CANADIAN METEOROLOGICAL SERVICES' PLANS FOR USING MULTI-PROCESSOR SYSTEMS

Andrew Staniforth
Atmospheric Environment Service, Environment Canada
Dorval, Québec, Canada

1. INTRODUCTION

The Atmospheric Environment Service (AES) of Environment Canada has a mandate to provide comprehensive information to Canadians concerning past, present, future and possible conditions of the atmosphere. To fulfil this mandate AES has developed, and is continuing to develop, numerical models of the atmosphere for both research and real-time operational applications in the broad areas of

(i) weather prediction,

(ii) climate, and

(iii) air quality.

A powerful computer is evidently required for this kind of numerical modelling, and late in 1983 a Cray 1-S computer was installed at AES's facility in Dorval, Montreal, to replace a Control Data Cyber 176. An upgrade to a Cray XMP-22 (two-processors, two million words of memory) is scheduled for the Fall of 1986, and AES is currently examining how best to convert model codes from the 1-S and optimize them for the XMP.

Weather prediction codes that need to be run on the XMP include those for objective analysis and regional and global forecasting. The objective analysis code uses optimal interpolation techniques, the global codes use the spectral technique, the regional codes use finite-element and semi-Lagrangian techniques, and the global and regional forecast models have comprehensive parameterizations of sub-grid-scale physical processes.

The general circulation model code used for climate simulations is very similar to the global spectral weather prediction code. Model codes used for air quality studies and the long-range transport of pollutants include transport modules based on Eulerian and semi-Lagrangian techniques, as well as modules for the chemical interaction of many chemical species.

AES has an agreement with the National Sciences and Engineering Research Council of Canada to make 10% of the Cray computer's capacity available to researchers in many disciplines in Canadian universities. As a consequence many other codes are also run on AES's Cray in addition to those previously mentioned. They include, for example, diverse codes in oceanography, astrophysics, nuclear physics and engineering.

In this paper we mostly restrict our attention to coding considerations for global and regional forecast models, since these form the bulk of the computational load on AES's vector computer. Lacking any concrete experience of our own with a Cray XMP, an analysis (summarized in the following sections) was performed about a year ago with a view to anticipating (and avoiding) some of the difficulties likely to be encountered due to the architectural differences between the Cray XMP and the Cray 1-S. Now that the Cray XMP has been available to users at various sites for a year or so, we are very interested to hear their experiences (both good and bad), and the present workshop provides an ideal opportunity to do so. In this way we hope to be able to avoid some of the problems encountered by the first users. As can be seen from the content of the paper, an adequate amount of memory (both main and SSD) has been identified as a major concern.

2. MEMORY CHARACTERISTICS OF WEATHER PREDICTION MODELS

For a given level of accuracy spectral models require fewer degrees of freedom (i.e. less memory) than competing methods, but at the expense of more CPU time/degree of freedom, and their use of orthogonal basis functions permits a natural "slicing" of model computations such that I/O overhead is minimized. On the other hand, regional models trade-off accuracy at later time-frames for increased accuracy in the short time-frame (up to 2 days) by a redistribution of the degrees of freedom and computing effort. Such models use techniques that inherently require more memory than global or hemispheric spectral models, but this is generally compensated for by the fact they use less CPU time/degree of fredom.

In order to improve the accuracy of weather element forecasts at all time ranges, more sophisticated parameterizations of the unresolved physical processes are being incorporated into atmospheric models. To make the parameterizations more complete additional prognostic and diagnostic fields must be introduced. For example, a parameterization of turbulent fluxes presently being implemented requires two further 3-D prognostic variables, viz the turbulent kinetic energy and the turbulent mixing length, while a more accurate treatment of clouds requires prognostic cloud variables. All these extra variables improve the accuracy of the models, but they make it increasingly difficult to overlap I/O operations with CPU calculations for a given memory configuration. One of the current research thrusts in AES is to increase the efficiency of the timestepping algorithm, with the expectation of being able to reinvest the computational time saved in increased resolution. Thus it appears that pressure on the amount of available memory will further increase.

3. SOME MEMORY CONSIDERATIONS WHEN CONVERTING CODES FROM A CRAY 1-S TO A CRAY XMP

At any given instant in time a two-processor Cray XMP will hopefully be working on twice the number of operands as a single processor Cray 1-S. Furthermore, because they have multiple paths to memory (compared to the single path on the Cray 1-S) they complete their calculations in fewer clock cycles and are thus ready sooner to operate on the next set of operands. If these operands are unavailable then the CPU will be idle. This problem would never arise if the main memory were sufficiently large to store all the fields required for subsequent calculations. However in the real world, main memories are both limited in size and costly and we have to use a backing store, which on a well-configured Cray XMP is a SSD.

Some coding considerations that arise because of the architectural differences between the Cray 1-S and the Cray XMP are (and these are particularly important for the coding of regional models):

(i) the high overhead incurred to obtain the first word in a main memory/SSD transfer makes it advantageous to transfer data in large chunks (of the order of a hundred thousand words or more); the relative overhead is much higher than that of main memory/IOP transfers on a Cray 1-S because the first word still takes as many clock cycles to arrive on a Cray-XMP as on a Cray 1-S, even though subsequent words arrive 10 times as fast because of the increased channel speed;

(ii) transferring in large chunks means we must have large chunks of main memory available to accept them while the CPU is busy operating on other large chunks;

(iii) because there is a 6-fold increase in memory bandwidth (from memory to CPU) for a 2-processor Cray XMP compared to a Cray 1-S, the CPU's are ready for new operands much sooner and it is therefore more difficult to ensure all operands are available (after transfer from the SSD) when required by the CPU:

(iv) performing calculations in large chunks increases vector lengths which further enhances CPU performance and increases the chances that the CPU will be waiting for main memory/SSD transfers to complete.

We conclude from the above that for a given code having I/O overlapped with CPU execution (as is generally the case with weather prediction codes) that substantially larger amounts of main memory will need to be made available to a given code, otherwise the code is likely to become I/O bound.

4. MULTI-PROGRAMMING CONSIDERATIONS

If a computer system permits multi-programming, there will in general be an increase in the total amount of useful work performed during a given time period when compared to the same machine executing the same jobs sequentially; it also permits improved turnaround for short jobs. This is achieved by an increased parallelism and a more sustained usage of the various computer components (CPU, channels, memory etc), and is of particular importance in an environment that must process a large number of jobs having significantly different memory and CPU requirements, which is the case in AES.

Most centers that use a super-computer for atmospheric modeling fall into one of two classes. The first is characterised by being a real-time

operational environment serving a relatively limited number of users running a limited number of specialized programs (e.g. NMC, ECMWF), while the second is characterized by being research oriented with few, if any, real-time constraints (e.g. NCAR, GFDL). In either case it is acceptable (and current practice) to permit individual jobs to operate fairly close to the limits of memory. For the first class there are a limited number of users who tune their models to simultaneously use as many of the resources of the machine as possible and only a limited multi-programming capability is required. For the second class it is of paramount importance to get the result at the resolution required and, because it is not a real-time environment, such centers also have the luxury of permitting jobs to operate close to the limits of memory and accepting limited multi-programming. However, AES falls into neither of these two categories.

AES's Cray must support a vigorous research environment and a heavy operational one. Unlike ECMWF, the Canadian operational runs are spread fairly uniformly over 24 hours, including a heavy use of prime time, and account for approximately half of the total computer time; all other (research) jobs must be run either in parallel with the operational jobs or in the remaining time which is extremely fragmented. Furthermore the Cray computer configuration has to support university researchers as well as an ever-expanding number of AES users. There is clearly a need for multi-programming to increase machine throughput, in an environment that is both operationally and research oriented.

But multi-programming requires lots of memory, both main memory and backing-store memory (SSD). If we wish to simultaneously execute jobs in parallel, additional main memory is required otherwise we are unable to

simultaneously (for example) perform I/O for one job while the CPU is busy with another.

Let us examine the situation where an operational model is in execution during prime time (consuming large amounts of both main and SSD memory) and we wish to run jobs which have large memory requirements but small execution times (of the order of seconds compared to an hour or so for an operational model). If there is a sufficiently large amount of SSD memory available, then a copy of the contents of main memory associated with the operational model can be "rolled out" to SSD memory and another similar-size (but short execution) job "rolled in"; it is imperative that these transfers be effected extremely fast to minimize overhead (CPU idle time) and this requires the use of SSD memory rather than buffer-memory. By proceeding in this fashion the R&D community can still continue to use the machine in prime time for "large-memory-but-short-execution" jobs while operational jobs are being processed (which is an almost continuous phenomenon during prime time, given that the operational run consumes approximately one half of the computer resources spread fairly uniformly throughout the day). Several-times-a-day turnaround during prime time for such "large-memory-but-short-execution" jobs is vital for a viable atmospheric R&D program. Each such job requires an image of main memory contents to be stored on the SSD as well as the usual amount of SSD memory associated with the job. Since several of these jobs need to be SSD resident it is clear that an adequate amount of SSD memory is necessary, otherwise the potential throughput of a Cray XMP in our environment wil not be realized.

5. MULTI-TASKING CONSIDERATIONS

A multi-processor machine can potentially improve the total throughput of the machine for many users, or reduce the real-time execution of a job for a single user, or do both but to a lesser degree, and the optimum mix will depend on the goals of an organization. For the reasons mentioned in the previous section we anticipate that we will find ourselves comfortably (or perhaps uncomfortably) between the two extremes. We need adequate throughput to satisfy high demand, but at the same time operational results must not be unduly delayed, and these considerations have an impact on program design.

The fundamental question that needs to be addressed in designing programs for multi-processor execution is at what hierarchy level should the user(s) multi-task, that is partition the work-load into a set of more-or-less independent tasks for independent execution. Should it be at the level of

(i) the job (different jobs executing on different processors),

(ii) the job-step (different programs of a single job executing on different processors),

(iii) the program (different subroutines of a single program executing on different processors),

(iv) the subroutine (different loops of a subroutine executing on different processors), or

(v) the loop (different parts of a loop executing on different processors)?

At one extreme (i.e. level (i) of the above hierarchy) the user relies on the operating system to optimize throughput and helps the system by minimizing resource requirements, such as memory: this extreme would perhaps be appropriate in an environment where turnaround is not an issue.

At the other extreme (i.e. level (v) of the above hierarchy) a single user tries to squeeze the maximum real-time juice out of the proverbial lemon.

The following factors are also important when adopting a multi-tasking strategy:

(i) the (scratch) memory overhead associated with splitting a job into a large number of small tasks rather than a proportionally smaller number of large tasks;

(ii) the flexibility of the strategy (e.g. how easily can the program be adapted to other multi-processor machines having a larger number of processors);

(iii) the balancing of the computational load across processors;

(iv) the synchronization of tasks that depend on the completion of other tasks;

(v) the possible use of an alternative algorithm more suited to a multi-processor environment; and

(vi) the programming effort required to achieve the desired result.

In our environment we tentatively conclude from the above considerations that it is appropriate to multi-task model codes at the highest possible levels (consistent with acceptable real time performance) using a relatively small number of tasks, and to rely on the system to optimize throughput across user jobs. The advantages of such a strategy appear to be:

(i) a small number of tasks will need less scratch memory (which is in short supply);

(ii) the strategy is reasonably flexible inasmuch as it should be relatively straightforward to adapt the code to multi-processor

machines having a larger number of processors, by either further dividing the computational load between processors at the same level of multi-tasking hierarchy (when possible), or by descending one level in the hierarchy for critical portions of the code (when necessary);

(iii) the balancing of the computational load between processors for a given job need only be approximate (enough to give acceptable real-time performance) without needing to be optimal (since it is highly unlikely to be the only job in the machine and other jobs will interfere with it to some extent anyway); and

(iv) the fewer the number of tasks, the easier it is to program and synchronize them and the smaller the multi-tasking overhead.

The only real disavantages appear to be a degradation of real-time performance if run in isolation, and a reliance on system software to optimize machine throughput. This latter point brings us full circle back to the need to have adequate main and SSD memory. We can only hope the circle isn't vicious...

As regards putting the above theory into practice in the context of our weather prediction codes, a possible first attempt being considered for the dynamical calculations is to define the calculation of the right-hand sides of the momentum, thermodynamic, continuity equations, etc. as tasks and send them to different processors. On the other hand, the majority of the physical parameterization calculations are horizontally independent and can be split up into sets of vertical columns, each set of which defines an independent task. It seem prudent to try to restrict the number of independent tasks between synchronization points to be less than the total number of processors, and thus avoid initiating too many tasks that generate scratch memory demands.

6. RESULTS

With respect to our actual experience with a "multi-processor", it is very limited. Two of our models (regional finite-element and spectral) have been run by Robert Wellck of Cray Reasearch on a single-processor Cray XMP and execution times compared to that given by identical codes executing on a Cray 1-S; the results are summarized in Table 5.1. Although these results do not tell us what speed-ups we can expect to achieve with optimized codes on a multi (two) processor Cray XMP with respect to a Cray 1-S, they do at least provide lower bounds.

	Finite element		Spectral	
	CPU	wall-clock	CPU	wall-clock
Cray 1-S	1002	1745	172	427
Cray XMP (one-processor)	558	736	94	149
Speed up, Cray 1-S/Cray XMP	1.8	2.4	1.8	2.9

Table 5.1 Execution times (in seconds)

7. SUMMARY

AES objectives are such as to require the modeling of a wide variety of meteorological scales. It is argued that regional ("small-scale") models achieve more accurate local forecasts (but valid for a more limited time period) than spectral ("large-scale") models, by using methods which inherently require more memory but use less CPU time/degree of freedom. The need to simultaneously support real-time operational execution of models as well as government and university R&D is also such as to require more memory (both main and SSD) than might otherwise be needed, because of multi-programming considerations. Furthermore there are

also pressures on memory use due to multi-processor/multi-tasking considerations, all of which leads us to identify the amount of available memory as being a matter of concern, particularly for regional and mesoscale models.

In the context of our environment, it is further argued that it is appropriate to multi-task model codes at the highest possible levels (consistent with acceptable real time performance) using a relatively small number of tasks, and to let the operating system optimize machine throughput across user jobs. A tentative strategy is also given in the context of applying these principles to our weather prediction codes. In the longer term it is important to develop new algorithms (or adapt unused older ones) to take best advantage of the multi-processor architectures.

ACKNOWLEDGEMENTS

Discussions with Michel Valin and the expert typing of Maryse Ferland are gratefully acknowledged.