delays in the latter are caused by the system overhead to set up a logical communication link, and the delays internal to processors due to memory access during communication. The cluster buses are about twenty times faster than the inherent delay in any single message. Hence, they are never a bottleneck, as they can support simultaneous communication of all the processors in each cluster. The global Suprenum-bus is accessible from individual nodes through a special communication node.

The total time to integrate the model for 15 minutes was estimated at 25.2 s. The computational intensity thus achieved would be 70 % and the sustained performance would attain 1.4 Gflop/s. This is roughly comparable to the performance expected to be seen on the 8-processor Cray YMP, where a slightly different version of the model has already been tested with a sustained performance of about 1.2 Gflop/s.

5. Outlook

The market of massively parallel computers is still very unsettled. Companies come and go almost on a monthly basis and only a few have had sustained presence. Virtually all sales of massively parallel computers so far have been for experimental purposes. Typically, these have been heavily subsidized from public funds. There are signs that this pattern is beginning to change.

Due to the difficulty of programming and lack of portability, as well as unrobustness of performance, due mainly to lack of global communication bandwidth, it seems that massively parallel computers may never attain the same generality as present vector supercomputers. Instead, they will be used as special purpose scientific and engineering engines. Even though not able to execute dusty deck Fortran codes efficiently, they seem to be general enough for solving most partial differential equations quite fast. These continue to form the hard core of scientific and engineering computing, in particular numerical weather prediction.

Massively parallel computers will have to compensate for the above mentioned deficiencies by providing clearly superior peak performance and performance/price ratio. An approximate rule of thumb might be that this factor must be around ten before the majority of scientific programmers start to consider massively parallel computers seriously for their production work. Good programming tools would, of course, speed up this process.

It may be that in the future current and massively parallel supercomputers will increasingly coexist. A supercomputer site like ECMWF generally needs some robust supercomputer capacity in addition to a fast model engine. Getting data out of a massively parallel computer may also be a non-trivial task. Having a fast front-end will be very helpful.

Networking will give more and more researchers access to a wide range of computers. This leads quite naturally to increasing specialization among computing facilities. Hence, provided that massively parallel computers can realize the expectations of Teraflop range performance placed on them, there will be a niche for them to succeed even commercially. Specialized supercomputer sites like operational weather forecasting centres may well be among the first to benefit from this.

6. Literature

[Bengtsson] L. Bengtsson: Computer Requirements for Atmospheric Modelling. In: Multiprocessing in Meteorological Models. G.-R. Hoffmann, D. F. Snelling (eds.). Springer, Berlin Heidelberg New York 1988. pp. 108-116.

[Bomans] L. Bomans, R. Hempel: The Argonne/GMD Macros in FORTRAN for portable parallel programming and their implementation on the Intel iPSC/2. Arbeitspapiere der GMD 406, Sankt Augustin 1989. Submitted to Parallel Computing.

[Convention] Convention establishing the European Centre for Medium-Range Weather Forecasts. Her Majesty's Stationery Office Cmnd. 5632. London June 1974.

[Dent] D. Dent: The ECMWF Model: Past, Present and Future. In: Multiprocessing in Meteorological Models. G.-R. Hoffmann, D. F. Snelling (eds.). Springer, Berlin Heidelberg New York 1988. pp. 369-381.

[Foster] I. Foster, S. Taylor: STRAND: New concepts in parallel programming. Strand Software Technologies, Watford 1989.

[Hempel] R. Hempel: The SUPRENUM Communications Subroutine Library for Grid-oriented problems. ANL-87-23. Argonne National Laboratory, Argonne 1987.

[Hockney] R. Hockney, I. Curington: $f_{1/2}$: A parameter to characterize memory and communication bottlenecks. Parallel Computing 10 (1989) pp. 277-286.

[Jordan] H. Jordan, M. Benten, G. Alaghband, R. Jakob: The Force: A highly portable parallel programming language. CSDG 89-2, Department of Electrical and Computer Engineering, University of Colorado in Boulder 1989.

[Occam] Inmos Limited: Occam2 reference manual. Prentice-Hall, New York 1988.

[Paalvast] E. Paalvast, A. van Gemund, H. Sips: A method for parallel program generation with an application to the Booster language. To appear in: 1990 ACM International Conference on Supercomputing, June 11-15, 1990, Amsterdam.

[Trottenberg] U. Trottenberg: Suprenum - the concept. Supercomputer 30, 1989, pp. 5-19.

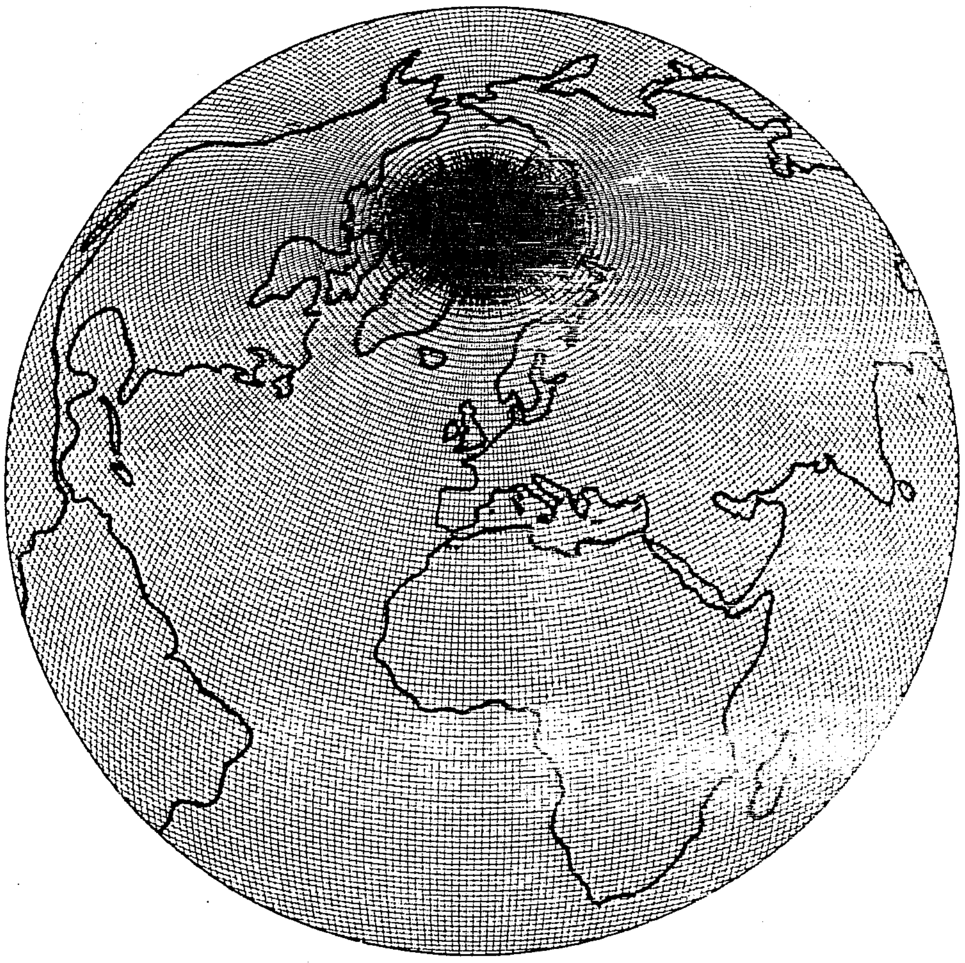

Fig. 1 Coalescing grid lines on a sphere. Mutual data dependency is inversely proportional to the distance between two grid points.

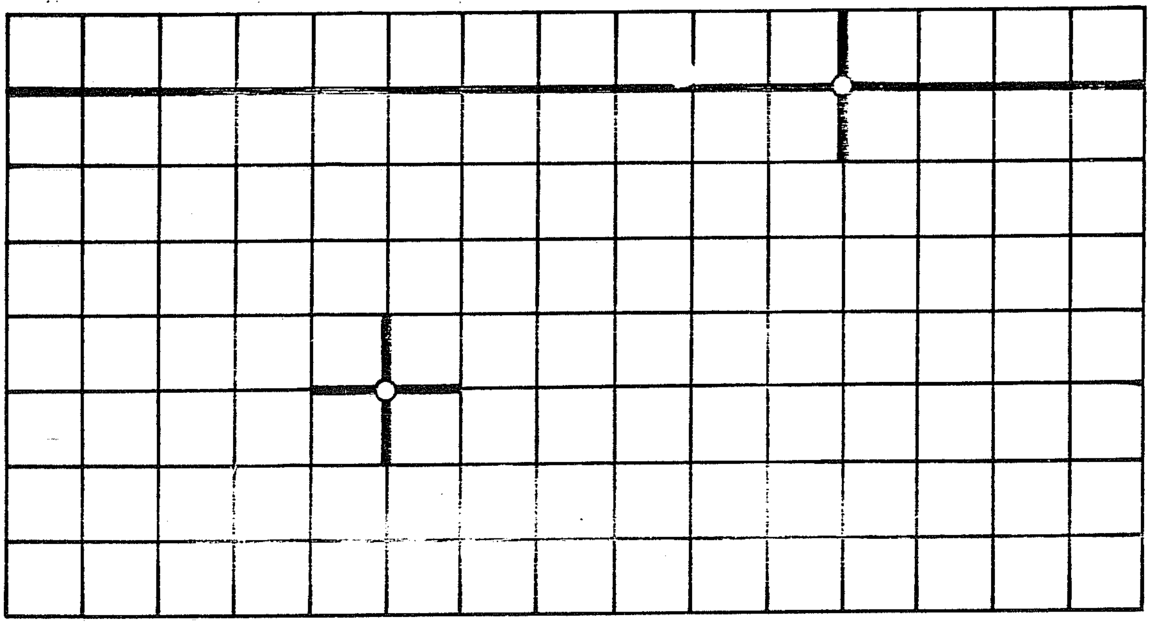

Fig. 2 Grid topology of an explicit scheme. Thick lines show data dependency patterns when the grid is mapped onto a sphere and Fourier filtering is used.

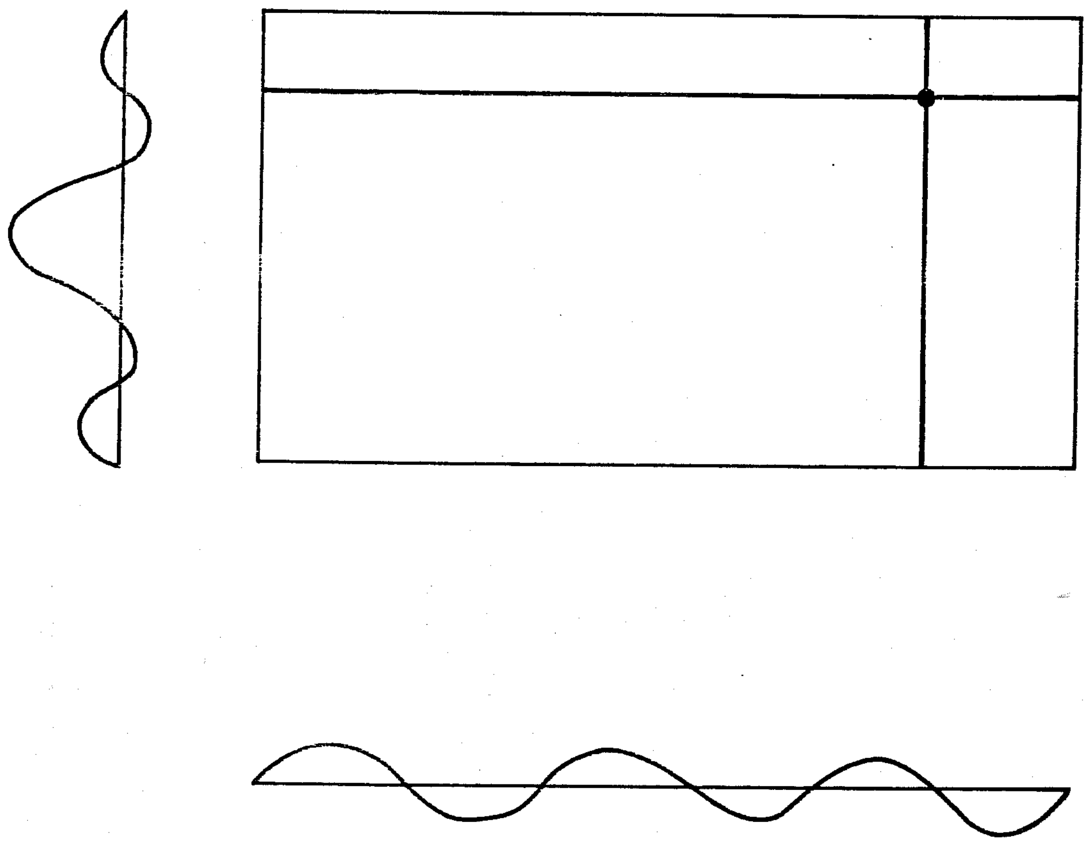

Fig. 3 A global two dimensional tensor product basis is a tensor product of two one dimensional global bases: every point is dependent on every other point on the longitude and the latitude it is on.

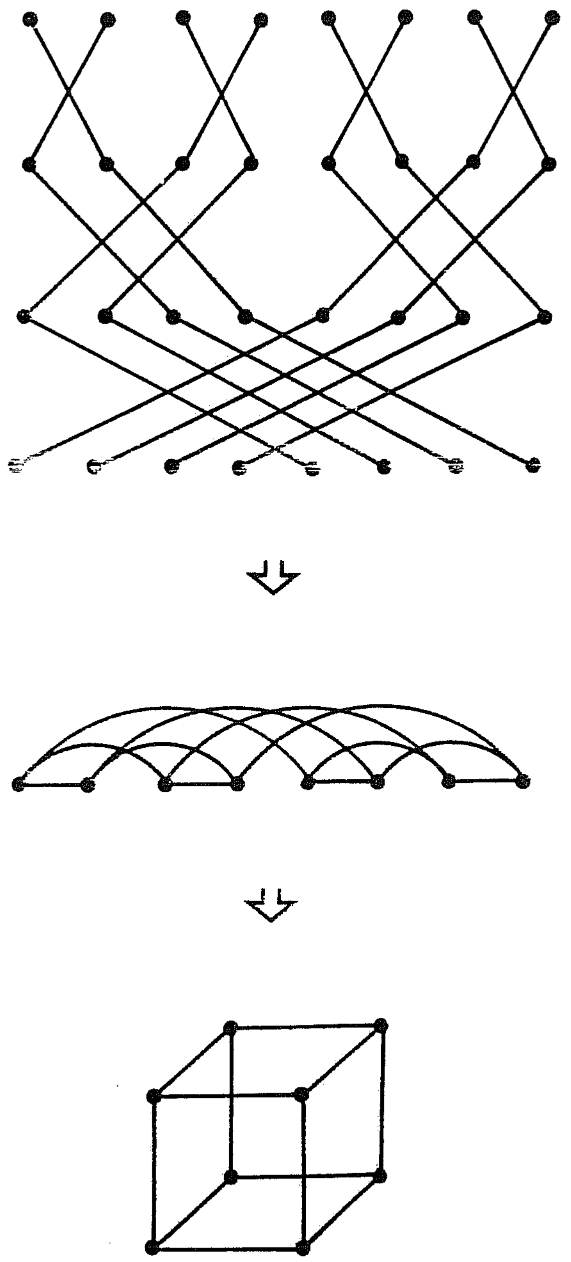

Fig. 4 Fast Fourier Transform as a hypercube. The butterfly depicts data flow in the FFT. If we compress the butterfly and insert an edge between all nodes that need to communicate, the emerging pattern is a one dimensional rendering of the edges in a hypercube.

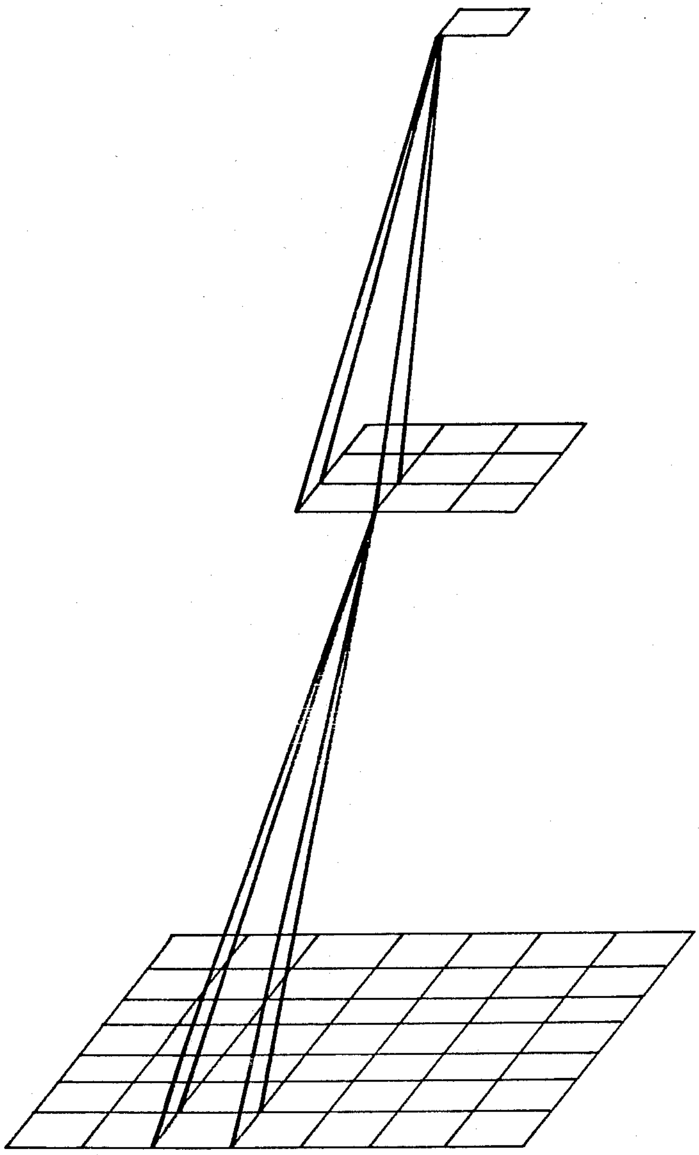

Fig. 5 Pyramidal topology of a multigrid algorithm. Only some of the vertical edges are shown.

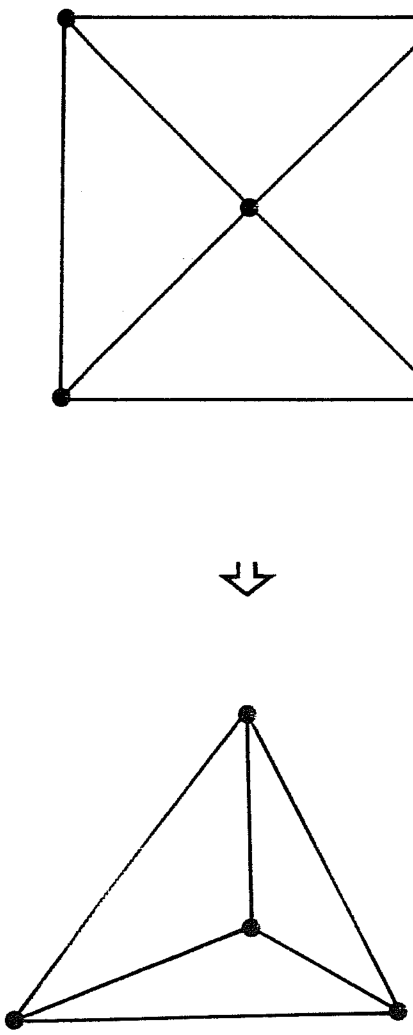

Fig. 6 A cross-bar connection between four processors is a two dimensional rendering of a simplex. In the case of four processors this is a tetrahedron.