WRF Software Architecture

John Michalakes, Head WRF Software Architecture
Michael Duda
Dave Gill
Outline

- Introduction
- Computing Overview
- WRF Software Overview
Introduction – WRF Software Characteristics

- Developed from scratch beginning around 1998, primarily Fortran and C
- Requirements emphasize flexibility over a range of platforms, applications, users, performance
- WRF develops rapidly. First released Dec 2000; current release WRF v3.3.1 (Sep 2011); next release WRF v3.4 (April 2012)
- Supported by flexible efficient architecture and implementation called the WRF Software Framework
Introduction - WRF Software Framework Overview

- Implementation of WRF Architecture
  - Hierarchical organization
  - Multiple dynamical cores
  - Plug compatible physics
  - Abstract interfaces (APIs) to external packages
  - Performance-portable
- Designed from beginning to be adaptable to today’s computing environment for NWP

http://box.mmm.ucar.edu/wrf/WG2/bench/
Computing Overview

APPLICATION

SYSTEM

HARDWARE

Patches
Tiles
WRF Comms

Processes
Threads
Messages

Processors
Nodes
Networks
Hardware: The Computer

- The ‘N’ in NWP
- Components
  - Processor
    - A program counter
    - Arithmetic unit(s)
    - Some scratch space (registers)
    - Circuitry to store/retrieve from memory device
    - Cache
  - Memory
  - Secondary storage
  - Peripherals
- The implementation has been continually refined, but the basic idea hasn’t changed much
Hardware has not changed much...

A computer in 1960

IBM 7090

6-way superscalar
36-bit floating point precision
~144 Kbytes

~50,000 flop/s
48hr 12km WRF CONUS in 600 years

A computer in 2008

IBM P6

Dual core, 4.7 GHz chip
64-bit floating point precision
1.9 MB L2, 36 MB L3
Upto 16 GB per processor

~5,000,000,000 flop/s
48 12km WRF CONUS in 52 Hours
...how we use it has

- Fundamentally, processors haven’t changed much since 1960
- Quantitatively, they haven’t improved nearly enough
  - 100,000x increase in peak speed
  - 100,000x increase in memory size
- We make up the difference with parallelism
  - Ganging multiple processors together to achieve $10^{11-12}$ flop/second
  - Aggregate available memories of $10^{11-12}$ bytes

$\sim 1,000,000,000,000$ flop/s $\sim 250$ procs
48-h,12-km WRF CONUS in under 15 minutes
If the machine consists of 4 nodes, each with 4 processors, how many different ways can you run a job to use all 16 processors?

- 4 MPI processes, each with 4 threads
  
  ```bash
  setenv OMP_NUM_THREADS 4
  mpirun -np 4 wrf.exe
  ```

- 8 MPI processes, each with 2 threads
  
  ```bash
  setenv OMP_NUM_THREADS 2
  mpirun -np 8 wrf.exe
  ```

- 16 MPI processes, each with 1 thread
  
  ```bash
  setenv OMP_NUM_THREADS 1
  mpirun -np 16 wrf.exe
  ```
Examples

- If the machine consists of 4 nodes, each with 4 processors, how many different ways can you run a job to use all 16 processors?

  - 4 MPI processes, each with 4 threads
    
    ```
    setenv OMP_NUM_THREADS 4
    mpirun -np 4 wrf.exe
    ```

  - 8 MPI processes, each with 2 threads
    
    ```
    setenv OMP_NUM_THREADS 2
    mpirun -np 8 wrf.exe
    ```

  - 16 MPI processes, each with 1 thread
    
    ```
    setenv OMP_NUM_THREADS 1
    mpirun -np 16 wrf.exe
    ```
Examples

- If the machine consists of 4 nodes, each with 4 processors, how many different ways can you run a job to use all 16 processors?

  - 4 MPI processes, each with 4 threads
    
    \texttt{setenv OMP\_NUM\_THREADS 4}
    \texttt{mpirun –np 4 wrf.exe}

  - 8 MPI processes, each with 2 threads
    
    \texttt{setenv OMP\_NUM\_THREADS 2}
    \texttt{mpirun –np 8 wrf.exe}

  - 16 MPI processes, each with 1 thread
    
    \texttt{setenv OMP\_NUM\_THREADS 1}
    \texttt{mpirun –np 16 wrf.exe}
Application: WRF

- WRF can be run **serially** or as a **parallel** job
- WRF uses *domain decomposition* to divide total amount of work over parallel processes
• The decomposition of the application over processes has two levels:
  – The domain is first broken up into rectangular pieces that are assigned to MPI (distributed memory) processes. These pieces are called patches.
  – The patches may be further subdivided into smaller rectangular pieces that are called tiles, and these are assigned to shared-memory threads within the process.
Parallelism in WRF: Multi-level Decomposition

Model domains are decomposed for parallelism on two-levels

**Patch**: section of model domain allocated to a distributed memory node, this is the scope of a mediation layer solver or physics driver.

**Tile**: section of a patch allocated to a shared-memory processor within a node; this is also the scope of a model layer subroutine.

Distributed memory parallelism is over patches; shared memory parallelism is over tiles within patches.

- Single version of code for efficient execution on:
  - Distributed-memory
  - Shared-memory (SMP)
  - Clusters of SMPs
  - Vector and microprocessors
## Distributed Memory Communications

### When Needed?
Communication is required between patches when a horizontal index is incremented or decremented on the right-hand-side of an assignment.

### Why?
On a patch boundary, the index may refer to a value that is on a different patch.

Following is an example code fragment that requires communication between patches.

### Signs in code
Note the tell-tale $+1$ and $-1$ expressions in indices for $rr$, $H1$, and $H2$ arrays on right-hand side of assignment.

These are **horizontal data dependencies** because the indexed operands may lie in the patch of a neighboring processor. That neighbor’s updates to that element of the array won’t be seen on this processor.
Distributed Memory Communications

(module_diffusion.F)

SUBROUTINE horizontal_diffusion_s (tendency, rr, var, . . .

. . .

    DO j = jts,jte
    DO k = kts,ktf
    DO i = its,ite
        mrdx=msft(i,j)*rdx
        mrdy=msft(i,j)*rdy
        tendency(i,k,j)=tendency(i,k,j)-
        (mrdx*0.5*((rr(i+1,k,j)+rr(i,k,j))*H1(i+1,k,j)-
                     (rr(i-1,k,j)+rr(i,k,j))*H1(i-1,k,j))+
        mrdy*0.5*((rr(i,k,j+1)+rr(i,k,j))*H2(i,k,j+1)-
                     (rr(i,k,j-1)+rr(i,k,j))*H2(i,k,j-1))-msft(i,j)*(H1avg(i,k+1,j)-H1avg(i,k,j)+
                     H2avg(i,k+1,j)-H2avg(i,k,j))/dzetaw(k)
        )
    
    ENDDO
    ENDDO
    ENDDO

. . .
Distributed Memory MPI Communications

- Halo updates

memory on one processor

memory on neighboring processor
Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers
Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers

Average Daily Total rainfall (mm) - March 1997
Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers

all y on patch

all z on patch

all x on patch
Distributed Memory (MPI) Communications

- Halo updates
- Periodic boundary updates
- Parallel transposes
- Nesting scatters/gathers
### Review – Computing Overview

<table>
<thead>
<tr>
<th>APPLICATION</th>
<th>SYSTEM</th>
<th>HARDWARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(WRF)</td>
<td>(UNIX, MPI, OpenMP)</td>
<td>(Processors, Memories, Wires)</td>
</tr>
</tbody>
</table>

#### Distributed Memory Parallel

- Domain contains Patches contain Tiles
- Job contains Processes contain Threads
- Cluster contains Nodes contain Processors
Outline

- Introduction
- Computing Overview
  - WRF Software Overview
WRF Software Overview

- Architecture
- Directory structure
- Model Layer Interface
- Data Structures
- I/O
Hierarchical software architecture

- Insulate scientists' code from parallelism and other architecture/implementation-specific details
- Well-defined interfaces between layers, and external packages for communications, I/O, and model coupling facilitates code reuse and exploiting of community infrastructure, e.g. ESMF.
WRF Software Architecture

• **Driver Layer**
  - **Domains**: Allocates, stores, decomposes, represents abstractly as single data objects
  - **Time loop**: top level, algorithms for integration over nest hierarchy
Mediation Layer

- Solve routine, takes a domain object and advances it one time step
- Nest forcing, interpolation, and feedback routines
WRF Software Architecture

- **Mediation Layer**
  - The sequence of calls for doing a time-step for one domain is known in Solve routine
  - Dereferences fields in calls to physics drivers and dynamics code
  - Calls to message-passing are contained here as part of Solve routine
• **Model Layer**
  
  - **Physics and Dynamics**: contains the actual WRF model routines are written to **perform some computation** over an arbitrarily sized/shaped, 3d, rectangular subdomain
Call Structure Superimposed on Architecture

module_microphysics_driver (phys)
WRF Software Overview

- Architecture
- Directory structure
- **Model Layer Interface**
- Data Structures
- I/O
All state **arrays** passed through argument list as simple (not derived) data types

Domain, memory, and run dimensions passed unambiguously in **three dimensions**

Model layer routines are called from mediation layer (physics drivers) in **loops over tiles**, which are multi-threaded
Restrictions on Model Layer subroutines:

- No I/O, communication
- No stops or aborts
- Use `wrf_error_fatal`
- No common/module storage of decomposed data
- Spatial scope of a Model Layer call is one “tile”
SUBROUTINE driver_for_some_physics_suite (   
  . . .
  !$OMP DO PARALLEL
    DO ij = 1, numtiles
      its = i_start(ij) ; ite = i_end(ij)
      jts = j_start(ij) ; jte = j_end(ij)
      CALL model_subroutine( arg1, arg2, . . .
          ids , ide , jds , jde , kds , kde ,
          ims , ime , jms , jme , kms , kme ,
          its , ite , jts , jte , kts , kte )
    END DO
  . . .
END SUBROUTINE
WRF Model Layer Interface

**template for model layer subroutine**

```fortran
SUBROUTINE model_subroutine ( &
    arg1, arg2, arg3, ..., argn, &
    ids, ide, jds, jde, kds, kde, & ! Domain dims
    im, ime, jms, jme, kms, kme, & ! Memory dims
    its, ite, jts, jte, kts, kte ) ! Tile dims

IMPLICIT NONE

! Define Arguments (State and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, ...
REAL, DIMENSION (ims:ime,jms:jme) :: arg7, ...

...!
! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, ...
..."
```
template for model layer subroutine

...  
! Executable code; loops run over tile 
! dimensions 
DO j = jts, MIN(jte,jde-1) 
  DO k = kts, kte 
    DO i = its, MIN(ite,ide-1) 
      loc1(i,k,j) = arg1(i,k,j) + ... 
    END DO 
  END DO 
END DO 
END DO 
END DO
template for model layer subroutine

SUBROUTINE model ( &
arg1, arg2, arg3, ..., argn, &
ids, ide, jds, jde, kds, kde, & Domain dims
ims, ime, jms, jme, kms, kme, & Memory dims
its, ite, jts, jte, kts, kte ) ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, ...
REAL, DIMENSION (ims:ime,jms:jme) :: arg7, ...

! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, ...

! Executable code; loops run over tile dimensions
DO j = MAX(jts,jds), MIN(jte,jde-1)
  DO k = kts, kte
    DO i = MAX(its,ids), MIN(ite,ide-1)
      loc1(i,k,j) = arg1(i,k,j) + ...
      END DO
    END DO
  END DO
END DO

• Domain dimensions
  • Size of logical domain
  • Used for bdy tests, etc.
template for model layer subroutine

SUBROUTINE model ( &
    arg1, arg2, arg3, ... , argn, &
    ids, ide, jds, jde, kds, kde, & ! Domain dims
    ims, ime, jms, jme, kms, kme, & ! Memory dims
    its, ite, jts, jte, kts, kte ) ! Tile dims
IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, ...
REAL, DIMENSION (ims:ime,jms:jme) :: arg7, ...

! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, ...

! Executable code; loops run over tile dimensions
DO j = MAX(jts,jds), MIN(jte,jde-1)
    DO k = kts, kte
        DO i = MAX(its,ids), MIN(ite,ide-1)
            loc1(i,k,j) = arg1(i,k,j) + ...
        END DO
    END DO
END DO
END SUBROUTINE model

- Domain dimensions
  - Size of logical domain
  - Used for bdy tests, etc.
- Memory dimensions
  - Used to dimension dummy arguments
  - Do not use for local arrays
template for model layer subroutine

SUBROUTINE model ( &
    arg1, arg2, arg3, ..., argn, &
    ids, ide, jds, jde, kds, kde, & ! Domain dims
    ims, ime, jms, jme, kms, kme, & ! Memory dims
    its, ite, jts, jte, kts, kte ) ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, ...
REAL, DIMENSION (ims:ime,jms:jme) :: arg7, ...

! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, ...

! Executable code; loops run over tile dimensions
DO j = MAX(jts,jds), MIN(jte,jde-1)
    DO k = kts, kte
        DO i = MAX(its,ids), MIN(ite,ide-1)
            loc1(i,k,j) = arg1(i,k,j) + ...
        END DO
    END DO
END DO

• Domain dimensions
  • Size of logical domain
  • Used for bdy tests, etc.
• Memory dimensions
  • Used to dimension dummy arguments
  • Do not use for local arrays
• Tile dimensions
  • Local loop ranges
  • Local array dimensions
template for model layer subroutine

SUBROUTINE model ( &
    arg1, arg2, arg3, ... , argn, &
    ids, ide, jds, jde, kds, kde, & ! Domain dims
    ims, ime, jms, jme, kms, kme, & ! Memory dims
    its, ite, jts, jte, kts, kte ) ! Tile dims

IMPLICIT NONE

! Define Arguments (S and I1) data
REAL, DIMENSION (ims:ime,kms:kme,jms:jme) :: arg1, . . .
REAL, DIMENSION (ims:ime,jms:jme) :: arg7, . . .
. . .

! Define Local Data (I2)
REAL, DIMENSION (its:ite,kts:kte,jts:jte) :: loc1, . . .
. . .

! Executable code; loops run over tile dimensions
DO j = MAX(jt,jds), MIN(jte,jde-1)
    DO k = kts, kte
        DO i = MAX(its,ids), MIN(ite,ide-1)
            loc1(i,k,j) = arg1(i,k,j) + ...
        END DO
    END DO
END DO
END DO
END DO

! Domain dimensions
- Size of logical domain
- Used for bdy tests, etc.

! Memory dimensions
- Used to dimension dummy arguments
- Do not use for local arrays

! Tile dimensions
- Local loop ranges
- Local array dimensions

- Patch dimensions
  - Start and end indices of local distributed memory subdomain
  - Available from mediation layer (solve) and driver layer; not usually needed or used at model layer
WRF Software Overview

- Architecture
- Directory structure
- Model Layer Interface
- Data Structures
  - I/O
WRF I/O

• Streams: pathways into and out of model
  – History + auxiliary output streams (10 and 11 are reserved for nudging)
  – Input + auxiliary input streams (10 and 11 are reserved for nudging)
  – Restart, boundary, and a special Var stream
WRF I/O

- Attributes of streams
  - Variable set
    - The set of WRF state variables that comprise one read or write on a stream
    - Defined for a stream at compile time in Registry
  - Format
    - The format of the data outside the program (e.g. NetCDF), split
    - Specified for a stream at run time in the namelist
WRF I/O

• Attributes of streams
  – Additional namelist-controlled attributes of streams
    • Dataset name
    • Time interval between I/O operations on stream
    • Starting, ending times for I/O (specified as intervals from start of run)
Outline - Review

• Introduction
  – WRF started 1998, clean slate, Fortran + C
  – Targeted for research and operations

• WRF Software Overview
  – Hierarchical software layers
  – Patches (MPI) and Tiles (OpenMP)
  – Strict interfaces between layers
  – Contract with developers
  – I/O