Section 2

MEMORY ORGANIZATION & CODE STRUCTURE

The model is written in standard FORTRAN, but there are some Cray-specific aspects that will be discussed in this section.

2.1 Pointers

MM5 makes use of Cray FORTRAN `POINTERs` to associate model variables with values for a given nest. For instance, three-dimensional array **UA**, appearing in the model physics and dynamics code, is used with a pointer to distinguish which part of the memory it is located in. It has one location for each nest.

`POINTERs` are addresses to identify an array or variable name (the pointee) with a part of the memory, e.g.,

```
POINTER (IAUA, UA(MIX, MJX, MKX))
```

at the top of a subroutine indicates that the location of 3-D array **UA** should be taken to start at address **IAUA** in the memory. Only variables with values that change on different nests require pointers. Physical and model constants and variables local to a subroutine do not have pointers.

Typically in the model the value of **IAUA** and the ~300 other pointers depends upon which nest is being calculated in that part of the program. There would be one value of **IAUA** for each nest and the array **UA** is dimensioned by the maximum **IX** and **JX** of any nest **(MIX**, **MJX**, **MKX)**.

The values of all ~300 (=**NUMVAR**) pointers for each domain are calculated in routine ADDALL at the beginning of a run or restart and stored in array **IAXALL (NUMVAR**, **MAXNES)** where **MAXNES** is the maximum number of nests during the simulation.

When calculations shift from one nest to another, routine ADDRX1C is called to reset the values of **IAUA** etc. for the next nest, taking the values from array **IAXALL**.

Sometimes information from two nests, a coarser and finer nest, are required concurrently such as in initializing nests or calculating nest inputs and feedbacks. In this case ADDRX1N is called to define the pointers for the finer mesh such as

```
    POINTER(INAUA, UAN(MIX, MJX, MKX))
```

so that **UA** represents the coarser mesh and **UAN** represents the finer mesh u-component of velocity.

The pointers are divided into 16 common blocks as follows (see section 4 in document for description of the corresponding pointees).

Three-dimensional prognostic variables

```
      COMMON/ADDR1/IAUA, IAUB, IAVA, IAVB, IATA, IATB, IAQVA,
     1        IAQVB, IAQCA, IAQCB, IAQRA, IAQRB, IAQIA, IAQIB,
     2        IAQNIA, IAQNIB
```

Two-dimensional model variables

```
      COMMON/ADDR2/IAPA, IAPB, IARC, IARN, IAF, IAMX, IAMD, IAHT,
     1        IALT, IALO, IALD, IAAL, IATH, IASH, IAZN, IATM,
     2        IAMA, IAEM, IAZO, IAHO, IAMO, IAPL, IARE, IAHX,
     3        IAQX, IAUT, IAPR, IATGA, IATGB, IASA, IAPSI, IARP,
     4        IAGS, IAGL, IACG, IASC, IAEF, IACR, IASR
```

Boundary variables

```
      COMMON/ADDR3/IAUE, IAUW, IAUET, IAUWT, IAVE, IAVW, IAVET, IAVWT,
     1        IATE, IATW, IATET, IATWT, IAQE, IAQW, IAQET, IAQWT,
     2        IAQCE, IAQCW, IAQCET, IAQCWT, IAQRIAQRW, IAQRET,
     3        IAQRWT IAQIE, IAQIW, IAQIET, IAQIWT, IAQNIAQNIW,
     4        IAQNIET,IAQNIWT,IAUN, IAUS, IAUNT, IAUST, IAVN, IAVS,
     5        IAVNT, IAVST, IATN, IATS, IATNT, IATST, IAQN, IAQS,
     6        IAQNT, IAQST, IAQCN, IAQCS, IAQCNT, IAQCST, IAQRN,
     7        IAQRS, IAQRNT, IAQRST, IAQIN, IAQIS, IAQINT, IAQIST,
     8        IAQNIN, IAQNIS, IAQNINT, IAQNIST, IAPE, IAPW, IAPET,
     9        IAPWT, IAPN, IAPS, IAPNT, IAPST, IAUJ1, IAUJL, IAUJ2,
     0        IAUJLX, IAVJ1, IAVJL, IAVJ2, IAVJLX, IAUI1, IAUIL,
     1        IAUI2, IAUILX, IAVI1, IAVIL, IAVI2, IAVILX
```

Other variables (scalars)

```
      COMMON/ADDR4/IAIL, IAJL, IAKL, IAKP1, IAKM, IAILX, IAILM,
     1        IAJLX, IAJLM, IAKTU, IAKTR, IAXT, IADT, IADT2, IADTM,
     2        IADX, IADX2, IADX4, IADX8, IAD16, IADXQ, IAXK, IAXKM,
     3        IAC0, IAC1, IAC3, IADMI, IADMA, IAQMI, IAQMA, IAQME,
     4        IAQMR, IAFNG, IAGNG
```

Nest scalars

```
      COMMON/ADDRNN/INN1, INN2, INN3, INN4, INN5, INN6, INN7, INN8,
     1        INN9, INN10, INN11, INN12, INN13, INN14, INN15, INN16,
     2        INN17, INN18, INN19, INN20, INN21, INN22, INN23, INN24,
     3        INN25, INN26, INN27
```

Split explicit arrays

```
      COMMON/ADDRSP/ISPM, ISPDTA, ISPAM, ISPAN, ISPDS,
     1        ISPHS, ISPZMX, ISPZMR, ISPBZ, ISPCZ, ISPHB
```

FDDA grid-nudging variables

```
      COMMON/ADDR5/IFUBO, IFVBO, IFUBT, IFVBT, IFTBO, IFTBT, IFQBO,
     1        IFQBT, IFVOR
      COMMON/ADDR6/IFPSO, IFPST, IFPSC, IFPSD, IFPSB, IFWDT, IFWCS,
     1        IFWQ, IFDMI, IFDMS, IFIPR, IFIPS, IFMPD, IFMPX
      COMMON/ADDR7/IFSFCO, IFBLWN, IFBLWT, IFBLWS, IFBLPC,
     1        IFBLPD, IFWXY, IFSFT, IFTIB, IFTIE, IFNTB, IFNTE, IFIQC,
     2        IFIDD, IFIDC, IFIDH
```

FDDA observation-nudging variables

```
      COMMON /ADDR8/ IFSVWT, IFVAR, IFRIO, IFRJO, IFRKO, IFTIM, IFERR
```

Three-dimensional nonhydrostatic arrays

```
      COMMON/NONHYD/INHWA, INHWB, INHWTS, INHPPA, INHPPB, INHUTS,
     1        INHVTS, INHPTS
```

Boundary nonhydrostatic arrays

```
      COMMON/NONHYDB/INHWE, INHWW, INHWET, INHWWT, INHPPE, INHPPW,
     1        1INHPPET, INHPPWT, INHWN, INHWS, INHWNT, INHWST,
     2        INHPPN, INHPPS, INHPPST, INHPPNT
```

Nonhydrostatic reference state arrays

```
      COMMON/NHCNS/INHT0, INHPS0
```

Navy PBL variables

```
      COMMON/ADDRV/IATURK, IATHSB, IATHQW, IAQWSB, IAUTV, IAVTV,
     1        IATTV, IAQVTV, IAQCTV
```

Atmospheric radiation array

```
      COMMON/RADIAT/IRTT
```

Upper radiative boundary condition

```
      COMMON/UPRAD/IUPR
```

This is the sequence of the pointers in **IAXALL** and also of the pointees in the memory where they are EQUIVALENCED to large arrays **ALLARR (IRHUGE**, **MAXNES)** and **INTALL (IIHUGE,MAXNES)**, where **IHUGE** is the sum dimension of all the pointees on a single domain. The arrays **ALLARR** and **INTALL** are written to the save files and read in for restarts, but are not used directly during the calculations. They are stored in /HUGE/ while **IAXALL** is stored in /ADDR0/.

If a user adds variables or arrays with pointers, several changes need to be made,

(i) Add the new common blocks, or add to the existing ones.
(ii) Change **NUMVAR** by the number of new pointers.
(iii) Change **IHUGE** by the total dimension of the new variables (and IIHUGE if new integer).
(iv) Change ADDALL to calculate the starting location of the new variables.
(v) Change ADDRX1C and ADDRX1N to equate the new pointers with the relevant part of **IAXALL**.

2.2 Multi-Tasking

On a multiprocessor computer such as a Cray Y-MP it is possible to perform certain tasks in parallel on different processors and MM5 is written to take advantage of this when available. This is done mostly for the outer **J** loops so that different north-south slices are done by different processors in parallel, i.e. at the same time. This saves wall-clock time but not CPU time. In fact if the tasks on different processors are not well-balanced it may cost more CPU time due to idle waiting time. Much of MM5's calculation is multi-tasked so that on average it uses about 7 of the 8 processors simultaneously.

Multi-tasking therefore differs from vectorization in that it applies to the outer loop while vectorization applies to the inner loop. MM5 is also vectorized as far as possible.

When multi-tasking, some variables have to be declared as `SHARED` or `PRIVATE`.

`SHARED` means that all processors should have access to the same part of the computer memory. If multi-tasking over a **J** loop, arrays that have a **J** in their dimension should normally be shared. Constants that are independent of **J** should also be shared.

`PRIVATE` means that each processor should have its own private copy of an array with its own memory location that is not available to other processors. Arrays that are dimensioned **(I**, **K)** or **(I)** or **(K)** should be private if they are storing information relevant to only one **J** slice. Arrays that store only local **J** slices should be private. Variables that change their values in the **J** loop should generally be private. Most variables are automatically private unless declared shared, but there are cases where private needs to be declared.

A typical multi-tasking directive in the code is as follows for **K**-loop 210:

```
CMIC$   DO ALL AUTOSCOPE NUMCHUNKS (8)
CMIC$1  PRIVATE (HELP1, HELP2, HSCR1N,HSCR2N, HELP1N, HELP2N)
CMIC$2  SHARED (ISOUTHO, JWESTO, IYY, JXX, IYYN, JXXN,
CMIC$2           ISOUTH, JWEST, IRA)
        DO 210 K=1, KZZ
```

where the 'NUMCHUNKS(8)' aids efficiency by assigning about 1/8 of the **K** indices to each processor. This works best on a dedicated eight-processor machine such as a Y-MP and with loops where the load is equally balanced among the **K** values, i.e. it is not spending significantly different times on each value of **K**. Loops that are unbalanced, such as those with conditional statements dependent on the loop index, do not use the 'NUMCHUNKS' directive.

In MM5, multi-tasking takes place in subroutines SOLVE1, SOLVE3 and SOUND. All of the physics routines are called from these routines inside multi-tasked **J** loops, so new physics routines developed by the user need not be multi-tasked themselves. However, care should be taken that shared variables are not changed in the new routines.

Other multi-tasked routines are BDYUV, BDYVAL, FEEDBK, DOTS, STOTNDI, STOTNDT, BDYOVL1, INITNEST and the split-explicit routines DIVG, SPCORT, SPCORU, SPDIFF, SPDIVG, SPGEOP, SPGRAD, SPLITF, SPSTEP2.

On the Cray Y-MP the `cf77` command should include a switch for multi-tasking to recognize the `CMIC$` directives, thus `cf77 -Zu` should be used.