3.45

WRF Module Optimization Study

Thomas Nehrkorn* and George D. Modica

Atmospheric and Environmental Research, Inc.
Lexington, Massachusetts

1 INTRODUCTION

The work reported here was part of a broader effort directed at increasing the computational efficiency of the WRF model and related ancillary programs to improve execution times for the Air Force Weather Agency's (AFWA) planned operational implementation of the WRF model.

We concentrated on two main components of the WRF modeling system: the standard initialization package (WRFSI), and the forecast model (WRF). The results of the WRFSI are briefly summarized in this abstract, while those for the WRF model are presented in some detail. For each component, the most recent version—as implemented at AFWA—was ported to a similar computer system located at a Department of Defense High Performance Computing (HPC) Major Shared Resource Center (MSRC) located at the Naval Oceanographic Office (NAVO), Stennis Space Center, MS. The ported code was subjected to a CPU time profiling analysis (Sections 2 and 3). The results of these analyses served to identify candidate modules for optimization. The results of the optimizations are described in Section 4.

2 WRF PROFILING

Version 2.0.3.1 (as released on 6 Dec 2004) was used, with a modified version of the Registry.EM file. Aside from minor stylistic changes, this version of the WRF model is identical to the version used on the HPC Distributed Center development platform at AFWA. The model was compiled on the "marcellus" platform at the NAVO MSRC, an IBM pSeries 690 148 node SP cluster, with each node containing eight 1.3 GHz Power4 processors.

The model was configured for a nested grid configuration with a 301 x 238 outer grid with 15-km grid spacing, and an inner 268 x 244 grid with 5-km grid spacing, and 42 levels in the vertical. The outer grid was advanced with a 60 sec time step, the inner grid with a 20 sec time step, using feedback from the inner to the outer grid, and no smoothing on the outer grid. The multiprocessing run used 64 processors, with one tile per processor (8 nodes with 8 processors each). The LoadLeveler batch system scripts specified exclusive use of each node for the WRF model run, and the use of shared memory for inter-processor communication on each node. History output files were written once an hour on the inner nest, and once every three hours on the outer grid, using the GRIB1 format option. All I/O was performed on the 9774 GB workspace file system (/scr) using the GPFS (General Parallel File System).

The Eulerian mass core (EM) was used with a third-order Runge-Kutta time stepping scheme, with diff_opt=1 (2nd order diffusion on coordinate surfaces), with diffusion coefficients computed from horizontal Smagorinsky first order closure (km_opt=4). Vertical velocity damping was enabled (w_damping=1). Horizontal advection uses 5th order differencing, vertical advection 3rd order. Specified boundary conditions were used on the outer grid, relaxation boundary conditions on the inner grid. The physics options for the baseline WRF model configuration are given in Table 1.

WRF initial and boundary condition files for a 48-hour forecast, initiated from a WRF 3dvar analysis valid at 06 UTC 11 Nov 2004, were used for all profiling runs described here. Wind fields and cloud condensate mixing ratios are shown in Figure 1 for level .2580 (near the jet stream level, with pressures ranging from 290 to near 180 hPa for domain 1). This case is characterized by upper-level zonal flow with embedded short-wave features. Cloudiness is present at this and other levels, in both domains, associated with large-scale and convective precipitation over relatively small portions of each domain.

To verify proper compilation and execution of the WRF on the marcellus computer, a one-hour forecast was run and compared with the corresponding run on the AFWA HPC platform. The root mean square (rms) differences and normalized differences (normalized, that is, by the rms values of the full fields) were computed over the inner domain between the two forecasts (not shown). The normalized values agree to better than 10^{-2} for the horizontal winds and water vapor mixing ratio, and better than 10^{-4} for potential temperature and pressure. Because of the strong nonlinearities inherent in moist processes, and small values of the full field variables, normalized differences for liquid and solid moisture mixing ratios reach values as large as 0.4 at isolated levels, although they are considerably smaller at most levels. Both of these runs used a compilation at optimization level 2 (-O2 flag was

* *Corresponding author address:* Thomas Nehrkorn, AER, Inc., 131 Hartwell Ave., Lexington, MA, 024231-3126; email: trn@aer.com

used during compilation and linking of Fortran codes). A test run with optimization level 3 (-O3 flag) resulted in differences of very similar magnitudes.

Table 1. Baseline WRF Model Configuration

Physics setting	Description
mp_physics = 4	WRF single moment 5-class (WSM5)
mp_zero_out = 2	enforce non-negative water vapor specific humidity, and zero out other moist arrays below a threshold value
ra_lw_physics = 1	RRTM scheme
ra_sw_physics = 1 • icloud = 1 • radt = 30	Dudhia scheme with cloud effect to the optical depth in radiation minutes between radiation physics calls
sf_sfclay_physics = 1	Monin-Obukhov scheme
sf_surface_physics = 2 • isfflx = 1 • ifsnow = 0	Noah land-surface model (with 4 soil layers) with fluxes from the surface without snow-cover effect
bl_pbl_physics = 1 • bldt = 0	YSU scheme BL physics called every time step
cu_physics = 3	Grell-Devenyi ensemble scheme (using default values of parameters)
cudt = 5	minutes between cumulus physics calls

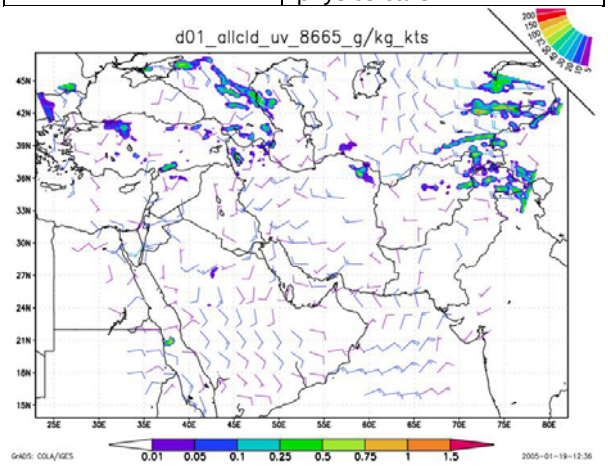

Figure 1. Winds (shown in knots, colorized according to circular color scale in upper right corner) and total cloud condensate mixing ratio (shown in g/kg) for model level 0.2580, domain 1, at 3 hours into the forecast.

3 TIMING RESULTS

Timing runs were run out to 6 hours on marcellus, using both a level 2 and level 3 optimization (-O2 and -O3). The profiling utility gprof was used in conjunction with the required -pg compiler flag. For comparison, the tprof utility was used on an executable compiled without the -pg compiler flag. Both utilities rely on sampling to determine the most CPU intensive parts of the code. The gprof profiles provide a calling tree analysis of CPU time, as well as a flat profile listing the individual routines in order of CPU time spent in each. Separate gprof profiles were generated for each of the 64 processors for each run.

An overview of the WRF call tree for processors 0 and 63 is shown in Table 2, which shows that most of the CPU time is spent in routines called by integrate, with smaller amounts (roughly 10% total) by nesting feedback and forcing, and, for processor 0, I/O.

Table 2. WRF call tree profile. "Self" and "Desc" are the CPU time (sec) spent in the routine, and the routines called by it, respectively.

Self		Descendents		Name
p0	p63	p0	p63	
0	0.02	1052.8	977.3	.__module_integrate_MOD_integrate
0.41	0.19	929.85	877.53	.solve_interface
0.04	0.06	86.67	87.85	.med_nest_force
0		18.98		.med_before_solve_io
0	0	9.41	8.56	.med_nest_feedback
0		4.7		.med_last_solve_io

Almost the entire CPU time for integrate is spent by routines called by solve_em, the main time stepping driver. Within solve_em, the most expensive parts of the computation are (in order): dry tendencies (rk_tendencies), the Cumulus parameterization, the microphysics parameterization, various parts of the time stepping computations, the PBL parameterization, the radiation package, and the surface flux parameterization. More detailed breakdowns (not shown) reveal a fairly even distribution among many routines for some packages (as rk_tendency), and fairly concentrated CPU times for others (particularly the microphysics and surface layer physics). A breakdown at the subroutine level (not shown) reveals the rankings (in terms of CPU time) and the percentage spent. This included results from gprof (for processor 0, and for CPU times summed over all processors), and from tprof.

Using the ranking of the CPU-intensive portions of the code as a guide, we examined the code for optimization opportunities. The results are summarized as follows, in order of priority:

- .__module_bl_ysu_MOD_ysu2d
- .__module_cu_gd_MOD_cup_enss
- .sintb
- .__module_small_step_em
- .__module_big_step_utilities_em

This ordering was based on a combination of the percentage of CPU consumed and perceived ease with which speedups can be obtained.

Although the single most CPU-intensive non-system routine is wsm52d (part of the WSM5 microphysics package), it was not selected for optimization because it had been previously optimized (by John Michalakes of NCAR): exponentiations (**) were replaced by (mathematically equivalent) calls to intrinsics (exp, log, and sqrt), and calls to low-level functions and routines were replaced by explicitly inlined code. A test run with these optimizations disabled resulted in approximately 10% more CPU time spent in this routine, and an overall increase of 2-3% in CPU time for our 6-hour forecast case.

The top candidate identified (routine ysu2d, which is part of the PBL package) contains expensive operations such as "**", which can be optimized using the same strategy as used in wsm52d. Routine cup_enss of the cumulus parameterization scheme was selected because a preliminary analysis identified some possible loop restructuring. The collection of routines that are called within module_small_step_em and module_big_step_utilities_em accounts for a sizable share of the CPU time, so it is worth considering for optimization. For the most part, the routines consist of fairly simple operations, so speedups can be expected only if loop restructuring can improve memory localization. Results from timing runs with calls to the IBM Hardware Performance Monitor (HPM) library indicated no clear candidates (modules with high wall clock times and low utilization rates).

4 WRF OPTIMIZATION RESULTS

Boundary Layer

The routine ysu2d accounts for most of the time spent in the YSU boundary layer package. Aside from some minor loop reorganizations (switching the nesting of i,k loops so that arrays are accessed in storage order), the optimizations in this routine concentrated on expensive floating point operations, primarily exponentiations. In the original code, exponentiations were all implemented as "**", using floating point constants or variables. In the optimized code, all exponentiations to floating point exponents were replaced by mathematically equivalent calls to the "exp" and "log" intrinsics if the argument to the log functions was positive definite. Expressions involving negative fractional exponents were replaced by divisions and use of the "sqrt" intrinsic. Integer constants were used instead of floating point constants where appropriate. In addition, we examined the effect of replacing "** n" by n multiplication operations, for n=2, 3, and 4. Sensitivity tests indicated that this optimization was beneficial for n=2, but not for higher powers. Correctness testing used rms difference statistics of 1-hour domain 2 forecast values. The rms differences were found to be of the same order as those from runs of the same code on different platforms.

The effects of optimization of the ysu2d routine were analyzed by repeating the 6-hour gprof profiling runs discussed in section 3, both with the original and optimized version of the ysu2d routine. CPU times were summed over all 64 processors for this analysis. The CPU time for routine ysu2d was reduced from 1639.71 sec to 1524.67 sec, representing a 6.9% speedup. The total CPU time for the 6-hour forecast was reduced by 2.0% (from 76741 sec to 75264 sec).

In addition, a 1-hour forecast was performed with a version of the code instrumented with calls to the HPM library. The code was compiled without the –pg profiling option for this analysis. The HPM statistics for the PBL driver section of the code is shown in Figure 2, which shows the wall clock time for each of the 64 processors over the 1-hour forecast. Although there is considerable variation from one processor to the next, the optimization results in a consistent improvement in terms of wall-clock and CPU time (by about the same percentage seen in the 6-hour profiling run), and numbers of instructions (not shown). Because the optimization resulted in fewer floating point operations, the flip rate and computation intensity (not shown) are actually slightly lower for the optimized version.

Cumulus Parameterization

The cup_enss routine identified in Section 3 is the most time-consuming routine of several routines in the cumulus parameterization module module_cu_gd. Examination of this and related routines in this module indicated numerous loops over i,k which were either not nested in memory order (the outermost index, i, was the fastest-varying array index), or which contained if-statements in the innermost loop. Because the algorithm is intrinsically column-oriented (i.e., there are various factors that vary by horizontal position represented by the innermost array-index, i), a simple reordering of loop nesting as was done in the boundary layer routine ysu2d was not possible here. Instead, we rewrote the code by reordering all local arrays such that the vertical index, k, is the fastest-varying array-index. In addition, all nested i,k loops were reordered such that the horizontal index i was the outermost loop. These changes were limited to the arrays local to the cumulus module, and the interface to the cumulus_driver is unaffected by these changes. Because this approach may not be optimal for some architectures, particularly vector processors, we also kept the original code, and defined a CPP precompiler macro that allows switching between the two versions. Correctness testing, again using rms differences between original and modified code runs out to 1 hour, showed identical results from the original and optimized versions of the code.

The effect of the optimization on 6-hour forecast CPU times from a profiling run were examined by summing the CPU times from all 64 processors for all routines of the cumulus parameterization package (module_cu_gd). The results showed a showed a 24% speedup (from 6455.71 sec to 5211.82 sec). The CPU time of the entire WRF model was decreased by 3.4% (from 76735.97 sec to 74226.13 sec).

The HPM statistics for the cumulus driver section for a 1-hour forecast are shown in Figure 3. As is to be expected, there is a larger load imbalance between

processors for the cumulus convection scheme than for the boundary layer. However, a consistent reduction in the number of operations and resulting CPU and wall-clock times (not shown) was observed at all processors. There are fewer instructions overall in the optimized code because of the improved data locality, but more floating point operations per cycle (resulting in higher flip rates, not shown). The percentage of floating point operations using the FMA unit is also increased (not shown).

An additional set of HPM instrumented runs were performed with the original and optimized cumulus parameterization package, using a different set of HPM statistics oriented at memory access performance of the code (event set 56, which includes counts of TLB misses, loads, stores, and L1 misses in addition to counts of cycles and instructions). Comparison of the output from the two sets of HPM runs for counts contained in both event sets showed that some statistics are quite reliable (consistent results were obtained from both runs), most notably Wall-clock time, User time, Processor Cycles, and Instructions completed. Others, particularly derived quantities such as utilization rate and MIPS, showed less run-to-run consistency. Analysis of the output for the cumulus driver section showed improvements in a number of memory related metrics, particularly in the number of L1 D cache load and store references, and the total number of load and store operations (not shown).

5 SUMMARY

We examined the WRFSI and the WRF model in order to identify code changes that could result in greater computational efficiency. Within WRFSI, optimization of the candidate modules in the hinterp and vinterp programs resulted in substantial speedups of those routines (20% for the hinterp program, and a 12-fold increase for the vinterp program). The resulting overall speedups for the program CPU time are on the order of 5-10%. For the hinterp program, a similar speedup is also obtained for the wall-clock time, whereas the vinterp program wall-clock time speedups were smaller. Results from a Hardware Performance Monitor run suggests that further wall-clock time speedups of the vinterp program would require reexamination of the generation of the netCDF files.

For the WRF model, the largest speedup was obtained by a reordering of the arrays inside the Grell-Devenyi cumulus parameterization package: the optimized cumulus parameterization (module_cu_gd) resulted in a 24% speedup, and a 3.4% overall decrease in the CPU time of the entire run. Optimization of the boundary layer routine ysu2d resulted in smaller speedups: a 7% reduction of the CPU for the boundary layer package, and a 2% reduction of the total CPU time.

Acknowledgements: This work was supported by the Air Force Weather Agency under contract FA8720-04-C-0009. Computing time at the NAVO HPC facility was provided by a grant from the DoD High Performance Computing Modernization Office.

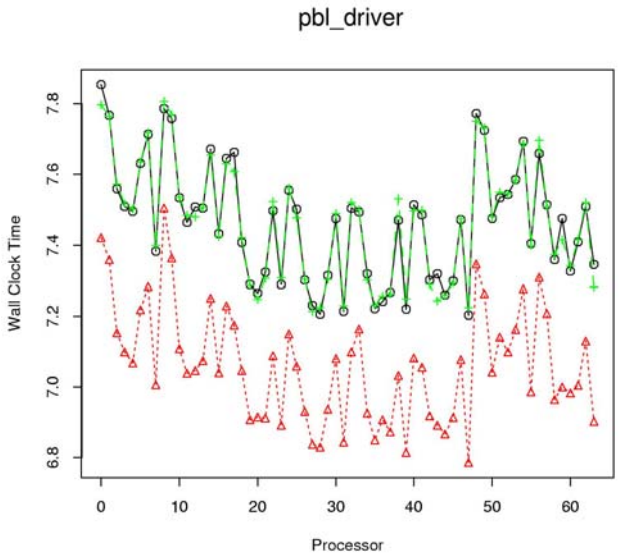

Figure 2. Wall clock time (sec) per processor for the PBL driver section for a 1-hour forecast. Results are shown for the original code (black octagons), code with optimized PBL package (red triangles), and optimized cumulus parameterization package (green crosses)

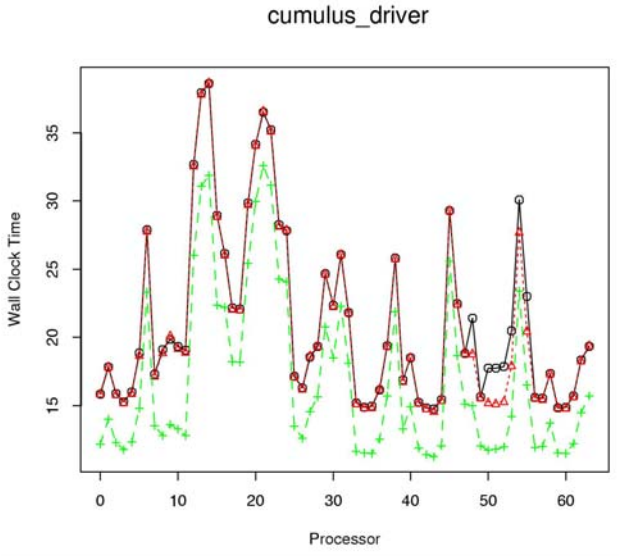

Figure 3. As in Figure 2, but for the cumulus driver section.