SAME-SOURCE PARALLEL IMPLEMENTATION OF THE PSU/NCAR MM5[1]

John Michalakes
Mathematics and Computer Science Division
Argonne National Laboratory
Chicago, Illinois 60439
michalak@mcs.anl.gov

ABSTRACT

We describe an IBM-funded project to develop a same-source parallel implementation of the PSU/NCAR MM5 using FLIC, the Fortran Loop and Index Converter. The resulting source is nearly line-for-line identical with the original source code. The result is an efficient distributed memory parallel option to MM5 that can be seamlessly integrated into the official version.

1. INTRODUCTION

The Pennsylvania State/National Center for Atmospheric Research Mesoscale Model is a limited-area model of atmospheric systems, now in its fifth generation, MM5 (Grell et al, 1994). Designed and maintained for vector and shared-memory parallel architectures, the official version of MM5 does not run on message-passing distributed memory (DM) parallel computers. Our previous work on the Massively Parallel Mesoscale Model (MPMM) and the follow-on Fortran90 implementation, MM90, demonstrated efficiency and scalability on distributed memory parallel computers and provided, as well, a more modular, dynamically configurable code (Foster and Michalakes, 1993; Michalakes, 1997b). Nevertheless, these earlier efforts fell short in that extensive modification for parallelization prevented integration with the official version.

With funding provided by the IBM Corporation, we have produced a new version called the same-source parallel implementation of MM5. It is line-for-line identical with nearly all of the official MM5, it is efficient and scalable, and the modifications have no effect whatsoever on function, operation, or performance of the model on other, non-DM parallel computers.

The same-source parallel MM5, designated MM5e, is running operationally on an IBM SP at the United States Air Force Weather Agency facility, Offutt AFB, Nebraska. The model is also in "friendly user" distribution to approximately twenty research, university, and commercial (vendor) users in the United States, Europe, and Asia who are providing valuable feedback. We are working with MM5 developers at NCAR to integrate the DM parallel option into the official MM5.

---

[1] This work was supported in part by the Mathematical, Informational, and Computational Sciences Division subprogram of the office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

The same-source approach to parallelization employs a lightweight yet powerful source translator, called FLIC (Fortran Loop and Index Converter [Michalakes, 1997a]), to generate the parallel code from the source model when the user types "make." The approach is essentially directiveless, requiring only a small amount of information—sufficiently general and concise to fit on the tool's command line—to direct the translation. The final version of the translator, under development with Applied Parallel Research Inc. of Sacramento, California, will be robust, commercial-grade software but will remain in the public domain for distribution with MM5.

Additional information on the same-source parallel implementation of MM5 is available at http://www.mcs.anl.gov/Projects/mpmm.

A number of individuals at Argonne have contributed to the development of the parallel MM5: Tom Canfield, Ken Dritz, Steve Hammond, Jace Mogill, Ravi Nanjundiah, and especially Ian Foster. NCAR collaborators included Georg Grell (now with the Fraunhofer-Institute for Atmospheric Environmental Research), Jim Dudhia, Dave Gill, Dan Hansen, and Bill Kuo. John Levesque and Rony Sawdayi of Applied Parallel Research are participating in development and improvement of the source translation software. The project is indebted to Jim Tuccillo, Ed Jedlicka, and IBM Corporation for supporting the work.

2. SINGLE-SOURCE AND SAME-SOURCE

Architecture-specific coding affects understandability, maintainability, extensibility, reusability, and portability to other, dissimilar architectures. Such coding may manifest itself in how arrays are dimensioned, aligned, and allocated in memory; how loops are nested or otherwise structured (blocked, unrolled, fused); at what level loops are positioned in the subroutine call hierarchy; how iteration is expressed (loops or array syntax); how information is exchanged between subroutines; and, with distributed memory, how communication is implemented. Maintaining separate codes is difficult and time consuming; and because changes and enhancements must be made by hand and tested over all versions, some inevitably fall behind. The ability to exploit a range of computer architectures with a single source code provides obvious software cost benefits. Approaches include conditional compilation, data-parallel languages, distributed shared-memory machines, and application-specific parallelization libraries.

Conditional compilation, the use of preprocessors such as the Unix C-preprocessor (CPP) to enable or disable architecture-specific sections, produces messy and difficult-to-read source code and merely obscures the fact that multiple codes are being maintained (just in the same set of files).

Parallel languages such as High Performance Fortran (Koelbel et al, 1994) aim to provide a means for writing efficient, single-source expressions of model software in which architecture-specific details are handled by the HPF compiler. Unfortunately, existing models may require substantial rewriting (which may be addressed using source translators (Hammond et al, 1995)). HPF affords limited decomposition options, say for load balancing, and may be ill suited for expressing less structured parts of a model computation, say for nesting. Portable performance has also been an area of concern.

Distributed shared-memory (DSM) machines maintain a single address space view of memory, thus allowing shared-memory parallel programming on distributed-memory hardware. DSM preserves investment in codes developed for shared-memory vector multi-processors (PVPs). The new OpenMP standard (OpenMP, 1997) provides a uniform set of directives for expressing shared memory parallelism. However, the appeal of the shared-memory programming model is based, in part, on the questionable notion that shared-

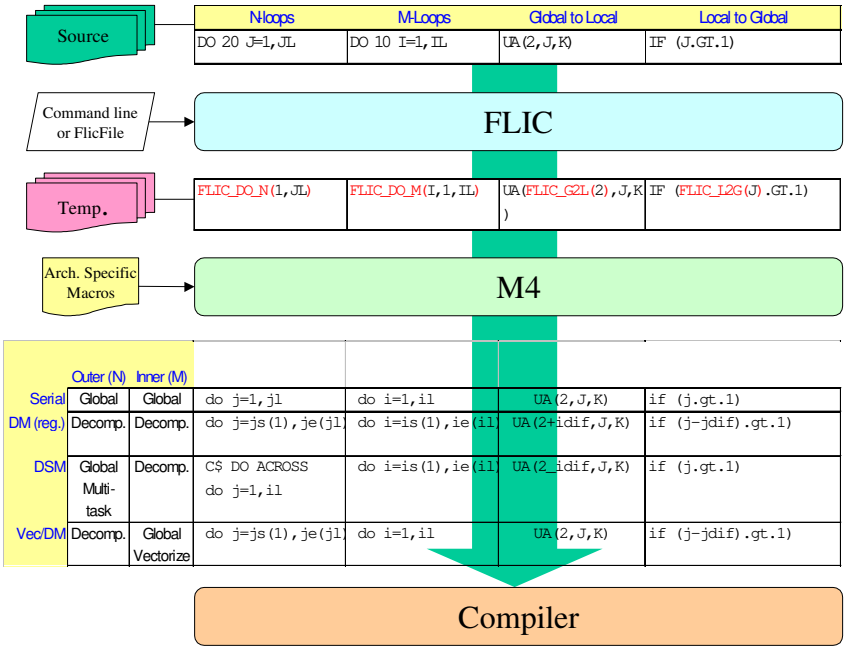

Figure 1 Same-source preprocessing

memory programming is easier than distributed memory for SPMD applications. In fact, for codes that have not been parallelized, either approach presents complexities and pitfalls. Existing models that have been parallelized for PVPs are usually parallel in only one dimension (leaving the other for vectorization), thus limiting parallelism and scalability. Finally, shared-memory codes are not portable to distributed-memory machines, whereas distributed-memory codes port trivially to shared memory. Lest one conclude portability to distributed memory is unimportant, a recent development in affordable supercomputing has been extremely low-cost networked configurations of personal computers (Cipra, 1997), a computational option unavailable to shared-memory programs.

Programming for distributed memory provides both portability and scalability. At one time, programming for distributed memory was a more difficult proposition, partly because it was unfamiliar. Now, however, most of the painful low-level detail originally associated with message-passing programming—domain-decomposition, message passing, distributed I/O, and load balancing—has been efficiently encapsulated in application-specific libraries (Hempel and Ritzdorf, 1991; Kohn and Baden, 1996; Michalakes, 1997c; Parashar and Browne, 1995; Rodriguez et al, 1995). However, these approaches still require modification to the code for handling loops over local data, global and local index translation, and distributed I/O. If one is able to design a new model or undertake a major redesign, these issues may be addressed directly in the code, as a number of groups have demonstrated (e.g., ECMWF's IFS and Environnement Canada's MC2 models). However, if a *same*-source and not only single-source implementation is required, the extent of allowable changes to the source is severely limited.

Source translation removes the remaining difficulties associated with implementing the model efficiently for distributed memory with minimum impact on an existing model source code (see Section 4). Further, source translation is applicable to a broader range of *performance portability*[2] concerns. Loop restructuring, data-in-memory restructuring and realignment, and

[2] Thanks to Pat Worley of Oak Ridge National Laboratory for this coinage.

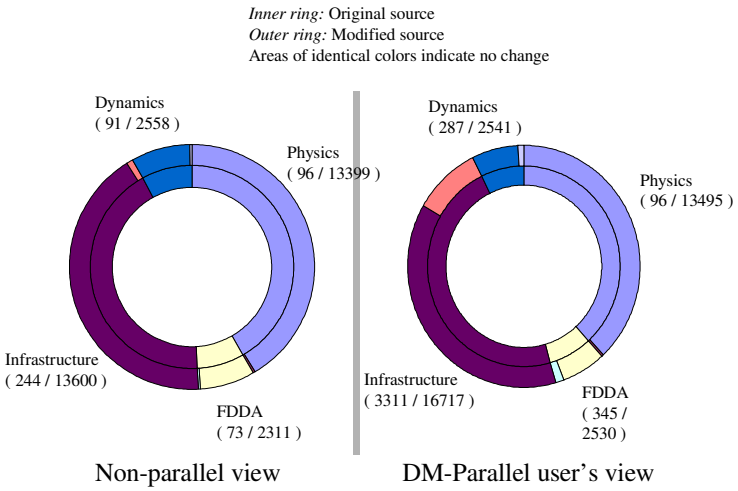

Figure 2 Impact on source

other manipulations are all effective code transformations for addressing single-processor cache performance, data locality, and communication cost. Source translation and analysis tools also uncover data dependencies in parallel routines (Friedman et al, 1995; Kothari, 1996). Finally, source translators may be used for nonperformance-related code transformations, such as adjoint generation for sensitivities and four-dimensional variational assimilation (Goldman and Cats, 1996). Source translation is a key enabling technology for the single-source development of fully integrated, fully portable models.

3. APPROACH

Parallelizing a weather model for distributed memory parallel computers involves dividing the horizontal dimensions of the domain and assigning the resulting tiles to processors. The code is then restructured to compute only the cells stored locally on each processor (by modifying DO loops and index expressions) adding communication to exchange data between processors. Hitherto, modifications have been made manually and appeared as changes to the source code. The same-source approach transfers the responsibility for making these changes to an automatic tool, the source translator, and in the process removes these changes from view of code developers, maintainers, and users.

INCREMENTAL DEVELOPMENT

We exploit a useful dichotomy to provide an incremental development path for implementing the same-source parallel option to MM5. This is as follows: communication is hard to design but easy to implement; computational restructuring for parallelism is easy to design but hard to implement.

Identifying where to put communication in MM5 and what fields to exchange is conceptually demanding but mechanically quite simple. Designing communication required painstaking manual inspection of the MM5 source code. Even so, actually adding communication was a trivial modification that, by itself, had almost no impact on the code. In contrast, modifying a code computationally is conceptually straightforward but

mechanically quite demanding. Although there are only a few simple rules for identifying and translating DO loops and index expressions for distributed memory, there are literally thousands of instances in more than two hundred subroutines, far too many for manual modification. Thus, there is a large advantage to be gained even from an automated approach that, initially, addresses only the extensive, error-prone computational restructuring of a code and leaves dependency analysis and communication, which are also amenable to automation, for a subsequent phase.

SOURCE TRANSLATOR

The Fortran Loop and Index Converter (FLIC) (Michalakes, 1997a) is a Fortran compiler with a special purpose back-end for generating the modified code. Because it employs full lexical, syntactic, and semantic analysis of the input Fortran, it is able to transform the code with minimal direction. Further, the information applies to all the files in the source code, providing extreme economy of expression in directing the translation.

From this information, FLIC examines array references within loops and infers which loops are over decomposed dimensions, it uncovers instances where decomposed dimensions are indexed by loop-invariant expressions and generates global to local index translations, and it uncovers instances where expressions of parallel loop variables are used in conditional expressions and generates local to global index translations. FLIC generates intermediate translations that are then mapped to the architecture-specific form, as shown in Figure 1. The multi-stage approach provides flexibility for adapting to the different architectures.

A research prototype of FLIC was constructed early in 1997 and has been used to parallelize a large subset of the current MM5. During this time, NCAR has made four releases of the model (from release 2 to the current release 6). Because the development version is so close to the official model, we have updated the model using automatic CVS updates on each occasion.

4. RESULTS

Apart from basic correctness (which is shown by bit-for-bit agreement with the source model), there are two other significant criteria for evaluating success: impact on software and model performance.

The impact on software is extremely small, especially from the point of view of the nonparallel user. Of the 32,000 lines in the model that have been addressed so far, the UNIX diff utility reports 504 lines are different (left half of Figure 2). This view of the code is significant because changes are out of the way of non-parallel users and code developers. One need not even install the DM parallel components, in which case the model is effectively the MM5 code as it exists today.

The right half of Figure 2 shows the parallel user and developer's point of view: the actual number of changes for distributed memory. Physics is virtually unaffected: only 96 of the total 13,495 lines in the parallelized subset are different. In other words, NCAR developers already wrote the parallel physics. Dynamics, which includes communication, is affected slightly more: 287 lines of a total 2,541. Infrastructure, which includes I/O and initialization, effects only 3,300 of a total 16,700. This is due largely to changes relating to distributed I/O, something FLIC does not address. Similarly, the FDDA nudging code is

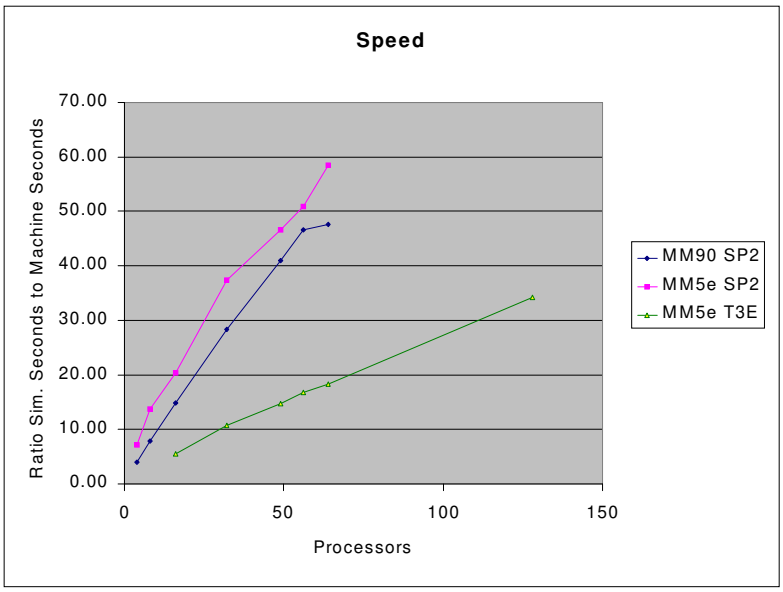

Figure 3 Preliminary performance data

affected because it also includes I/O and several large data reduction operations that FLIC does not, at present, handle.

Early performance data is shown in Figure 3. The results were gathered using the IBM SP at Argonne and the Cray T3E at NERSC (Lawrence Berkeley National Laboratory). MM90 runs on the SP are shown for comparison. The scenario was a 111 by 181 grid over the continental United States. Horizontal resolution was 27km, and there were 25 vertical layers; the time step was 90 seconds. Computational cost was 3.7 billion floating-point operations per time step. On the SP, MM5e was faster compared with MM90 (speed is shown as a ratio of seconds simulated to wall-clock run time). Scaling was worse, perhaps owing to the more sophisticated dynamic load-balancing mechanism in MM90 and also to the relative newness of MM5e. On the T3E, the model gives much better scaling, in this case from 16 to 128 processors, but absolute speed lagged behind the SP.

5. CONCLUSION

We have described an effort that will expand the set of architectures that will run the official NCAR version of the MM5, providing the benefit of scalable performance and memory capacity for large problem sizes to users with access to distributed memory parallel computers. The same-source approach uses source-translation technology for adapting MM5 in a way that does minimum violence to the code, simplifying maintenance and allowing new physics modules to be incorporated without modification. The fact that MM5 is a fully explicit model is a convenient simplification that may not be available in other models, many of which imply implicit methods in their horizontal dynamics (Baillie et al, 1997). Future work involves adapting and expanding this approach to incorporate other computational techniques, including spectral, semi-implicit, and other methods with nonlocal data dependencies. Another focus will be on augmenting source code analysis and translation to address cache and other performance portability issues. Same-source tools and techniques provide a reasonable approach to obtaining good performance over the range of high-performance computing options from a single version of the model source code.

REFERENCES

Baillie, C., J. Michalakes, and R. Skâlin, 1997: *Regional Weather Modeling on Parallel Computers*, Parallel Computing, (to appear, December 1997).

Cipra, B. A., 1997: "In scientific computing, many hands make light work," SIAM News, 30.

Foster, I. and J. Michalakes, 1993: *MPMM: A Massively Parallel Mesoscale Model*, in Parallel Supercomputing in Atmospheric Science, G. R. Hoffmann and T. Kauranne, eds., World Scientific, River Edge, New Jersey, pp. 354--363.

Friedman, R., J. Levesque, and G. Wagenbreth, 1995: *Fortran Parallelization Handbook*, Applied Parallel Research Inc., Sacramento.

Goldman, V. and G. Cats, 1996: *Automatic adjoint modeling within a program generation framework: A case study for a weather forecasting grid-point model,* in Computational Differentiation, M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1996, pp. 184-194.

Grell, G. A., J. Dudhia, and D. R. Stauffer, 1994: *A Description of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5),* Tech. Rep. NCAR/TN-398+STR, National Center for Atmospheric Research, Boulder, Colorado.

Hammond, S., R. Loft, J. Dennis, and R. Sato, 1995: *Implementation and performance issues of a massively parallel atmospheric model*, Parallel Computing, 21, pp. 1593-1619.

Hempel, R., and H. Ritzdorf, 1991: *The GMD Communications Library for Grid-oriented Problems,* Tech. Rep. GMD-0589, German National Research Center for Information Technology.

Koelbel, C., D. Loveman, R. Schreiber, G. Steele, and M. Zosel, 1994: *The High Performance Fortran Handbook*, MIT Press, Cambridge.

Kohn, S. R., and S. B. Baden, 1996: *A Parallel Software Infrastructure for Structured Adaptive Mesh Methods*, in Proceedings of Supercomputing '95, IEEE Computer Society Press.

Kothari, S., 1996: *Parallelization Agent for Legacy Codes*, draft technical report, Iowa State University, Ames, Iowa, 1996. See also http://www.cs.iastate.edu/kothari.

Michalakes, J., 1997a: *FLIC: A Translator for Same-source Parallel Implementation of Regular Grid Applications*, Tech. Rep. ANL/MCS-TM-223, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois.

Michalakes, J., 1997b: *MM90: A Scalable Parallel Implementation of the Penn State/NCAR Mesoscale Model (MM5)*, Parallel Computing (to appear); also preprint ANL/MCS- P659-0597.

Michalakes, J., 1997c: *RSL: A Parallel Runtime System Library for Regional Atmospheric Models with Nesting*, Proceedings of the IMA workshop "Structured Adaptive Mesh Refinement Grid Methods," (to appear); also preprint ANL/MCS-P663-0597.

OpenMP Architecture Review Board, 1997: *OpenMP: A Proposed Standard API for Shared Memory Programming,* Tech. Rep. Available on http://www.openmp.org/openmp.

Parashar, M., and J. C. Browne, 1995: *Distributed dynamic data-structures for parallel adaptive mesh-refinement*, Proceedings of the International Conference for High Performance Computing, pp. 22--27.

Rodriguez, B., L. Hart, and T. Henderson, 1995: *A Library for the Portable Parallelization of Operational Weather Forecast Models,* in Coming of Age: Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology, World Scientific, River Edge, New Jersey, pp. 148--161.