

PERFORMANCE–PORTABILITY AND THE WEATHER RESEARCH AND FORECAST MODEL

J. Michalakes
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439
+1 303 497 8199
michalak@mcs.anl.gov

R. Loft
Scientific Computing Division
National Center for Atmospheric Research
Boulder, Colorado 80301 U.S.A.
loft@ucar.edu

A. Bourgeois
Mesoscale and Microscale Meteorology Division
National Center for Atmospheric Research
bourgeoi@ucar.edu

Keywords: performance–portability, profiling, numerical weather prediction

Abstract

Performance and portability are important but conflicting concerns in the development of the Weather Research and Forecast model, a next–generation community mesoscale model for numerical weather prediction and atmospheric research. Efficiency depends on the ability to realize significant percentages of peak processor performance, yet without hyper–engineering the codes for a particular brand or even type of CPU. This involves first developing a quantitative understanding how aspects of the software, in particular data and looping structures, affect performance and then engineering the code to enable flexible tuning of these aspects across a variety of different platforms in a single source code. This paper describes work to characterize the performance effects of WRF model loop and data structures on representative micro– and vector–processors.

INTRODUCTION

Numerical simulation of the atmosphere for weather forecasting is one of the first applications of high–performance computing and, in terms of impact and relevance to the public, remains one of the most important today. Despite the continuous increases in computing power up to the current advent of tera–scale computing systems, the thirst for computation continues: resolutions become finer, domains become larger, time scales become longer, and complexity of models and assimilation systems continues to grow. It is as important as ever to use high–performance systems efficiently, which argues for careful engineering and tuning of models to obtain reasonable percentages of theoretical peak performance. Arguing in the other direction, however, is the cost of developing and maintaining production–quality community modeling software in the face of a diverse and dynamically changing computer architecture landscape. Balancing these conflicting concerns for performance on the one hand and maintainable software on the other is a principal challenge that is being addressed in the design and implementation of the new Weather Research and Forecast (WRF) model. This paper focuses on WRF performance–portability across the two prevailing processor types: micro– and vector–

processors. Aspects of the WRF design for portability across shared-memory, distributed-memory, and hybrid parallel architectures are described in Michalakes et al. (1998, 2001).

Tuning for processor performance involves measuring the effect of particular controllable aspects of the program and then modifying the program accordingly. The extent to which an aspect is controllable for different platforms depends on the effort required and the degree to which such adjustment affects the maintainability, extensibility, understandability, and portability of the program. Run-time and compiler flags, use of fast libraries, and other modifications that require no modification of the code itself are, of course, easy. Modifications such as adjustments to padding, alignment, sizing of arrays, and blocking factors for loops are still relatively straightforward, provided one has the ability to design specifically for this in a new code. In WRF, dynamic allocation of data structures, the tile-callable interface between model and driver layers, and multi-level domain decomposition provide a great deal of flexibility for dealing with these issues at run-time without changes to the code. Finally, there are issues such as storage order and loop nesting order that are very difficult to modify once established within a code.

The next section of this paper, , provides an overview of the WRF development project and the role of model performance and software architecture in the overall effort. The section describes a first set of whole-code WRF experiments that provided initial information on the effect of storage order on code performance and portability. The section describes a second set of experiments that involved performance measurement of computational kernels in WRF to develop a more complete characterization of performance effects, together with an understanding of specific causal mechanisms. SUMMARY discusses implications for ongoing WRF model development.

BACKGROUND

The WRF project is developing a next-generation mesoscale forecast model and assimilation system with advanced numerics and improved physics to advance both the understanding and the prediction of mesoscale systems and promote closer ties between the atmospheric research and operational forecasting communities. A large number of U.S. organizations, including the National Center for Atmospheric Research, a number of National Oceanic and Atmospheric Administration laboratories, the Environmental Protection Agency, NASA, Air Force Weather Agency, Naval Research Laboratory, the University of Oklahoma, and many other universities are participating in the WRF project. The model is intended for a wide range of applications, including forecasting, atmospheric chemistry, regional climate, and idealized simulations for research with priority emphasis on horizontal grids with 1–10 kilometer resolution. A WRF prototype was first released to the community in the Fall of 2000. A fully implemented research quality version of the WRF system, providing multiple relocatable interacting nests and variational data assimilation, is to be released in 2002; a fully operational numerical weather prediction (NWP) version will follow in 2004. Based on its merits, WRF will be a candidate to replace existing forecast models such as the Penn State/NCAR Mesoscale Model (MM5), the Eta model at NOAA/NCEP, and the RUC system at NOAA/FSL.

The issue of whether to design WRF exclusively for microprocessor based systems or to maintain compatibility with vector systems was the subject of controversy within the WRF collaboration. Without exception, all of the WRF participants have transitioned from vector supercomputers to scalable, microprocessor-based systems. This has occurred for both technical

(e.g. scalability and cost–performance) and non–technical reasons.¹ Furthermore, the WRF model is intended not only for "big–iron" operational users, but also for university and other researchers with access to more modest high–end workstations or PC–clusters.

On the other hand, WRF, like MM5, is being used internationally, where vector systems are more widely deployed. Also, the generally held belief that vector and RISC coding practices are antithetical had not been rigorously examined in the context of the WRF application. It was therefore decided to study the issues and possible penalties involved with keeping vector systems in the stable of workhorse architectures for WRF.

The storage order of vertical (K), west–east (I), and south–north (J) axes in three–dimensional arrays and the corresponding loop nesting order are crucial issues for the design of NWP codes. Conventional wisdom, bolstered by some scattered experience, held that a KIJ ordering of array data and of loops is optimal for RISC processors, based on the assumption that having the vertical K–dimension stride–one/innermost allows entire vertical columns of the domain to fit into cache. Compared to IJK implementations, KIJ physics implementations reduce horizontal IJ size temporary arrays to scalars that can be stored in registers, thereby reducing the memory footprint and memory traffic relative to IJK formulations. At the same time, KIJ ordered column physics does have potential performance drawbacks on RISC microprocessors. First, KIJ inhibits vectorization of intrinsic math functions, such as EXP and LOG, which occur frequently in radiative and thermodynamic equations. For example, vectorizing MASS libraries on the IBM reduce the cost of an exponential by a factor of three (Andersson, et al. 1998). Second, modern pipelined microprocessors with multiple functional units may be unable to take advantage of performance gains associated with inner loop unrolling in the KIJ order because of dependencies in the vertical.

Conventional wisdom also held that KIJ ordering is anathema to vector processors because K is the shortest dimension in an atmospheric model domain and because the K dimension has data dependencies that break vectorization, particularly in model physics. It was axiomatic that vector systems preferred IJK or, better still, L_{ij} K data and loop organizations², because these provide long, dependency–free stride–one dimensions needed for efficient vectorization. Unlike variable blocking factors and other aspects that could be allowed for in the code design, the choice of storage and loop order, once made, would be difficult to change.

A 1997 study of the effects of non–optimal storage order on vector and RISC–based systems running a geophysical kernel code by Michael Ashworth of Daresbury Laboratory, U.K., provided evidence to support the idea that IJK was better on vector and KIJ was better on microprocessors, but with the additional quantitative result: RISC systems were much more tolerant of vector–friendly code organizations, suffering considerably smaller penalties than did vector systems in the reverse case (Ashworth, 1998). Our tentative conclusion was that if WRF could target only RISC systems, the K–innermost ordering was the clear choice; however, if we intended to support both RISC and vector with a single code we would need to consider an I–innermost ordering — provided the penalty on RISC was not too great (say, less than 30 percent). To inform the decision for WRF, a recapitulation of Ashworth’s study was conducted

¹ Certainly, a major factor in this trend was the effective demise of the U.S. vector supercomputing industry combined with the fact that purchasing foreign–manufactured vector supercomputers has not been an option available to U.S. government or quasi–governmental institutions, especially in light of tariffs imposed on import of these systems. At this writing, the effect of the May 3, 2001 action by the U.S. International Trade Commission to rescind the anti–dumping order and tariffs against Japanese vector supercomputers and the subsequent agreement allowing Cray Inc. to market the NEC SX series of vector supercomputers in North America is unclear.

² L_{ij} means the horizontal I and J dimensions are combined into a single 1–D vector.

using both WRF kernel codes and full-up versions of the, then, relatively small dry (virtually no physics) WRF model prototype. These were cast in IJK, KIJ, and one new storage order, IKJ.

Based on the results of the study, which was conducted using Compaq Alpha EV56 and EV6, IBM Power3, and Fujitsu VPP5000 processors, an IKJ ordering was chosen for WRF in April, 2000.³ Since that time, with the addition of new dynamics, many model physics packages, lateral boundary conditions, and I/O, the WRF code has grown considerably, so that recasting the entire code for the purpose of studying storage order is now infeasible. However, a more thorough investigation has continued using computational kernels from WRF.

WRF STORAGE ORDER

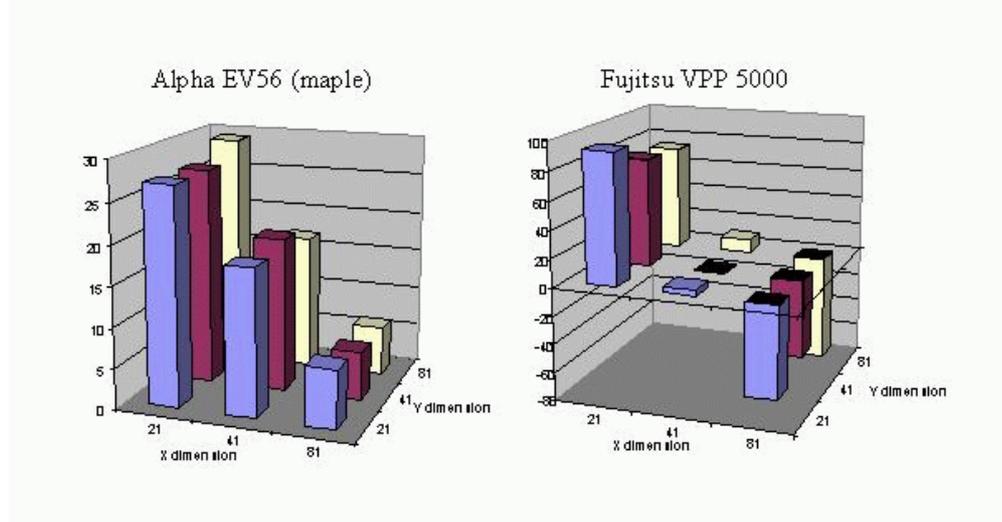


Figure 1. Comparison of KIJ ordered WRF code to IJK ordered code on Alpha EV56 processor and on Fujitsu VPP5000 for a number of domain sizes. Positive values indicated the KIJ code is favored

³ <http://box.mmm.ucar.edu/mm5/mpp/wrresults>

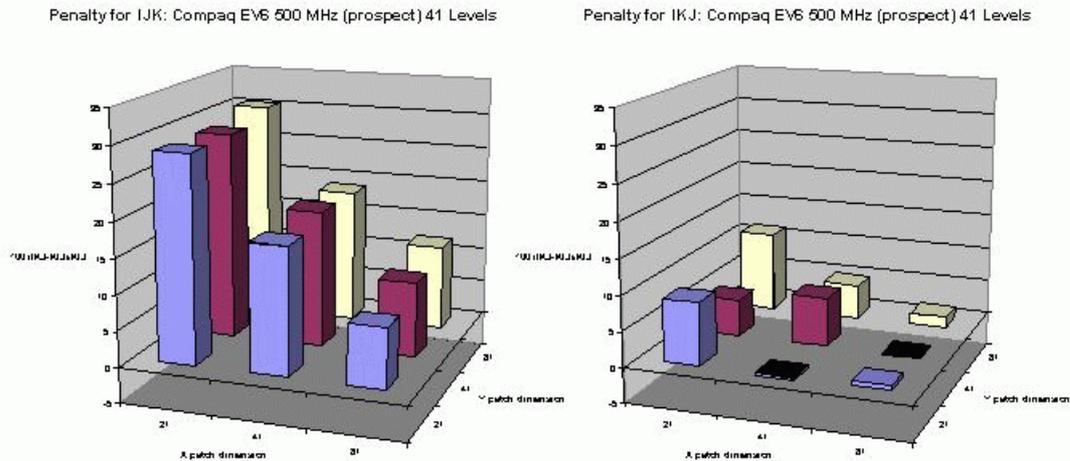


Figure 2. Alpha EV6 comparisons of IJK ordered code to KIJ ordered code (left) and of IKJ ordered code to KIJ ordered code (right). Positive values indicated the KIJ code is favored over the other code. The penalty for choosing an IKJ ordering instead of KIJ is considerably less than for IJK.

A series of experiments was conducted to reach an expeditious decision on storage order and loop nesting order for WRF. Initially, two versions of the dry prototype code, one using IJK ordering and the other KIJ, were written. The outputs from these were verified to be correct with respect to known solutions of idealized cases and to agree bit-for-bit with respect to each other. Array dimensions in the IJK version were ordered such that the west-east (I) dimension was stride-one, the south-north (J) dimension was middle, and the vertical (K) index was the slowest varying. Similarly, all triply nested loops were written so that the west-east loop was innermost, south-north middle, and bottom-top outermost. The KIJ code had the vertical dimension as stride-one/innermost.

Tests of the codes were then performed for several different domain sizes on several different machines: a Compaq workstation using an Alpha EV56 processor, a Compaq ES40 node using an EV6 Alpha processor, an IBM SP with Power3 processors, and a Fujitsu VPP 5000 processor. Figure 1 shows the penalty for IJK ordering compared with KIJ ordering on a microprocessor (left) and a vector machine (right). The Z axis on these plots is penalty shown as a percentage: $(T_{\text{slow}} - T_{\text{fast}}) / T_{\text{fast}} \times 100$. If the bar is going up KIJ is faster and T_{slow} is the time for the IJK ordering; if down then T_{slow} is time for the KIJ ordering.

These results bear out Ashworth's findings that KIJ ordering is favored on microprocessor machines and the IJK is favored on vector machines. Further, the penalty for the "wrong" ordering is more severe on the vector machine. Penalties for the vector-friendly IJK ordering on RISC are 25–30 percent for smaller subdomains to around 5–10 percent for larger domains. This is a combined effect of the KIJ performance decreasing slightly as the subdomain size increases and the IJK performance actually improving slightly as the domain size increases in the minor dimension. On the vector processor, the penalty for using KIJ is in the 60–80 percent range, assuming subdomain sizes are large enough in their horizontal dimensions. An unexpected result was that the vector processor prefers KIJ for small domains when the K-dimension is longer than the I-dimension.

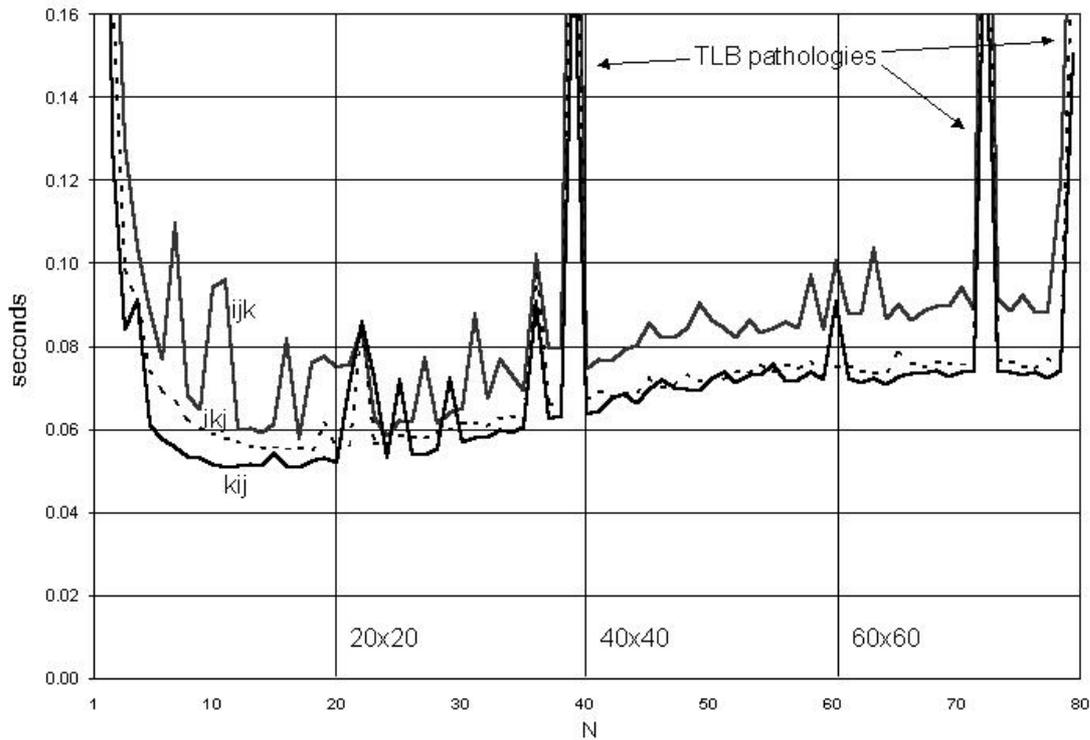


Figure 3. CPU time per 6400 cells to compute square $N \times N$ (41 level) domains ranging for $N=1$ through 80 for IJK, IKJ, and KIJ storage orders. Peaks are the result of TLB-miss pathologies and range between .2 and .3 seconds for IJK, between .19 and .26 seconds for IKJ, and between .14 and .18 for KIJ.

Based on these results, and considering the size of the penalty on RISC for the vector-friendly IJK ordering, it appeared that WRF would need to abandon vector systems and simply adopt a K-inner ordering. It was at this point, however, that co-author Loft suggested trying an IKJ ordering, structuring the array data into J number of IK slabs. A third set of experiments was run, this time comparing the penalty for using IJK instead of KIJ with the penalty for using IKJ instead of KIJ (Figure 2). The penalty for using IKJ was found to be considerably less than the penalty for IJK, and IKJ provides stride-one in the long, vector-friendly I dimension. Based on these results, the decision was made in April 2000 to proceed with an IKJ storage and loop order for WRF; the other two prototypes were abandoned. It was also recommended that, as time permitted, feasibility be investigated for using source translation to automatically reorder array dimensions and loop nesting.

DETAILED PROFILING

The WRF implicit solver for vertical propagation of acoustic modes, `ADVANCE_W`, performs a tri-diagonal solve for each vertical column in a WRF domain. It is interesting as a kernel for study because it is computationally significant in WRF, it is fully three-dimensional, and it includes recurrences in the vertical dimension. Though part of a WRF dynamical core, `ADVANCE_W` is similar computationally with many WRF physics routines in that it has potentially vector-unfriendly recurrences in the vertical dimension. The `ADVANCE_W` calculation performs an average of approximately 125 floating point operations, adds and multiplies counted separately, per three-dimensional grid cell using 20 three-dimensional arrays, 3 two-dimensional arrays, several one-dimensional (vertical) arrays, and a number of zero-dimensional scalars.

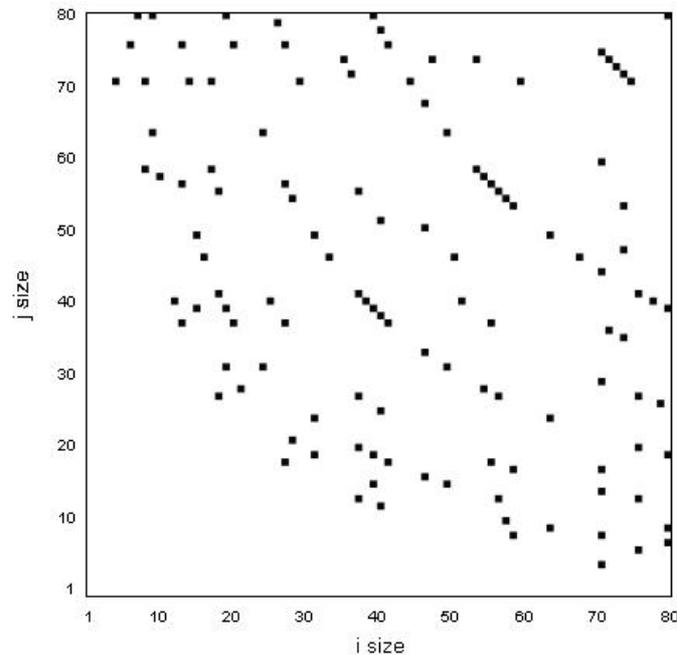


Figure 4. Number of TLB misses per 6,400 grid points as test domain size is varied from 2x2 to 80x80 in both horizontal dimensions. Typically, TLB misses number $O(10^4-10^5)$; pathological points have $O(10^6)$ or greater.

A small, stand-alone test driver program inputs domain and pad size information, dynamically allocates and initializes the arrays from a small file of data generated from a WRF run, then calls `ADVANCE_W`, passing the arrays and all dimension information as arguments. As specified in the WRF software design, dimension information for logical domain size, memory size, and run-length are passed separately and unambiguously as starts and ends of each logical domain dimension, local memory dimension, and run-length. Calls to instrumentation subroutines that access CPU hardware event counters appear before and after the call to `ADVANCE_W` in the test driver. The instrumentation, from the Htrace library by John Dennis of NCAR Scientific Computing Division (Dennis, 2001), provides a variety of statistics, depending on the processor. IBM, Sun, and Intel CPUs are supported. Three versions of the `ADVANCE_W` routine, each with a different storage order and loop nesting order, were developed and tested: IJK, IKJ, and KIJ. In each experiment, the instrumented code was run 6241 times, once for each domain size varying the two horizontal dimension sizes from 2 to 80 cells, inclusive. Each series runs was conducted with horizontal array pad widths of 0, 1, 2, and 3 cells. Each test was invoked as a separate run in which only one instrumented call to `ADVANCE_W` was performed; this established a “cold cache” condition that more realistically simulates the embedded behavior of `ADVANCE_W` in a full model.

The experiments presented in this paper were run on one processor of a four-processor IBM Power3 Winterhawk-II node of a system at NCAR (babyblue.ucar.edu), using FORTRAN compiler version 7.1.1 and `-O3` optimization. The AIX operating system of this machine was modified with special IBM-supplied patches that provide access to the CPU hardware counters (Andersson et al., 1998 and personal communication). The statistics gathered for each runs were number of processor cycles, number of floating point instructions, number of load instructions, number of level-one cache misses, number of Translation Look-aside Buffer (TLB) misses,

number of prefetches, and other information. Very high resolution (2.8 nanosecond) timings of the ADVANCE_W test routine were collected by directly measuring the number of processor cycles and then divided by the number of cycles per second (375×10^6). In general, detailed timing results bear out the earlier study of WRF storage order in that the execution time for the KIJ ordering is nearly always best, IKJ usually runs a close second, and IJK is nearly always worst. Figure 3 shows comparative timings for the set of square N by N domains, $N=2..80$, normalized to an 80 by 80 domain.

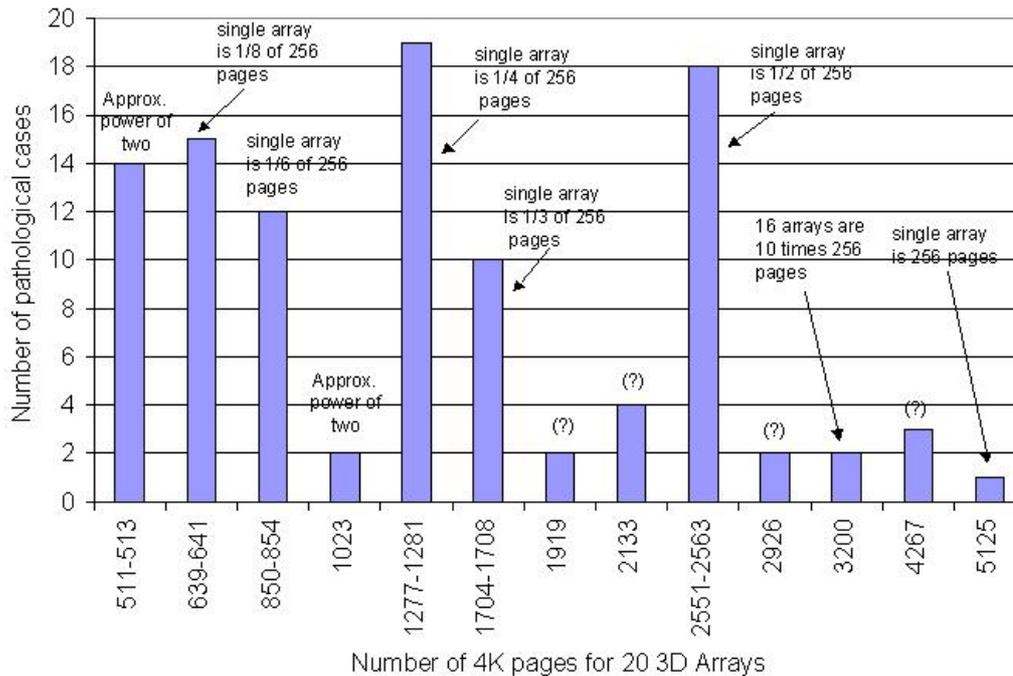


Figure 5. Histogram showing clustering of pathological sizes.

The most significant degradation of performance occurs for all three storage orders for domain sizes that exhibit very high numbers of misses in the processor's Translation Look-aside Buffer. The TLB maps virtual memory addresses to pages in physical memory. The IBM Power3 TLB holds 256 page entries using two-way set associativity and least-recently-used replacement. Cost of a TLB miss is "between about 25 cycles if the relevant parts of the page and segment tables are in L2 cache, to possibly hundreds of cycles in unfavorable cases." (Andersson, et al., 1998) Typically, a call to ADVANCE_W involves between 10000 and 20000 TLB misses per 6400 vertical grid columns. Certain pathological configurations, however, generate upwards of 1.5 million TLB misses. Figure 4 shows in black the distribution of pathological domain sizes according to I-size and J-size of the domains. The pattern is perfectly symmetrical about the diagonal. Although intensity of the pathological miss rates is highest for IJK and lowest for KIJ, the pattern itself is identical for IJK, IKJ, and KIJ ordered-codes, The pattern is also unaffected by padding of arrays, except that the pattern shifts to the left and down with each additional row and column of cells around the subdomain. Also with that shifting, another "chime" becomes apparent at the upper-right of the plot (one element of this chime is visible at (80,80) in the zero-padded plot shown in the figure.)

Figure 5 shows the frequency distribution of TLB pathology points according to the total amount of data accessed by the ADVANCE_W routine, measured in four kilobyte pages. Two sets in the

distribution have numbers of pages that are approximately powers of two: 512 and 1024. Six others — 639–641, 850–854, 1277–1281, 1704–1708, 2551–2553, and 5125 — have numbers of pages that are approximately integer fractions of 20 times the TLB capacity (ADVANCE_W uses 20 three-dimensional arrays). The relationship between set size and TLB size in the remaining four cases is not immediately obvious, but almost certainly also produce array-in-memory alignments that cause accesses to corresponding elements of different arrays (and different 4 kilobyte pages) to map to the same entry of the TLB, generating a TLB miss for virtually every access. Although the impact of these pathological domain sizes is serious in terms of overall run time, the fact that the effect is regular, predictable, and depends on the sizes of local arrays suggests a relatively simple run-time mechanism for avoiding these sizes by introducing extra padding when the arrays are allocated.

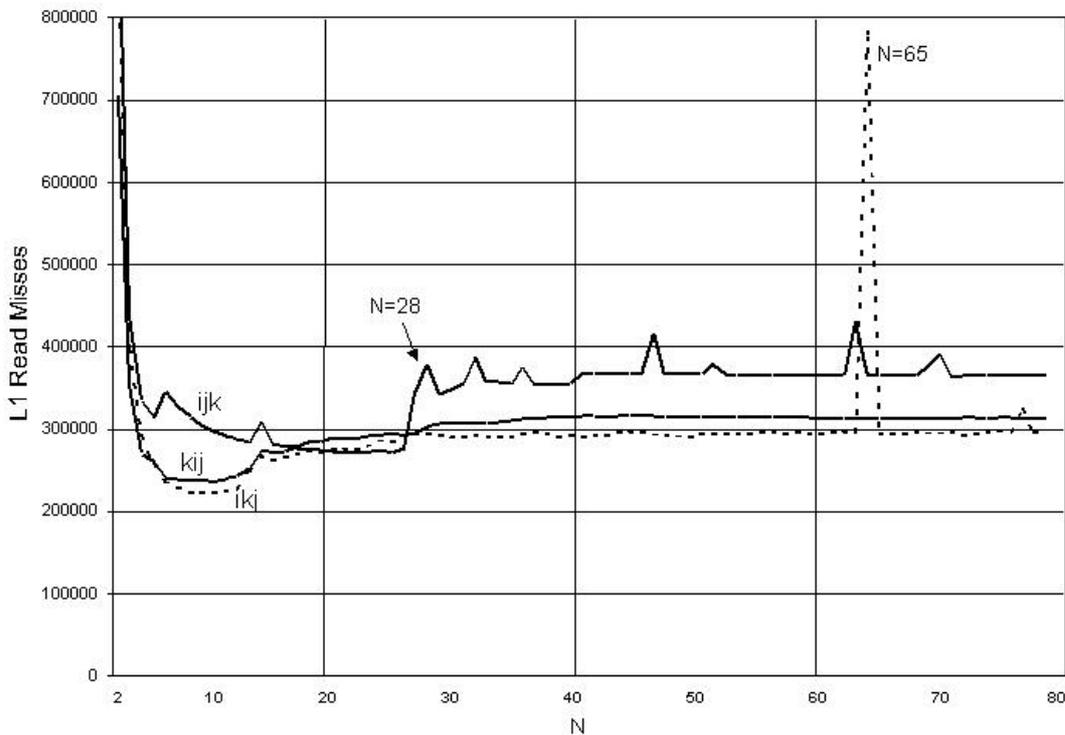


Figure 6. Level one cache misses per 6400 cells in computation of ADVANCE_W on square $N \times N$ (41 level) domains ranging for $N=1$ through 80 for IJK, IKJ, and KIJ storage orders. Peak at $N=28$ for IJK ordering occurs when the size of horizontal slices through 20 3D arrays plus the size of the 3 2D arrays in the computation exceeds 64 KB. The spike at $N=65$ for the IKJ layout is likely the effect of cache conflicts induced by the copious vertical stencils in ADVANCE_W.

The Power3 L1 data cache is 64 kilobytes, with 128-byte cache lines arranged into four 128-way set associative banks. The cost of read miss to L1 cache is 7 cycles if the data is in L2 cache, or 35 cycles if the data is in memory only (Andersson, et al., 1998) The effect of domain size and storage order on read misses to the level one (L1) data cache is shown in Figure 6. Other variation in the plots, particularly the smaller scale spikiness in the IJK plot, is not fully explained by the counter statistics that have been gathered to date. Disregarding spikiness in the plots, all three storage orders show an apparent "sweet spot" centered around domain sizes of 20×20 , and there is a relatively smooth degradation as N increases. The absence of obvious "steps" is somewhat surprising. If the ability to fit entirely within a particular level of cache is a factor, one expects relatively sharp increases around the transitional domain sizes. Furthermore, a step that appears at $N=28$ in the L1 cache miss plots for the IJK ordered code (Figure 6) is not

reflected in the timing plots. In the cases of IJK and IKJ ordered codes, the shapes of the timing plots appear inversely related to the amount of prefetching that occurs (Figure 7). The Power3 processor begins prefetching cache lines after the second miss in a consecutive cache line and one therefore expects prefetching to remain steady or to increase with the length of the stride-one dimension; this is not the case in the data. Additional work is needed to understand these issues.

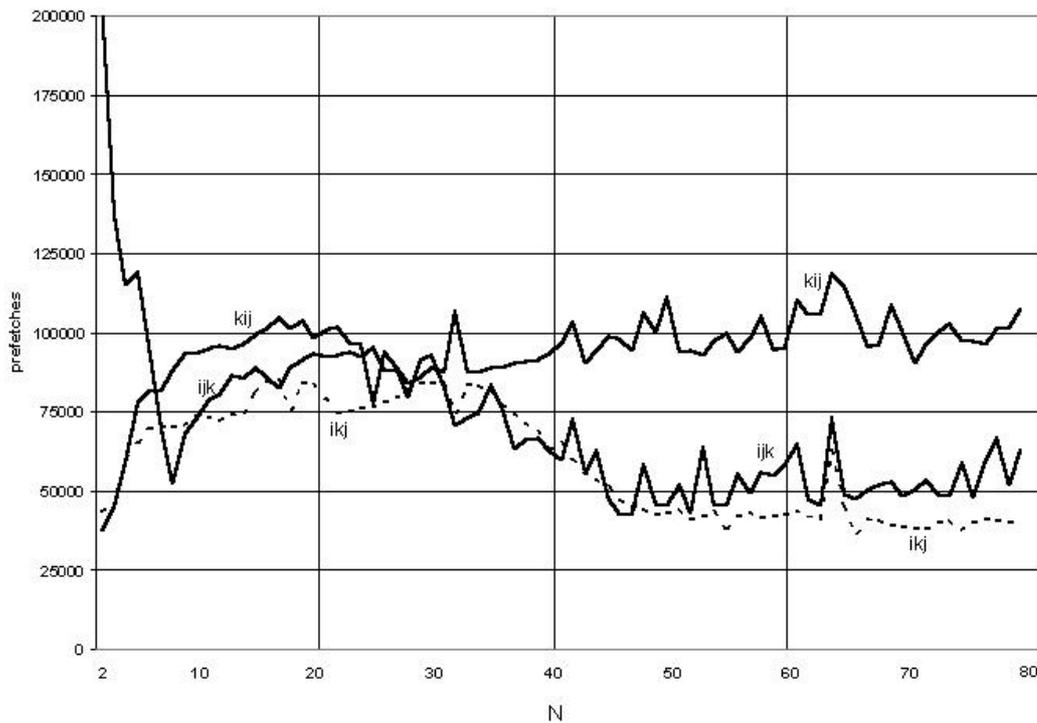


Figure 7. Number of cache-lines prefetched per 6400 cells in computing square $N \times N \times 41$ domains.

SUMMARY

Designing and implementing weather models for performance–portability is crucial to the efficient utilization of diverse computing platforms. The results to date for the Weather Research and Forecast model suggest that storage and loop order is an important factor for processor performance, and that I–innermost storage orders are essential for good vector performance, microprocessors are tolerant of this ordering, and that a K–middle ordering such as IKJ represents a good compromise that performs nearly as well as K–innermost on microprocessor machines, while preserving I as a stride–one innermost dimension for efficiency on vector systems. On IBM microprocessor based systems, the most serious effect on performance is large numbers of misses in the Translation Look–aside Buffer caused by pathological alignments of arrays in the TLB. The alignments are solely dependent on array sizes, suggesting a straightforward remediation strategy that can be employed at runtime to pad arrays away from pathological sizes when they are dynamically allocated in WRF. The next most important effect is observed to be correlated with the processor’s ability to employ prefetching of successive cache lines. The effect tends to favor local array sizes above a very small minimum up to an upper limit such that $15 < N < 35$ for square $N \times N \times 41$ domains in these experiments. Additional research is required to determine (a) why the prefetch rate trails off above $N=35$ for IJK and IKJ orderings and (b) the causal relationship, if any, between the prefetch rates and observed timings. Level–one data cache miss behavior is most pronounced with IJK, which exhibits a pronounced capacity miss for domains where $N=28$ and above. However, the effect of this step is not reflected in timing data, suggesting that L1 data cache miss rates by themselves are not as significant a factor for this computational kernel as originally expected.

REFERENCES

- ANDERSSON, S., BELL, R., HAGUE, J., HOLTHOFF, H., MAYES, P., NAKANO, J., SHIEH, D., AND TUCCILLO, J. (1998): RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide. IBM International Technical Support Organization Redbook, SG24–5155–00.
- ASHWORTH, M. (1999): Optimisation for vector and RISC processors. *Towards Tera–computing*. World Scientific, River Edge, New Jersey, pp. 353–359.
- DENNIS, JOHN (2001): NCAR, Personal communication.
- MICHALAKES, J., DUDHIA, J., GILL, D., KLEMP, J., AND SKAMAROCK, W. (1999): Design of a next–generation regional weather research and forecast model. *Towards Tera–computing*. World Scientific, River Edge, New Jersey, pp. 117–124.
- MICHALAKES, J., CHEN, S., DUDHIA, J., HART, L., KLEMP, J., MIDDLECOFF, J., SKAMAROCK, W. (2001): Development of a next–generation regional weather research and forecast model. Argonne National Laboratory preprint. ANL/MCS–P868–0101, 8 pp.