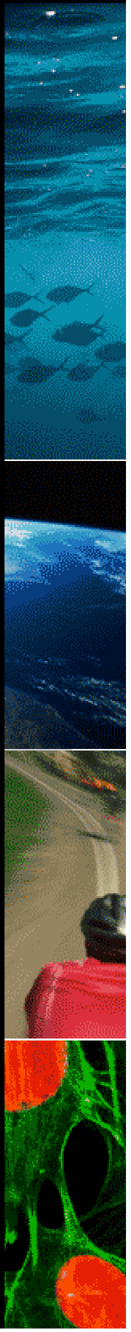
The Supercomputer Company

# Performance of WRF using UPC

CUG 2006

**Kim, Hee-Sik**

**Cray Korea Inc.**



## Overview

- Introduction of the Weather Research and Forecast (WRF) model
- Cray X1E system
- Communication Layer: RSL_LITE
  - The Role of RSL_LITE
  - UPC Version of RSL_LITE
  - Some Optimization in RSL_LITE
- Case study: CONUS Benchmark Results
- Conclusion

## Weather Research and Forecast (WRF) Model

- Seven-year, multi-agency collaboration to develop advanced community mesoscale model and data assimilation system with direct path to operations

- Current release WRFV2.1.2, Jun 2006
  - Two dynamical cores, numerous physics, chemistry
  - Variational Data Assimilation (released) and Ensemble Kalman Filter (in development)
  - Rapid community growth
    - More than 3,000 registered users
    - June 2005 Users Workshop: 219 participants, 117 inst., 65 countries
    - 46 scientific papers: real-time NWP, atmos. chemistry, data assimilation, climate, wildfires, mesoscale processes

- Operational capabilities implemented or planned
  - Air Force Weather Agency
  - National Centers for Environmental Prediction
  - KMA (Korea), IMD (India), CWB (Taiwan), IAF (Israel), WSI (U.S.)

* John Michalakes, 2005

# Cray X1E system

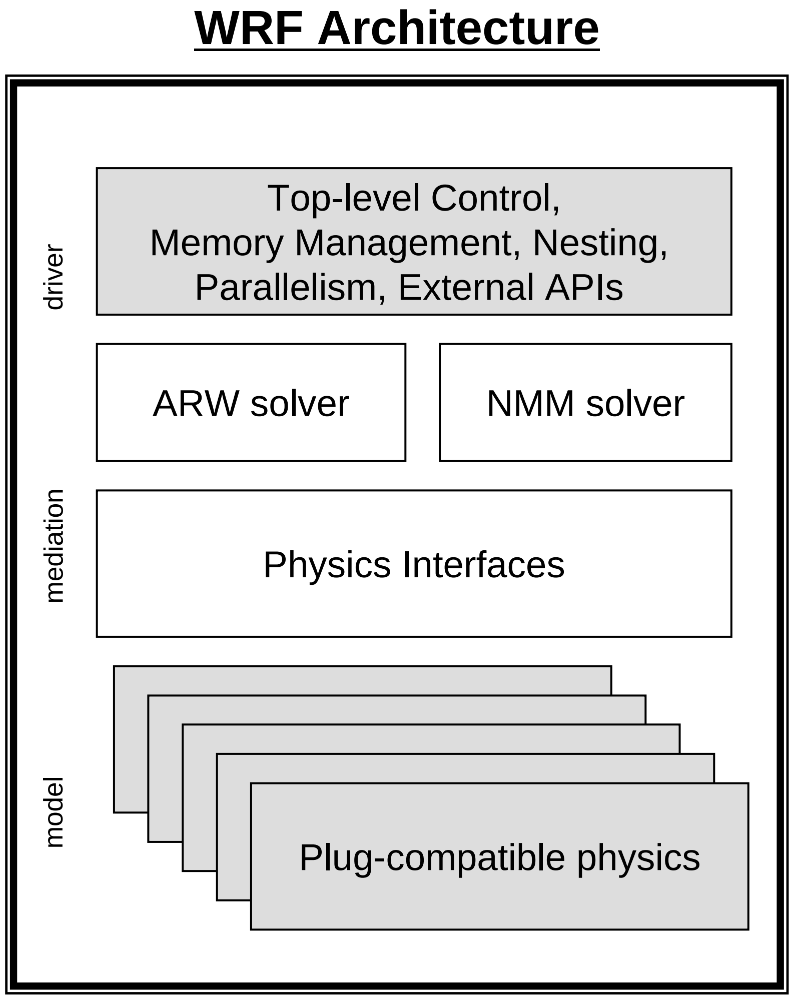
- We used KMA’s Cray X1E system

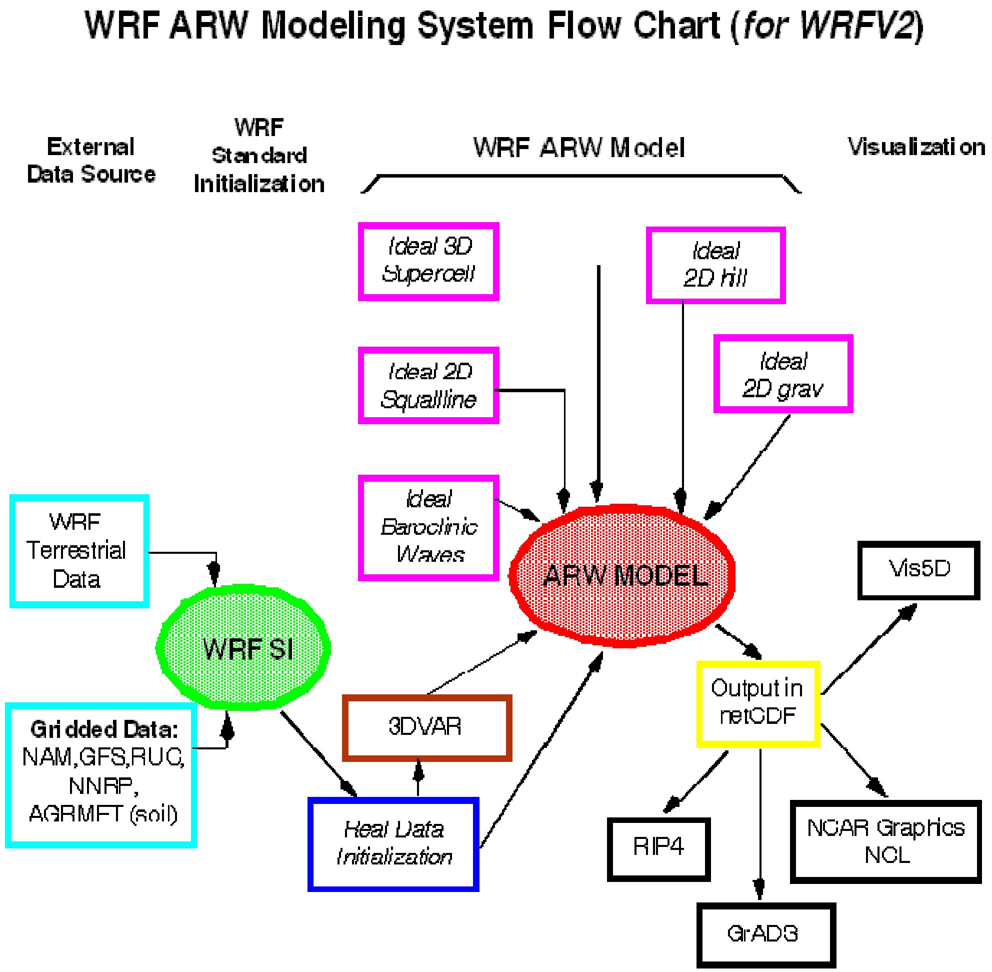
CRAY X1E

ADIC Scalar 10K 1PB

| Ref | CRAY X1E |
|---|---|
| Installation | 2005. 12 |
| CPU | 1024 MSP |
| Peak performance | 18.5 TF |
| Main Memory | 4 TB |
| Capacity of DAS Disk | 67 TB |
| Capacity of SAN Disk | 21 TB |
| Capacity of Tape drive | 1 PB |
| Functions | For NWP Operation Research & Development |

# WRF Description



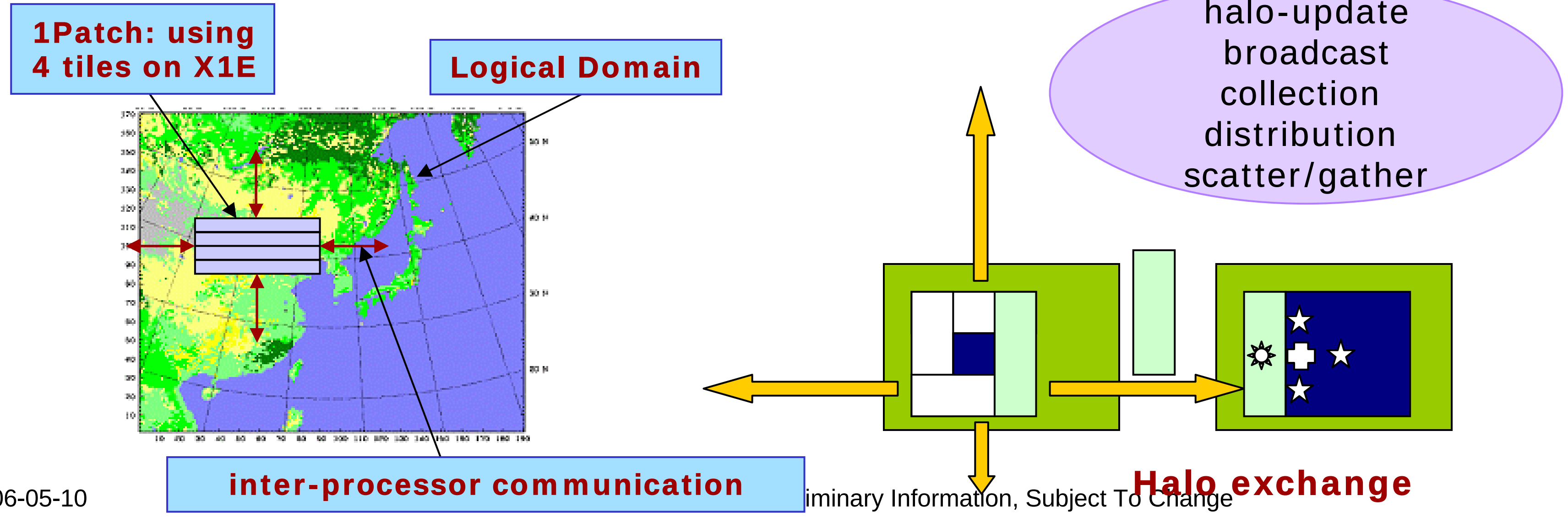


* WRF official site: www.mmm.ucar.edu

# Parallelism in WRF

- WRF uses domain decomposition to divide total amount of work over parallel processes on two levels
  - Patch: allocated to a distributed memory node
  - Tile: allocated to a shared-memory processor

- APIs for communication: Runtime system Library (RSL)
- The light version of RSL: RSL_LITE



## The Role of RSL_LITE

- The Sequence of Halo Exchange

| | |
|---|---|
| • RSL_LITE_INIT_EXCH | (Memory allocation for Y halo exchange) |
| • RSL_LITE_PACK | (Local copy of halo region in Y direction) |
| • RSL_LITE_EXCH_Y | (Halo exchange in Y direction) |
| • RSL_LITE_PACK | (Copy to Y halo region of received data) |
| • RSL_LITE_INIT_EXCH | (Memory allocation for X halo exchange) |
| • RSL_LITE_PACK | (Local copy of halo region in X direction) |
| • RSL_LITE_EXCH_X | (Halo exchange in X direction) |
| • RSL_LITE_PACK | (Copy to X halo region of received data) |

## The Role of RSL_LITE

- In RSL_LITE_INIT_EXCH
  - Each MPI process has send/receive buffers which will have data which should be exchanged between X and Y directions. The buffer will have ghost area values between domains which will be determined from the number of processes and X/Y domain decomposition.
  - This function will allocate memory which is needed for storing ghost area values before halo exchange. The size of memory is dependent on the number of variables and its ghost area size. If there is sufficient memory already allocated, no action will be occurred. If more memory is required, new memory allocation will be occurred.

## The Role of RSL_LITE

- In RSL_LITE_PACK
  - If there is a data which should be sent, the ghost area data will be copied to local memories which was allocated by RSL_LITE_INIT_EXCH routine. The data will be packed into one contiguous memory region so that one MPI send/receive call could be used for exchanging halo data.
  - If there is a data which was received, the data will be copied to the original ghost area position depending on the domain decomposition.

## The Role RSL_LITE

- In RSL_LITE_EXCH_X(Y)
  - The packed ghost area data will be exchanged between MPI process. The number of MPI send/receive calls per MPI process is dependent on the domain decomposition.
  - Non-blocking MPI_ISEND/MPI_IRECV call with MPI_WAIT sequence is used for this communication. This will require only weak synchronization between MPI processes which do halo exchange.

# UPC Version of RSL_LITE

- What is UPC?
  - UPC is based on the distributed shared memory mechanism.
  - There are private and shared memory space in UPC memory model. Shared memory space can be accessed by any THEADS in execution but has affinity to a specific THREAD.
  - The benefit of UPC will be
    - The one-side communication
    - Performance improvement if H/W architecture support distribute shared memory or fast one-side communication mechnism
    - Source code brevity and clarity compared with MPI
    - …

# UPC Version of RSL_LITE

- Receive buffer is unchanged.
- Send buffer is replaced by shared memory which has affinity to the local node and is allocated by "upc_alloc" and freed by "upc_local_free" UPC routines.
- Shared pointer to shared memory with local affinity trick will be used.
  - typedef shared [] char *sh_char_ptr;
  - shared [RSL_MAXPROC] sh_char_ptr upc_buf[THREADS][RSL_MAXPROC];
- Shared memory which has affinity to local node can be treated as local memory so normal memory copy and pointer operation can be used. So there is no change in RSL_PACK function.

## UPC Version of RSL_LITE

- MPI_ISEND/MPI_IRECV/MPI_WAIT sequence is replaced by upc_memget on one-side. So one-side communication is used.
- The lack of partial synchronization between UPC threads requires global synchronization, upc_barrier. This could be the bottleneck for large number of processes.

# UPC Version of RSL_LITE

- Example: RSL_LITE/c_code.c

```
...
#ifdef __UPC__
/* This will prevent performance improvement */
 upc_barrier;

 if (np_y > 1)
 {
  MPI_Cart_shift( *comm0, 0, 1, &ym, &yp ) ;
  if ( yp != MPI_PROC_NULL )
   upc_memget(buffer_for_proc(yp, yp_curs, RSL_RECVBUF),(shared [] char *)upc_buf[yp][MYTHREAD],yp_curs);
  if ( ym != MPI_PROC_NULL )
   upc_memget(buffer_for_proc(ym, ym_curs, RSL_RECVBUF),(shared [] char *)upc_buf[ym][MYTHREAD],ym_curs);
 }
```

```
#else
 if ( np_y > 1 ) {
  MPI_Cart_shift( *comm0, 0, 1, &ym, &yp ) ;
  if ( yp != MPI_PROC_NULL ) {
   MPI_Irecv ( buffer_for_proc( yp, yp_curs, RSL_RECVBUF ), yp_curs, MPI_CHAR, yp, me, comm, &yp_recv ) ;
  }
  if ( ym != MPI_PROC_NULL ) {
   MPI_Irecv ( buffer_for_proc( ym, ym_curs, RSL_RECVBUF ), ym_curs, MPI_CHAR, ym, me, comm, &ym_recv ) ;
  }
  if ( yp != MPI_PROC_NULL ) {
   MPI_Isend ( buffer_for_proc( yp, 0,      RSL_SENDBUF ), yp_curs, MPI_CHAR, yp, yp, comm, &yp_send ) ;
  }
  if ( ym != MPI_PROC_NULL ) {
   MPI_Isend ( buffer_for_proc( ym, 0,      RSL_SENDBUF ), ym_curs, MPI_CHAR, ym, ym, comm, &ym_send ) ;
  }
  if ( yp != MPI_PROC_NULL ) MPI_Wait( &yp_recv, &stat ) ;
  if ( ym != MPI_PROC_NULL ) MPI_Wait( &ym_recv, &stat ) ;
  if ( yp != MPI_PROC_NULL ) MPI_Wait( &yp_send, &stat ) ;
  if ( ym != MPI_PROC_NULL ) MPI_Wait( &ym_send, &stat ) ;
 }
#endif
```

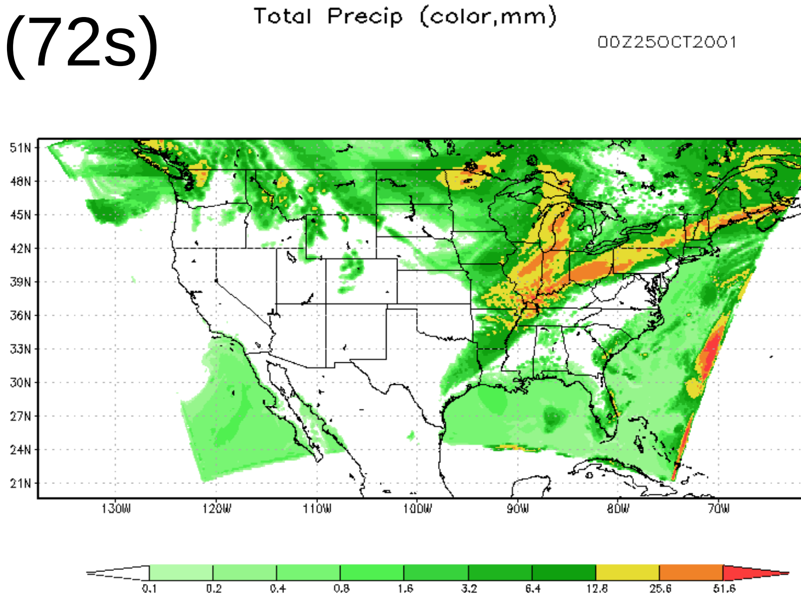
## Some Optimization technique in RSL_LITE

- In WRF version 2.1.2, RSL_LITE introduced f_pack.F which is somewhat similar vicopy.F in RSL. This will enable vectorization on many platforms in copying between halo area data and packed data for exchanging.

- What was modified?
  - Added 2-D f_pack*/f_unpack* subroutine for future utilization
  - Added additional rank for packed variables to better support vectorization and to aid compiler optimization
  - Added X1E specific directives to improve streaming and vectorization efficiency

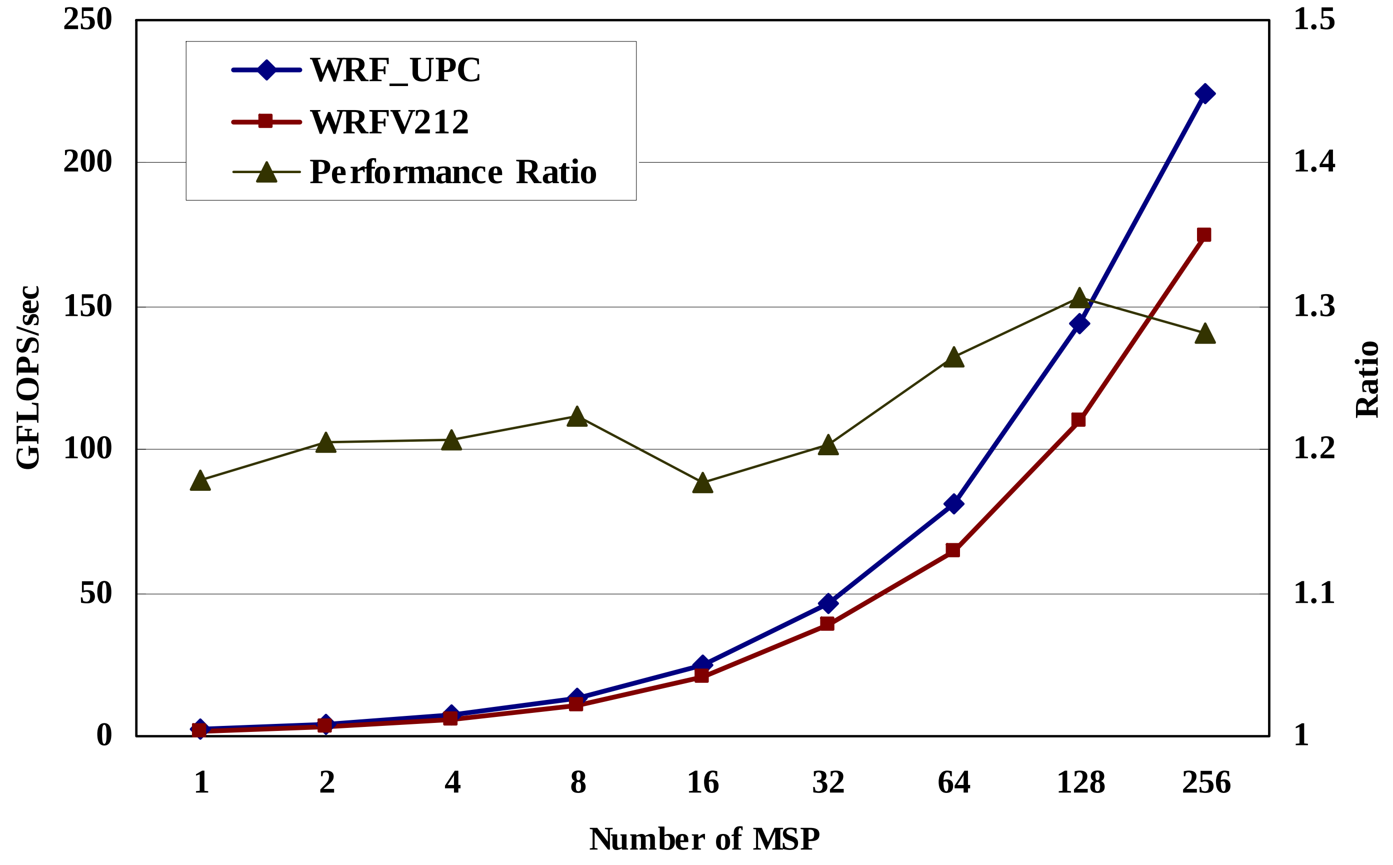
# Benchmark: CONUS

- Used standard CONUS WRF benchmark
  - CONUS means Continental US.
  - Domain size: 425x300x35
  - DX=12KM, DT=72s
  - 3 hour forecast with Eulerian Mass(EM) dynamics
  - 29458 Mflop per time step(72s)
  - Only computational part included(no initialization, file I/O)



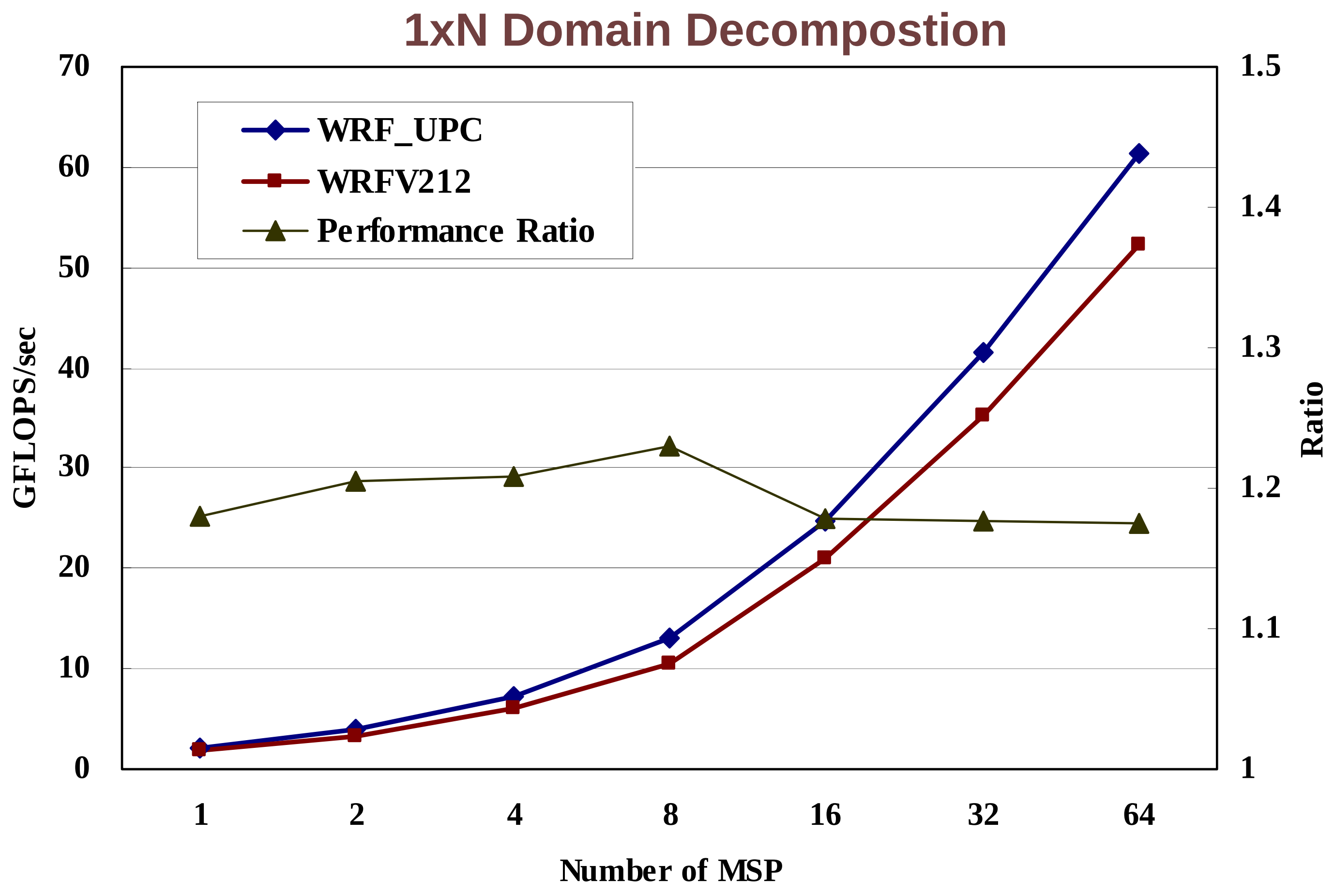
# WRF CONUS Benchmark

WRF 2.1.2 EM Core CONUS Benchmark, 425x300x35, DX=12KM, DT=72s



# WRF CONUS Benchmark

WRF 2.1.2 EM Core CONUS Benchmark, 425x300x35, DX=12KM, DT=72s



## Summary

- We compared a UPC version of RSL_LITE, the communication layer of WRF, with original version.
  - CONUS benchmark data was used.
  - We showed some improvements in model runtime.
  - The performance ration is between 1.2 ~ 1.3.
- There is possibility of further optimization
  - Global synchronization/ Partial synchronization
- Additionally, we showed some optimization technique in RSL_LITE.

The Supercomputer Company

# Performance of WRF using UPC

## Questions? / Comments!

CUG 2006

Thank You!

This Presentation May Contain Some Preliminary Information, Subject To Change