# Results of the AFWA MM5 Optimization Project

***Carlie J. Coats, Jr.***[1]
***coats@baronams.com***

## *Optimization and Contemporary Microprocessors: Concepts*

There are a number of general characteristics that are shared by most contemporary microprocessors; therefore, the corresponding optimizations will be broadly applicable to microprocessor based (parallel) computer systems in general. Among these general characteristics, and their consequences, are the following:

### *Improved Algorithms!*

This is where the biggest wins (factor-of-two, factor-of-ten, sometimes even more...) usually come from. Unfortunately, there are probably only a few opportunities for major algorithmic improvements in MM5 (but see the **`EXMOISR`** example, below!)**.**

### *Exploit Fine-Grained Parallelism*

Contemporary processors attempt to exploit fine-grained (instruction-level) parallelism that exists in programs in a number of ways: **pipelined**, so that different stages of multiple instructions are executing at the same time; **superscalar**, with multiple execution units that can work at the same time; and do **out-of-order** execution, based upon dependency analysis determined at run-time. These may allow a processor to have "over 200 instructions in flight at any given time" for the POWER4, to quote an IBM whitepaper [IBM], although 30-60 instructions is perhaps more common. The challenge is to have enough independent work available at any given time to keep the processor relatively busy. Note that data dependencies (where one calculation depends upon the result of another) may lead o less-than-peak processor performance, when the processor stalls waiting for results upon which computations depend. The ability of the processor to exploit fine-grained parallelism is aided by coding in terms of large basic blocks (e.g., innernmost-loop bodies), that contain much independent work to be performed. We will see examples of how this may be done in the section on **`EXMOISR`**, below.

1 coats@baronams.com
Baron Advanced Meteorological Systems, LLC
Research Triangle Park, NC

### *Loop and Array Reorganization, and Cache-Blocking*

Main computer memory systems are hundreds of times slower than the processors they serve. Computer system designers combat this problem by using **caches**, small sets of expensive, very-high-speed memory connected to the processor, to store frequently used data. There may be several levels of these caches, with level-1 very small but (being on-chip) very fast, level-2 probably off-chip, larger, and slower, etc. The tables that define virtual memory are implemented in a similar fashion with on-chip **translation-lookaside buffers (TLB's)**. Programs that access arrays out of the natural storage order (left-most subscript is "fastest" for Fortran) or that sweep through large arrays, using each value just once, harm memory system performance.

Within MM5, there are a number of opportunities to re-structure the model to use smaller (e.g., I-only instead of I-J-subscripted) arrays, particularly for scratch variables, or to permit repeated use of values already in cache and TLBs. We will see examples of how this may be done in the sections on **`EXMOISR,`** **`MRFPBL`** and dynamics unification, below. In some cases, we have seen that the reduction in memory footprint in one routine causes improved performance in its neighbors, because they are able to use cache more effectively. Needless to say, this effect is very case- and platform-dependent.

### *Avoid Expensive Operations*

Not all computations are equal; some are more expensive than others. Frequently one can use rules of algebra, laws of exponents, and a bit of program restructuring (such as saving andre-using values that are expensive to compute) to reduce computational cost. We will see examples of how this may be done in the sections on **`EXMOISR`** and **`MRFPBL`**, below. Some examples of expensive operations are:

**Divides** and **square roots** are much more expensive than adds or multiplies. Note that some MM5 codes multiply by **`RPSTAR`** instead of dividing by **`PSA`** for this reason. *Rationalize fractions where possible.*

**Log, exponential,** and **trigonometric functions** are very expensive. *Use laws of exponents.*

**Real-exponent powers** are very expensive, much more expensive than integer-exponent powers or square roots (where equivalent). *Or use laws of exponents.*

## *Optimization Specifics*

Here we describe in some detail work done on **EXMOISR** and **MRFPBL**, and unified dynamics. Similar optimizations have also been done to **HADV, VADV, LWRAD, SFLX, SOLVE,** and **SOUND**, and are currently (May 2004) in progress for the Grell cumulus parameterization, **CUPARA3**.

## *EXMOISR*

**Algorithm Improvement:** The original **EXMOISR** does full mixed-phase thermodynamics and precipitation computations for the entire 3-D modeling grid. We have implemented a version that attempts to reduce this extra work in a physically consistent manner:

Begin by doing a K-I-loop nest from the top of the model downward, comparing total (**QTOT=QV+QC+QR+QI+QS**) water mixing ratio with the saturation values **QVS** and **QVSI** of water vapor mixing ratio.

As long as **QTOT** stays below a threshold fraction of saturation (currently 0.5), assume that all condensed water should evaporate, and adjust the T and Q tendencies accordingly.

Save the highest level at which this fails in a new array as **CLDMAX(I)**. Note that **CLDMAX(I)** may be **KX+1,** indicating that there is no cloud in the vertical column at I.

For the rest of **EXMOISR**, perform the calculation only if **K** is less than **CLDMAX(I).**

For the final MM5v3.6 implementation, we set the threshold to be a scale-independent conservative value **QCRIT=0.5** (i.e., 50%), after sensitivity studies showed that this value showed negligible differences from the base case; the differences were quite small up through **QCRIT=0.7** and were noticeable for **QCRIT=0.8**. It may be worth re-visiting how best to set this value in the future; that is, however, a subject for specialists in microphysics rather than high performance computing.

**Array, Loop and Logic Structure:** If we extract just the loop structure (**DO** and **CONTINUE**) from **EXMOISR**, we have the following structure (for the thermodynamics and precipitation), with 11 loop nests having a mean length of 51 lines each:. 45 **I,K**-subscripted arrays are used to communicate intermediate results from nest to nest.

In the new code, there is a **K**-loop calculating **SIGMA**-fractions, an **I-K** nest calculating cloud top and evaporation thermodyamics, a complex **I-K** nest for full thermodynamics, a complex **K-N-I** nest computing precipitation, and an **I-K** "cleanup" loop", reducing the number of **I,K**–subscripted arrays to 5 and the number of I-subscripted arrays to 36 (greatly reducing the memory footprint):.

At the same time, **IF** statements were refactored to eliminate redundant "if freezing" conditions, etc. Also, the memory traffic caused by unnecessary initializations in which arrays were unnecessarily set to zero at the start of the routine has been removed; these variables are explicitly set to their real values elsewhere. This initialization practice is particularly pernicious, since it obviates the use of "trap on uninitialized variable" compiler flags during testing.

Because it preserves the I-innermost looping, the new structure offers improved performance on both microprocessor and vector platforms. An alternative loop structure with a single outermost I-loop, enclosing K-loops that do the cloud-top detection and evaporation thermodynamics, the full thermodynamics, and the precipitation was explored. As theory suggests, the alternative was somewhat faster on IBM *POWER* systems, but has poor performance on vector and (surprisingly) SGI *Rxxx* systems. Similarly, in the new structure above, the I-loops can be jammed to improve IBM *POWER* performance, at the cost of slowing down vector and SGI systems.

**Expensive Operation Removal:** Fractions were rationalized where appropriate, substantially reducing the number of expensive divisions. Laws of exponents were used to replace the particularly-expensive expression (inside the 3-D thermodynamics nest)

```
 SONV(I,K)=(CONST1A*(DRAIN/(RHO(I,K)&
*SNOW(I,K)*CONST1B)*(PI* &
DSNOW/(RHO(I,K)*SNOW(I,K)))**(BS/4))*
**0.94)**(4./(4.-0.94*BS))
```

by a once-and-for-all computation of

```
EX1=4./(4.-0.94*BS)
EX2=EX1*0.94*(1.0+0.25*BS)
  FSONV=(CONST1A*((DRAIN/CONST1B)*
(PI*DSNOW)**(0.25*BS))**0.94)**EX1
```

and the following much simpler computation in the 3-D thermodynamics nest

```
SONV(I,K)=FSONV*(RHO(I,K)*SNOW(I))**EX2
```

**Performance Improvement:** The performance improvement is highly case-dependent, but the new routine typically gives 1.5-3 times better computational performance than the original. This may, however, be obscured by load-balancing effects, since the model-as-a-whole is limited by the performance of the slowest (stormiest) parallel patch. For the shared-memory version of the model, one can partially alleviate this effect by use of **SCHEDULE(DYNAMIC)** directives; that alternative is not available in the distributed MM5-MPP. As I

understand it, the WRF driver should be capable of helping this problem.

## MRFPBL

**Array, Loop and Logic Structure:** in the original code, the calculation of the tendencies is artificially split into several steps, with much unnecessary memory traffic, according to the following pattern of calculation, split over four widely-separated **I-K** loop nests:

```
UTNP(I,K)=0.
UTEND=(A1(I,KK)-UX(I,K))*RDT
UTNP(I,K)=UTNP(I,K)+UTEND
UTNP(I,K)=PSB(I,J)*UTNP(I,K)
U3DTEN(I,J,K)=UTNP(I,K)
```

It is much more efficient to calculate the tendencies directly, and to eliminate the ***TNP** local arrays completely (where **PSRDT(I)=PSB(I,J)/DT**) in a single **I-K** loop nest containing:

```
UTEND=(A1(I,KK)-UX(I,K))*PSRDT(I)
U3DTEN(I,J,K) = UTEND
```

In the original **MRFPBL**, array **INTERIOR_MASK** is calculated (in scalar code!) by **SOLVE**, and used to control for which **I,J** values the cross point tendencies are updated, a pattern different from the rest of MM5, where variables **ICUT,JCUT** are used to construct loop bounds. In the original, the computational cost of this inefficient construct is relatively minor. However, in the optimized code (which executes more than twice as fast), it becomes a significant (~10%) portion of the computational cost for the routine. Due to the elimination of the **INTERIOR_MASK** argument, we have changed the name of the routine to **MRFPBL1,** in order to prevent linking of inconsistent executables.

**Expensive Operation Removal:** among others, one of the worst unnecessary-division examples in MM5 is found during the I-**K** loop nest that calculates diffusion coefficients for the free atmosphere, using ten divisions where properly-rationalized algebra would have only one! Generally, there were many opportunities to remove expensive divisions by rationalizing fractions, etc.

**NOTE:** one of the frustrating things about trying to optimize MRFPBL is thae there have been four generations of changes to it while this work was proceeding (particularly as a result of land-surface model changes... the present optimized version is already two generations out-of-date.

## Dynamics Unification

The flux, divergence-removal, Coriolis, curvature, and drag computation for state variables **X=U,V,W,T,PP,QV,QC,QR,QI,QNI,QG,QNC** is presently organized with five 3-D **(I-K-J)** loop nests per variable, most of which are hidden inside routines **HADV** and **VADV**. **HADV** uses **UA,VA** and uncoupled state variables, **VADV** uses **QDOT** and coupled state variables, and **SOLVE** uses divergence **DIVX**, to compute each variable's tendencies, in a structure almost guaranteed to put as much pressure on the memory system as possible. As an experiment in promoting memory system efficiency, we have re-coded this into three unified-advection routines, for **U,V,** (with one routine per **VADV** option) for **T,PP,W,** and (with one routine per **IMPHYS -IVQADV** option) for **Q**-variables. This structure not only promotes memory-system efficiency (repeatedly re-using arrays from cache), it also allows us to capture common subexpressions and reduce the total amount of computation performed. For **IMPHYS=6** the structure is:

```
DO J
DO K
  !  horiz flux & divergence terms
  DO I
     UE(I)=...
     UW(I)=...
     VN(I)=...
     VS(I)=...
     DD(I)=DIVX(I,J,K)+
        (UW(I)-UE(I))+(VS(I)-VN(I))
  END DO
  DO I
     QVT(I,J,K)=
       - UE(I)*QV3D(I  ,J+1,K)
       + UW(I)*QV3D(I  ,J-1,K)
       - VN(I)*QV3D(I+1,J  ,K)
       + VS(I)*QV3D(I-1,J  ,K)
       + DD(I)*QV3D(I  ,J  ,K)
  END DO
  DO I
     QCT(I,J,K)=...
  END DO
  ...
  DO I
     QNCT(I,J,K)=
  END DO
  IF(K.GT.1)THEN
  !!  drag terms
  DO I
     WT(I,J,K)=...
  END DO
  !!  vertical flux terms
  DO I
     QVT(I,J,K)=...
     ...
     QNCT(I,J,K)=...
  END DO
  END IF
END DO
END DO
```

This structure replaces 29 parallel loop nestss by just one (reducing parallel overhead), reuses arrays **UA,VA,QDOT,DIVX** seven times each, coupled variables **QVA,...,QNCA** and tendencies **QVT,...,QNCT** twice each (and completely

eliminates the extra reference to these tendencies for divergence removal). The performance improvement is case-, grid-, and platform-dependent, but is typically on the order of 20-50% of the computational cost of these routines.

A similar unification has been done for the **`DIFFU*`** and **`DCPL*`**, increasing moderately both computational and memory system efficiency.

Because the innermost-loop structure is basically unchanged (except, in some cases, for the factoring-out and re-use of common (vectorizable) subexpressions for multiple variables), this set of optimizations should improve performance for microprocessor based architectures, and should have beneficial or negligible effects upon vector architectures. There remains an issue with innermost-loop jamming. Ideally, this is an issue that should be resolved by the compiler system, which should "know" how best to optimize for its target hardware. Experience shows, however, that in many cases the expert programmer does a significantly better job than the compiler will; moreover, the default compiler flags in MM5's *configure.user* do not necessarily enable such loop transformations. In principle, one would in principle expect that the single I loop formulation would be best, since it offers the greatest instruction level parallelism for the compiler to use in keeping the processor core busy. However, there are at least two cases where this strategy fails:

The **Intel x86** architecture is too register-starved to take advantage of the additional instruction level parallelism, and performs best with the very simple vectorizable innermost loops.

**SGI IRIX/*Rxxxx*** should have sufficient resources to benefit from such very large **`I`** loops, but experiment shows that it performs best with innermost loops of intermediate size. Moreover, the loop-restrucuring facilities in the compiler do not perform as advertised, making manual loop-restructuring necessary.

## Acknowledgements and Disclaimers

This work was supported by the US Air Force Weather Agency under contract F19628-03-C-0089. Portions were also performed either on the author's own time and initiative, or supported by Baron Advanced Meteorological Systems, MCNC, or US EPA's STAR program grant number R825210 and R825211, prior to the start of that contract. Although the research described in this article has been funded wholly or in part by these agencies, it has not been subjected to their peer and policy review processes, and therefore does not necessarily reflect the views of any of the above; no official endorsement should be inferred.

The author thanks the Air Force Weather Agency, the US EPA, MCNC, and Baron Advanced Meteorological Systems for their support.

## References

IBM, *POWER4 System Microarchitecture*, October 2001. URL *http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html*