国家超级计算无锡中心
National Supercomputing Center in Wuxi

Sunway TaihuLight: Designing and Tuning Scientific Applications at the Scale of 10-Million Cores

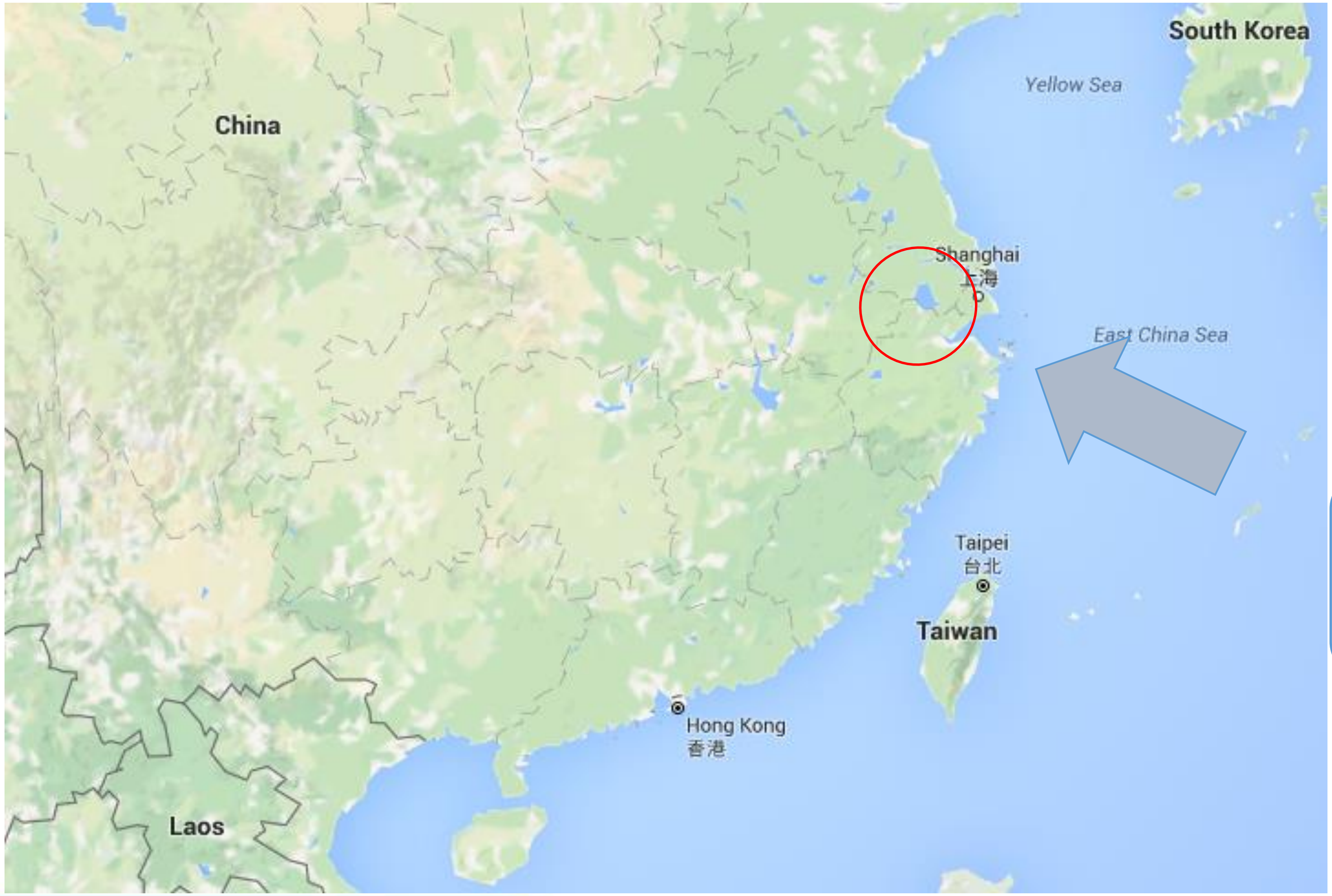
Haohuan Fu
National Supercomputing Center in Wuxi
Department of Earth System Science, Tsinghua University

March 14th 2017 @ SCF

Outline

- The Sunway Machine
- The Programming Model
- Scaling Over 10-Million Cores: A Climate Example

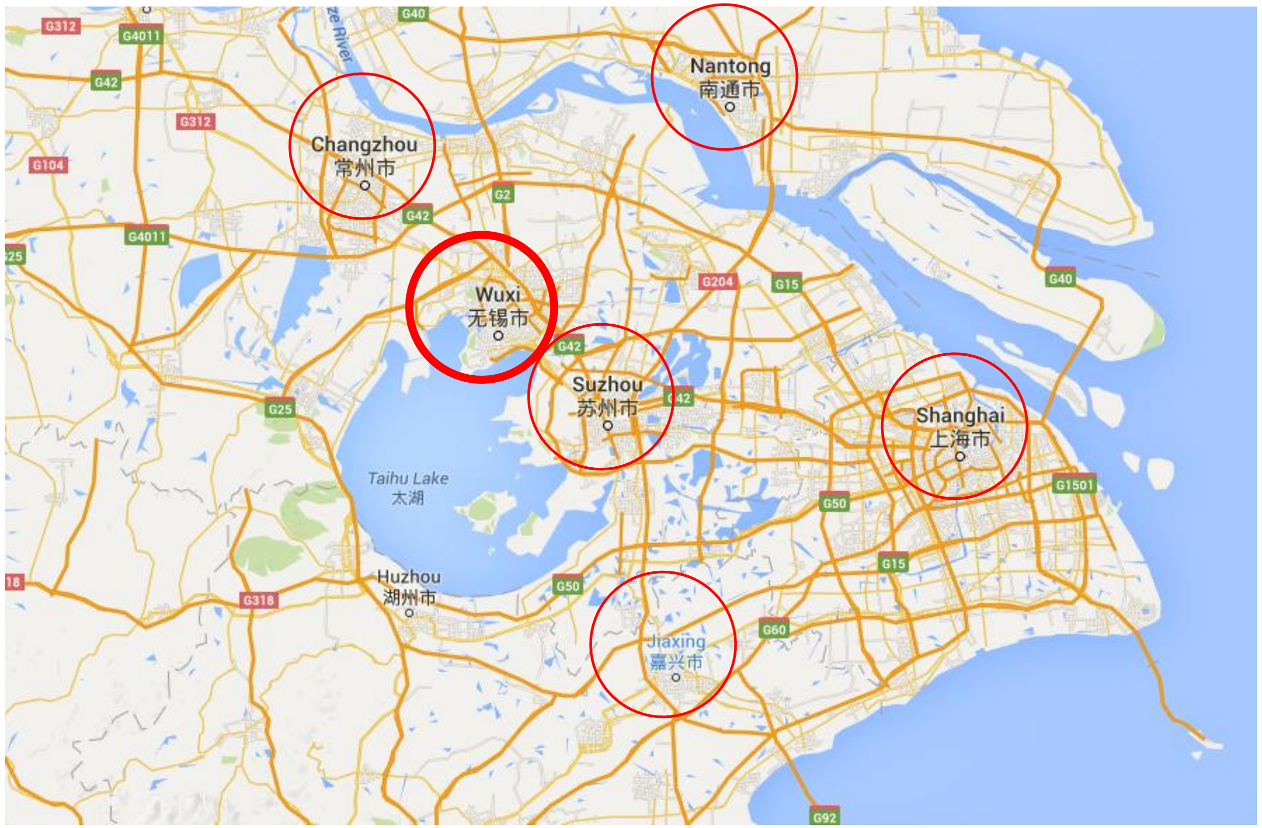
Sunway TaihuLight



神威

太湖之光

Sunway TaihuLight



City	Rank in Top100
Shanghai	1
Suzhou	7
Wuxi	14
Nantong	24
Changzhou	34
Jiaxing	50

The Sunway Machine Family

Sunway-I:
- CMA service, 1998
- commercial chip
- 0.384 Tflops
- 48th of TOP500

Sunway BlueLight:
- NSCC-Jinan, 2011
- 16-core processor
- 1 Pflops
- 14th of TOP500

Sunway TaihuLight:
- NSCC-Wuxi, 2016
- 260-core processor
- 125 Pflops
- 1st of TOP500

Sunway TaihuLight: Overview

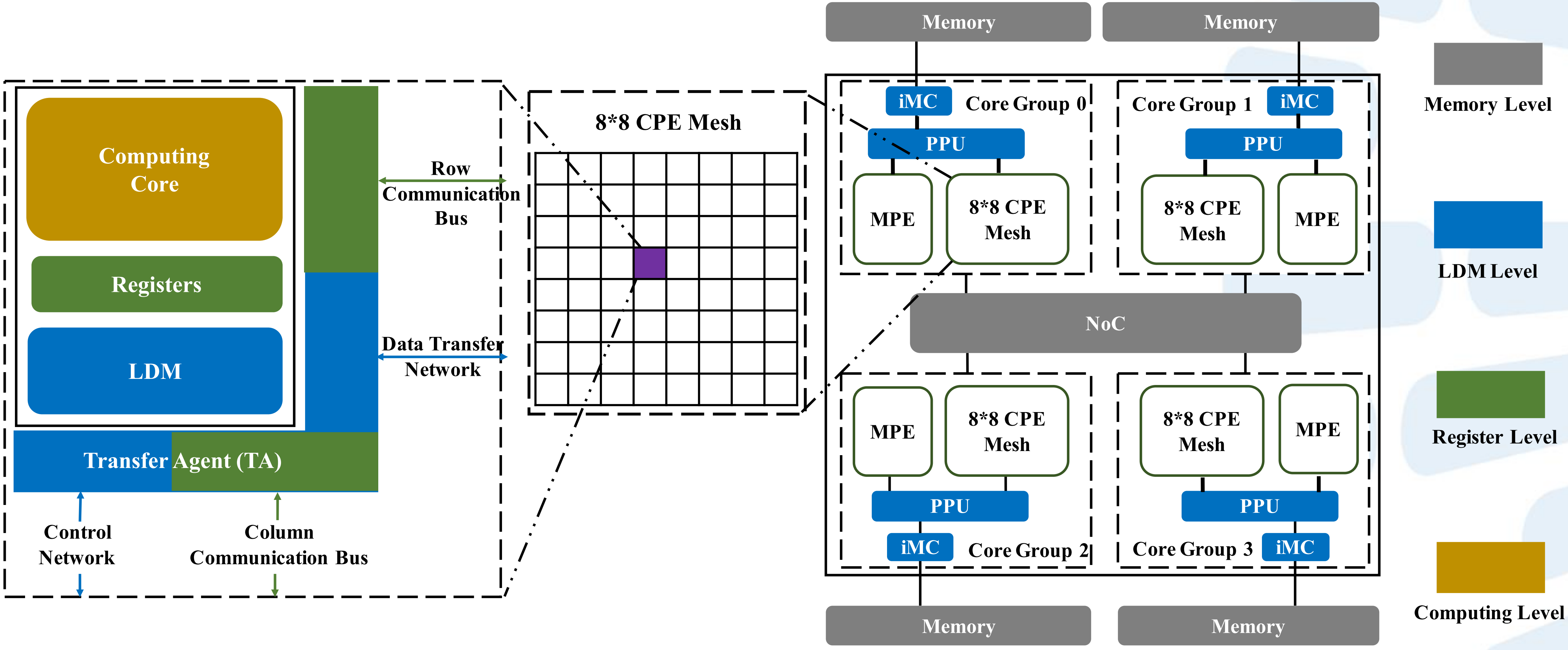
Entire System	
Peak Performance	125 PFlops
Linpack Performance	93 PFlops
Total Memory	1310.72 TB
Total Memory Bandwidth	5591.45 TB/s
# nodes	40,960
# cores	10,649,600

Sunway TaihuLight: Overview

Each Node	
Peak Performance	3.06 TFlops
Memory	32 GB
Memory Bandwidth	136.5 GB/s
# CPU	1
# cores	260

- 260 cores per processor
- 4 Core Groups (CGs), each of which has:
  - 1 Management Processing Element (MPE)
  - 64 (8x8) Computing Processing Elements (CPEs)

SW26010: Sunway 260-Core Processor

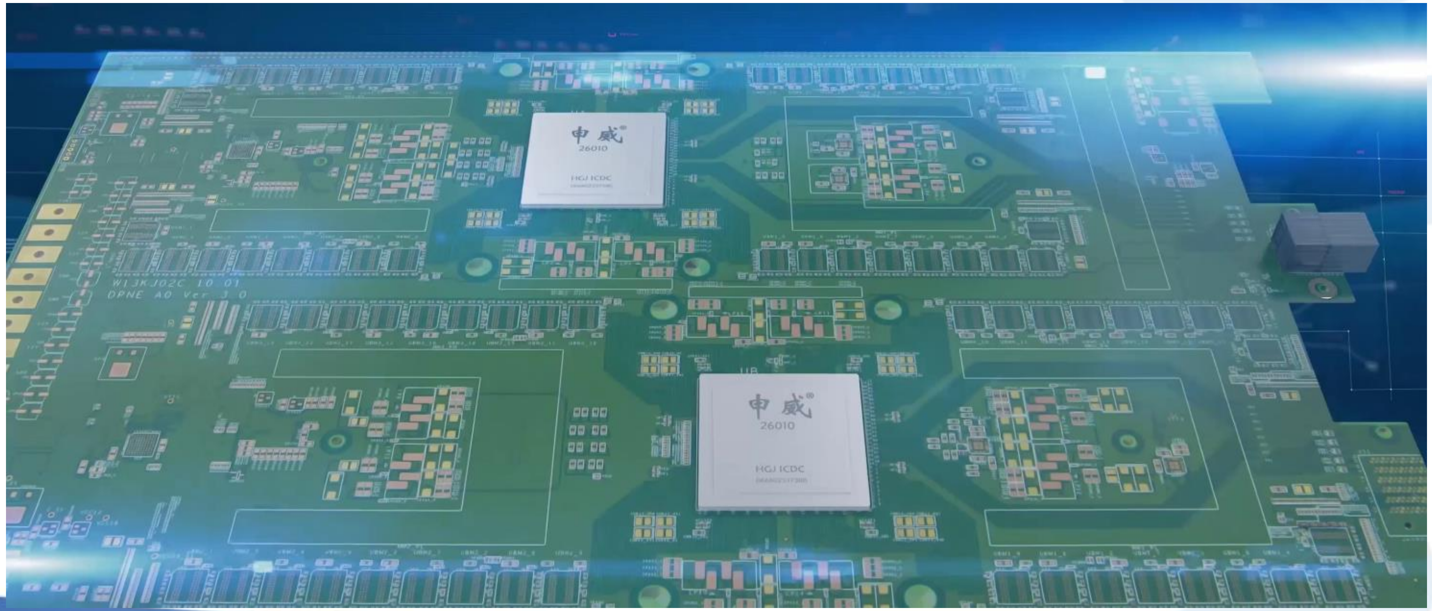


High-Density Integration of the Computing System

- A Five-Level Integration Hierarchy
  - computing node
  - computing board
  - super node
  - cabinet
  - entire computing system

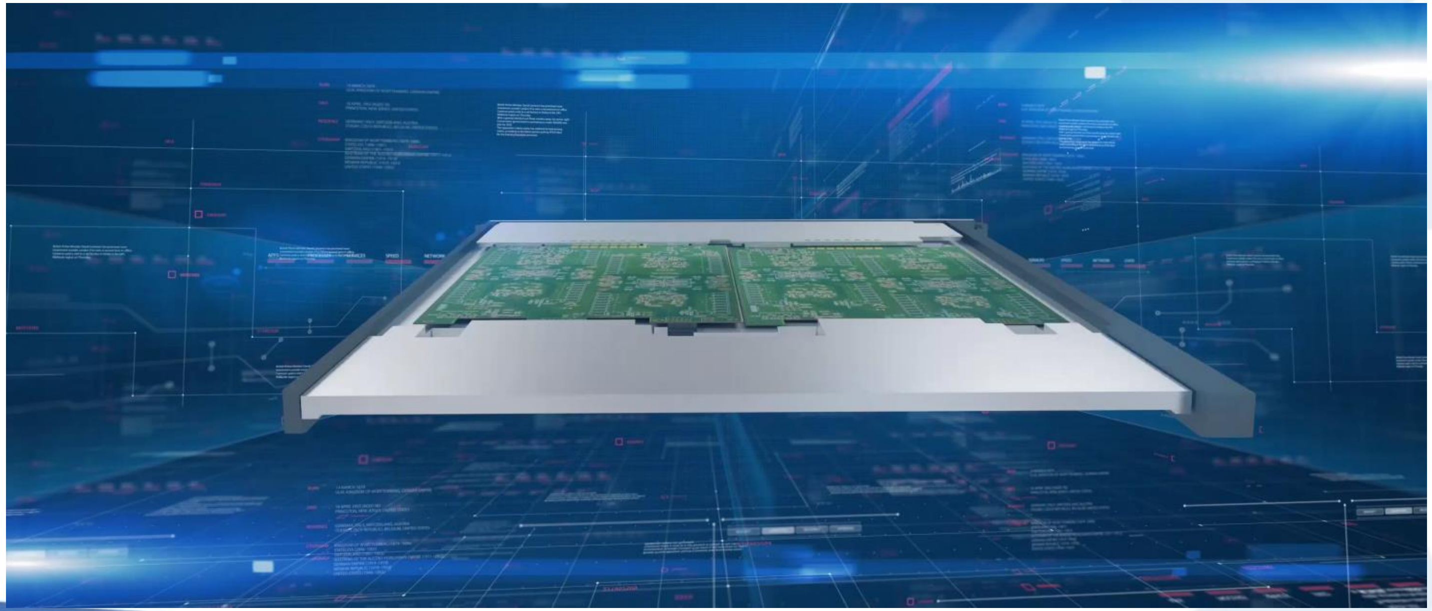
High-Density Integration of the Computing System

- A Five-Level Integration Hierarchy
  - computing node
  - computing board
  - super node
  - cabinet
  - entire computing system



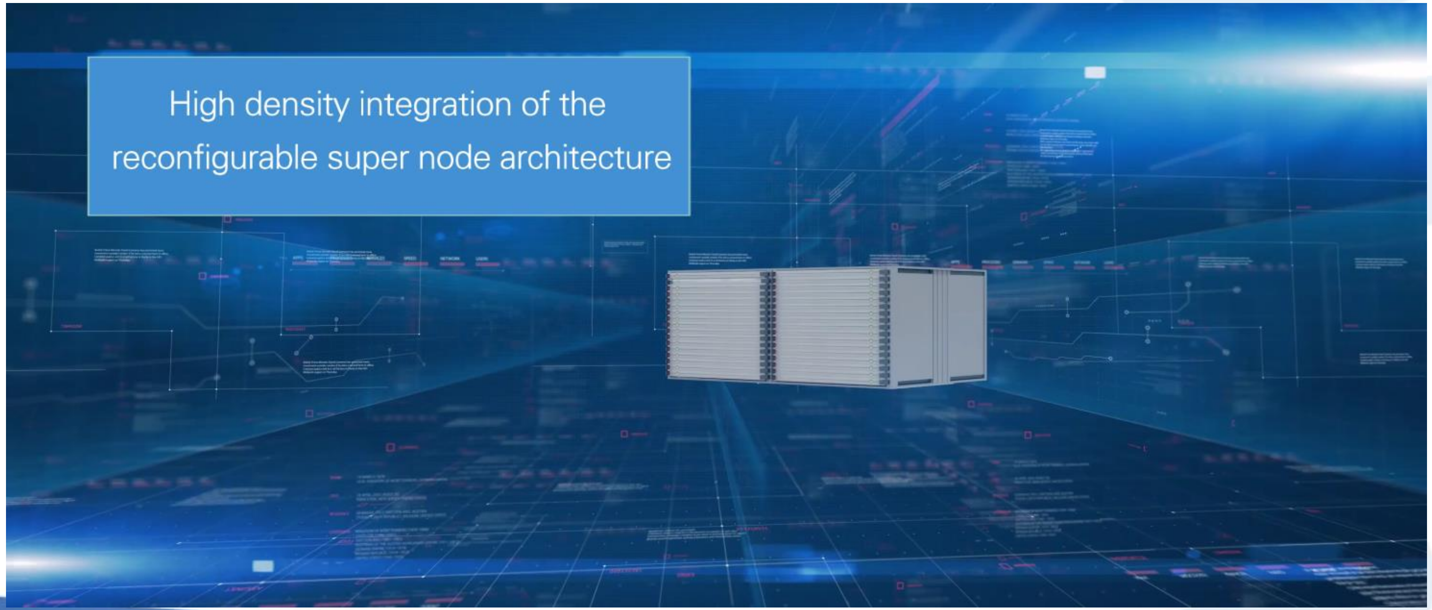
High-Density Integration of the Computing System

- A Five-Level Integration Hierarchy
  - computing node
  - computing board
  - super node
  - cabinet
  - entire computing system



High-Density Integration of the Computing System

- A Five-Level Integration Hierarchy
  - computing node
  - computing board
  - super node
  - cabinet
  - entire computing system



High-Density Integration of the Computing System

- A Five-Level Integration Hierarchy
  - computing node
  - computing board
  - super node
  - cabinet
  - entire computing system

How to Connect the 10 Million Cores?

$$40 \times 4 \times 256 \times 4 \times (1+8 \times 8) = 10{,}649{,}600$$

How to Connect the 10 Million Cores?

$$40\times4\times256\times4\times(1+8\times8) = 10{,}649{,}600$$

2D core array
with row and
column buses

How to Connect the 10 Million Cores?

$$40\times4\times256\times4\times(1+8\times8) = 10{,}649{,}600$$

2D core array with row and column buses

Network on Chip

How to Connect the 10 Million Cores?

$$40 \times 4 \times 256 \times 4 \times (1+8 \times 8) = 10,649,600$$

2D core array with row and column buses

Network on Chip

Customized Network Board to Fully Connect 256 Nodes

How to Connect the 10 Million Cores?

$$40\times4\times256\times4\times(1+8\times8) = 10{,}649{,}600$$

- $(1+8\times8)$: 2D core array with row and column buses
- 4: Network on Chip
- 256: Customized Network Board to Fully Connect 256 Nodes
- 40×4: Sunway Net

Sunway TaihuLight V.S. Other Systems

System	TaihuLight	Tianhe-2	Titan	Sequoia	Cori
Peak Performance (PFlops)	**125.4**	54.9	27.1	20.1	27.9
Total Memory (TB)	**1310**	1024	710	1572	879
Linpack Performance (PFlops)	**93.0(74%)**	33.9(62%)	17.6(65%)	17.2(85.3)	14.0(50%)
Rank of Top500	**1**	2	3	4	5
Performance/Power (Mflops/W)	**6051.3**	1901.5	2142.8	2176.6	3266.8
Rank of Green500	**4**	135	100	90	26
GTEPS	**23755.7**	2061.48	###	23751	###
Rank of Graph500	**2**	8	###	3	###
HPCG (Pflops)	**0.3712**	0.5801	0.3223	0.3304	0.3554
Rank of HPCG	**4**	2	7	6	5

Tweet Comments from Prof. Satoshi Matsuoka

Satoshi Matsuoka
@ProfMatsuoka

I was quite impressed with the engineering quality of TaihuLight, different from previous Chinese machines; now truly rivals US, Japan in SC twitter.com/profmatsuoka/s…

下午4:40 - 2016年11月3日 发自 東京 目黒区

Tweet Comments from Prof. Satoshi Matsuoka

Satoshi Matsuoka
@ProfMatsuoka

I was q
differen
Japan i

下午4:40

Satoshi Matsuoka
@ProfMatsuoka

TaihuLight physical design is excellent with low num. of chips, dual-sided surface mounting of all components for dense cold plate cooling .

下午5:57 - 2016年11月3日

Tweet Comments from Prof. Satoshi Matsuoka


Satoshi Matsuoka
@ProfMatsuoka

I was q
differen
Japan i

下午4:40

Satoshi Matsuoka
@ProfMatsuoka

TaihuLigh
dual-sided
plate cool

下午5:57 - 2

Satoshi Matsuoka
@ProfMatsuoka

Also impressive was their software and application efforts. Contrary to my speculations OpenACC does work, used in many of their real apps.

下午5:59 - 2016年11月3日

Tweet Comments from Prof. Satoshi Matsuoka

Satoshi Matsuoka
@ProfMatsuoka

Finally their design was cost&utility conscious. No expensive parts, quacky architecture, etc. Sunway apparently plans to sell the machine.

下午6:08 - 2016年11月3日

Outline

- The Sunway Machine
- The Programming Model
- Scaling Over 10-Million Cores: A Climate Example

Programming Model on TaihuLight

MPI + X

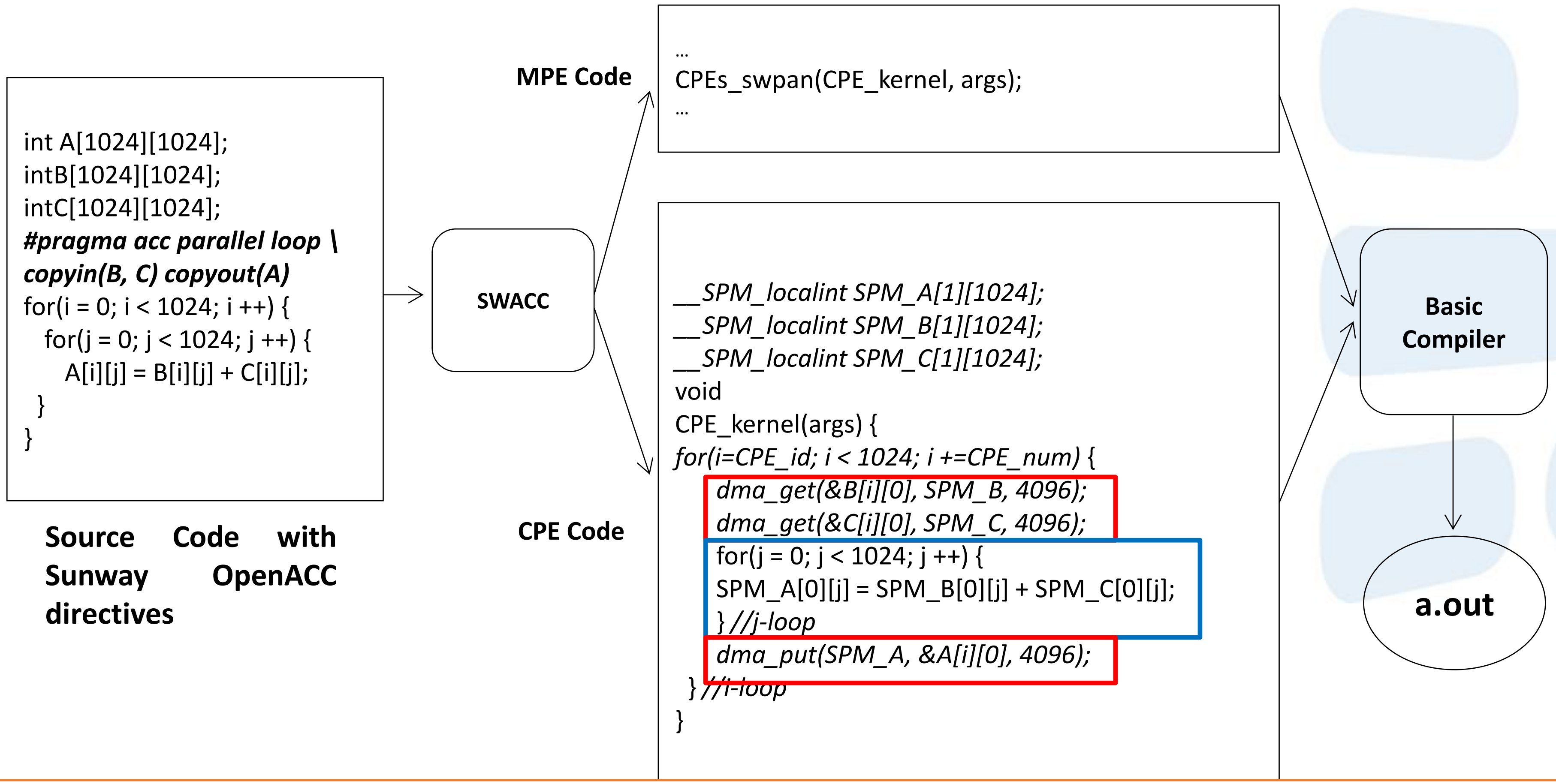
X: (Sunway OpenACC / Athread)

- MPI
  - One MPI process runs on one management core (MPE)
- Sunway OpenACC
  - Sunway OpenACC conducts data transfer between main memory and local data memory (LDM), and distributes the kernel workload to the computing cores (CPEs)
- Athread
  - Athread is the threading library to manage threads on computing core (CPE), which is used in the Sunway OpenACC implementation

Brief Overview of Sunway OpenACC

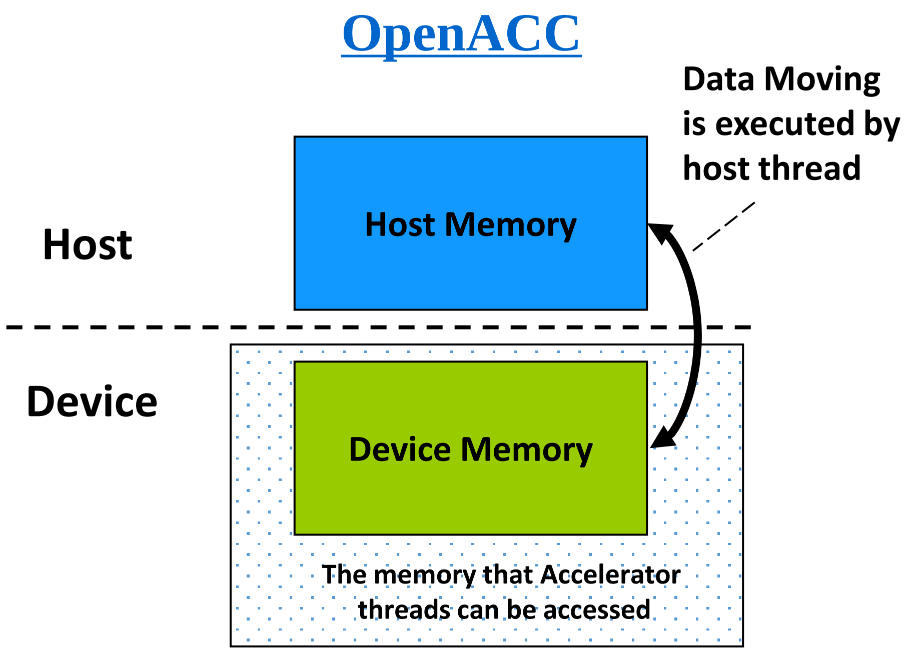
- Sunway OpenACC is a directive-based programming tool for SW26010
  - OpenACC2.0 based
  - Extensions for the architecture of SW26010
  - Supported by SWACC/SWAFORT compiler
  - Source-to-Source compiler
  - Based on ROSE compiler infrastructure (0.9.6a)
    - An open Source compiler infrastructure to build source-to-source program transformation and analysis
    - Developed by LLNL

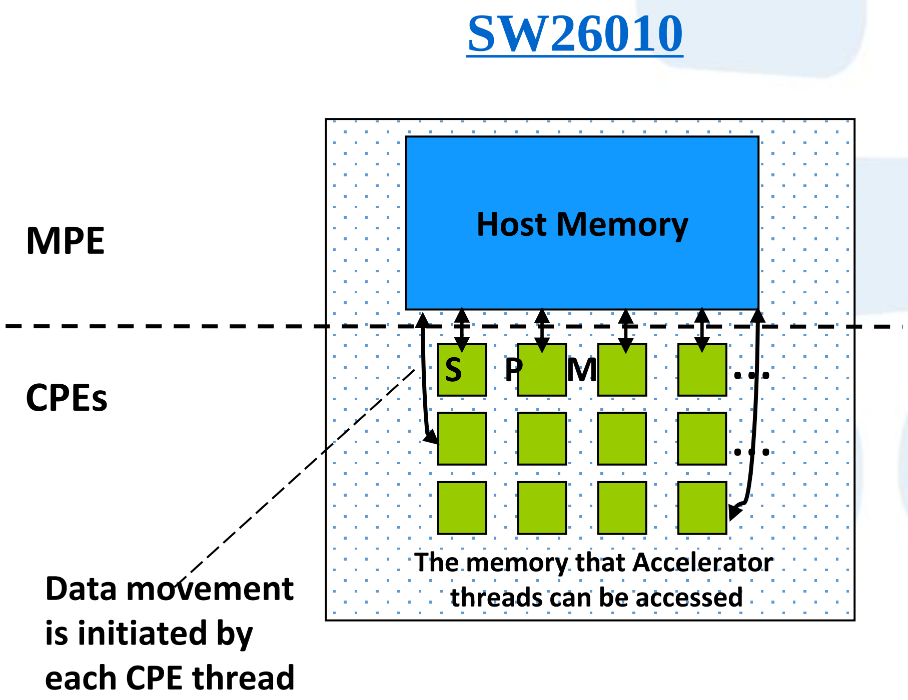
Brief View of Sunway OpenACC compiler



Compute pattern: data into SPM -> calculation -> data out to Main memory
Workload distribution and the size for data transfer are automatically determined by compiler

The difference between the Memory Models





The Principal Extension of Sunway OpenACC

- Extend the usage of data environment directives
  - Use *data copy* inside the accelerator parallel region
  - *copy on parallel* perform data moving and distributing between LDMs
- Add new directives/clauses
  - *local* clause, to allocate space on **LDM** of CPE thread.
  - Data transform support to speedup data transfer
    - *pack/packin/packout* clause
    - *swap/swapin/swapout* clause
  - *annotate* clauses to better controls of data movement and execution from compiler
    - *tilemask, entire, co_compute*

Directly data transfer between MEM and LDM

- Use *data copy* to handle data moving between Mem and LDM

```
!$acc data copyin(A) copyout(B)
!$acc parallel loop
  do i=1,128
     m = func(i)
      do j=1,128
        B(j, i) = A(j, m)
     enddo
  enddo
  !$acc end parallel loop
  !$acc end data
```

OpenACC2.0

- Moving A, B between host memory and device memory (e.g. global memory on GPU)
- Executed by host thread

```
!$acc parallel loop
do i=1,128
   m = func(i)
    !$acc data copyin(A(*, m)) copyout(B(*, i))
   do j=1,128
      B(j, i) = A(j, m)
   enddo
    !$acc end data
enddo
!$acc end parallel loop
```

Sunway OpenACC

- Moving A(*, m)、B(*, i) between host memory and LDM in each i-loop
- Executed by each CPE thread

Distributed moving data between MEM and LDMs

- Use *copy clause of parallel* to move and distribute data between Mem and LDMs

```
!$acc parallel loop copyout(C) copyin(A, B)
do i = 1, 256
  do j = 1, 512
     C(j, i) =  A(j, i) + B(j, i)
     end do
  end do
!$acc end parallel loop
```

OpenACC2.0

- Moving A、B between host memory and device memory
- Executed by host thread

Sunway OpenACC

- Moving A(*, i)、B(*, i)、C(*, i) between host memory and LDM in each i-loop
- Executed by each slave thread on CPE
- Data distribution controlled by compiler
- For readonly arrays with small size，can use *copyin(arr) annotate(entire(arr))* to specify that the *arr* will be put totally into LDM of Each CPE

Control the size of data moved to LDM

- Use *tile clause* to control the granularity of data moved to LDM

```
!$acc parallel loop copyin(A, B) copyout(C)
do i = 1, 64
  !$acc loop tile(2)
  do j = 1, 128
     do k = 1, 256
        C(k, j, i) = A(k, j, i)  + B(k, j, i)
      end do
    end do  !end of j-loop
    !$acc end loop
end do  ! end of i-loop
!$acc end parallel loop
```

tile:

- allocate buffer(256, 2, 1) in each LDM for A、B、C.
- same size of data being moved to each LDM each round.
- two loops on j is assigned to each CPE thread.

- Add *tilemask* for better data transfer
  - *Tilemask(var-list)* means the *tile* will not affect the variables in var-list.
  - Move more data in one transfer.
  - Buffer_C(1,1) will be allocated in LDM without *tilemask*.
  - Buffer_C(128, 1) will be allocated with *tilemask*.

```
!$acc parallel loop copy(C) copyin(A, B)
do i=1,128
   !$acc loop  tile(1) annotate(tilemask(C))
   do k=1,128
      do j=1,128
        C(k, i) = C(k, i) + A(j, i) * B(k, j)
      enddo
   enddo
   !$acc end loop
enddo
!$acc end parallel loop
```

Local and private data management

- Add *local clause* to allocate LDM space for private data
  - Usage: *local(var-list)*
  - Variables in var-list are private for CPE thread, and will be placed in LDM.
  - Used for private variables with small size.
- Use *private+copy* to manage private data with large size
  - Array with large size can not be put into LDM.
  - Step 1: private(var-list), private vars will be allocated in private space of each CPE in Main Memory.
  - Step 2: copy(the-same-var-list), the private data will be copied into LDM, piece-by-piece.
  - Buffer_tmp(256, 2) will be allocated and maintained in LDM.

```
!$acc     parallel loop copyin(A, B) copyout(C)
!$acc&  private(tmp) copy(tmp)
do i = 1, 64
  !$acc loop tile(2)
  do j = 1, 128
     do k = 1, 256
        tmp(k, j) = A(k, j, i)  + B(k, j, i)
     end do
      … …   ! some compute on tmp(*,*)
      do k = 1, 256
         C(k, j, i) = tmp(k, j)
      end do
   end do  end do    ! end of j-loop
   !$acc end loop
end do  ! end of i-loop
!$acc end parallel loop
```

Packing data to improve transfer efficiency

- *pack clause*
  - *pack/packin/packout*
  - *pack = dataPack + copy*

- pack multiple variables into a new variable by MPE, and copy data between MEM and SPM with the new one by CPE .

- Most useful for multiple scalars.

```
!$acc parallel loop copyout(D) packin(A, B, C)
do i = 1, 64
  do j = 1, 32
    D(j, i) =  C(j, i) + A(j, i) + B(j, i)
    end do
  end do
!$acc end parallel loop
```

```
A(32, 64)
B(32, 64)        --dataPack-->     pkin_data(3,32,64)
C(32, 64)

   D(j, i) =  pkin_data(3, j, i) +
                pkin_data(2, j, i)+pakin_data(1, j, i)
```

Transposing array to improve transfer efficiency

- *swap clause*
  - *swap/swapin/swapout*
  - *swap = ArrayTranspose + copy*
- Improve the space-locality and the data transmission efficiency to avoid repeated stride copy
- Use CPE threads to perform array transpose for better bandwidth utilization.
- Efficient transpose algorithm , can support 6-dim array.

```
!$acc     parallel loop copyout(A) copyin(C)
!$acc&  swapin(B(dimension order:2, 3, 1))
do i = 1, 64
do j = 1, 32
do k = 1, 128
   A(k, j, i) = C(k, j, i) + B(i, k, j)
enddo
enddo
enddo
!$acc end parallel loop
```

B(64,128,32) —Transposed / use Accelerator→ B'(128,32,64)

A(k, j, i) = C(k, j, i) + B'(k, j, i)

Other Extensions

- Cooperative computing
  - Treat the host thread the same as device thread
  - N device threads, 1 host thread
  - N+1 threads execute the parallel loop
- Add *co_compute* clause
  - Used on *loop* directive
  - *annotate(co_compute)*

```
#pragma acc parallel copyout(A)
{
   #pragma acc loop annotate(co_compute)
    for(i = 0; i < 130; i++)
    {
        A[i] = i;
    }
}
```

Athread

Threading library to manage threads on CPEs

Similar to posix Pthreads

Routine	Functionality
int athread_init()	Initialize the athread library
int athread_create(int id, start_routine fpc, void *arg)	Start a routine with id executing the function pointed by fpc, the parameters for the function fpc are pointed by arg
int athread_wait(int id)	Wait for the completion of the thread ID
int athread_end(int id)	Terminate the thread by specifying thread ID
int athread_spawn(start_routine fpc, void *arg)	Spawn a set of threads making use of all CPEs
int athread_get_max_threads()	Get the maximum number of active threads
int athread_get_id()	Get the ID of current threads
….	

Athread example

MPE code

a.c 〉 No Selection

```c
#include <athread.h>
extern SLAVE_FUN(fun_sw)();
int main() {
    int a[64] = {0};
    int i;
    for(i = 0 ; i < 64 ; i ++)
        a[i] = i;
    athread_init();
    athread_spawn(fun_sw, a);
    athread_join();
    for(i = 0 ; i < 64 ; i ++)
        printf("a[%d] = %d\n", i, a[i]);
    return 0;
}
```

CPE code

b.c 〉 No Selection

```c
#include<slave.h>
void fun_sw(int *a)
{
    int id = athread_get_id(-1);
    a[id] *= 2;
    return;
}
```

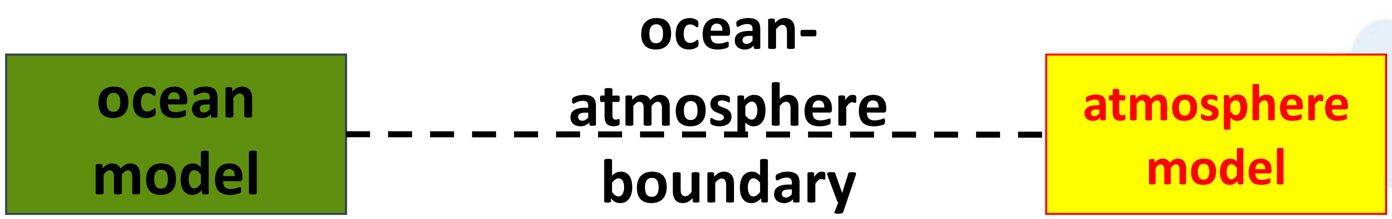
Outline

- The Sunway Machine
- The Programming Model
- Scaling Over 10-Million Cores: A Climate Example

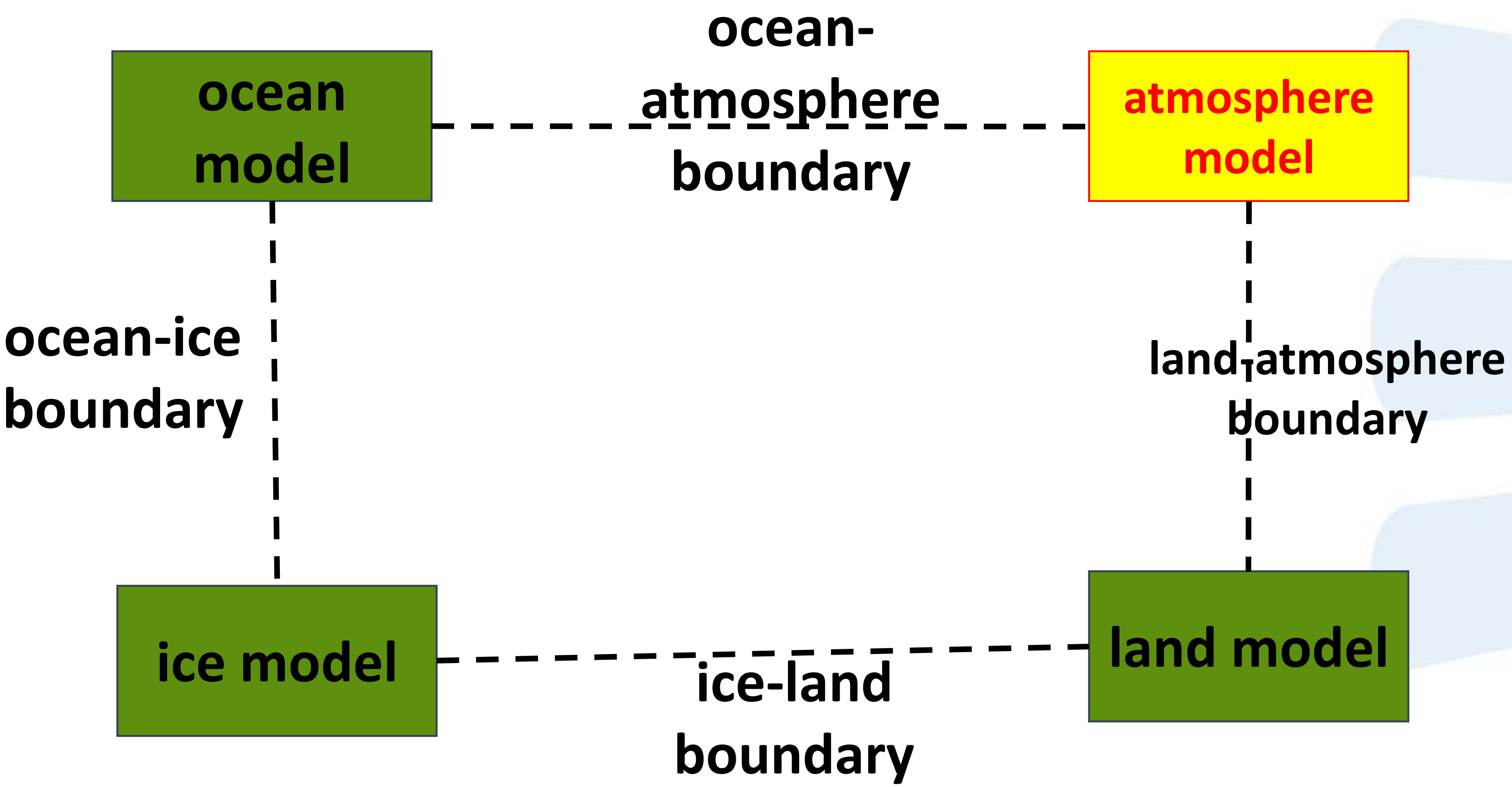
More and more component models

atmosphere model

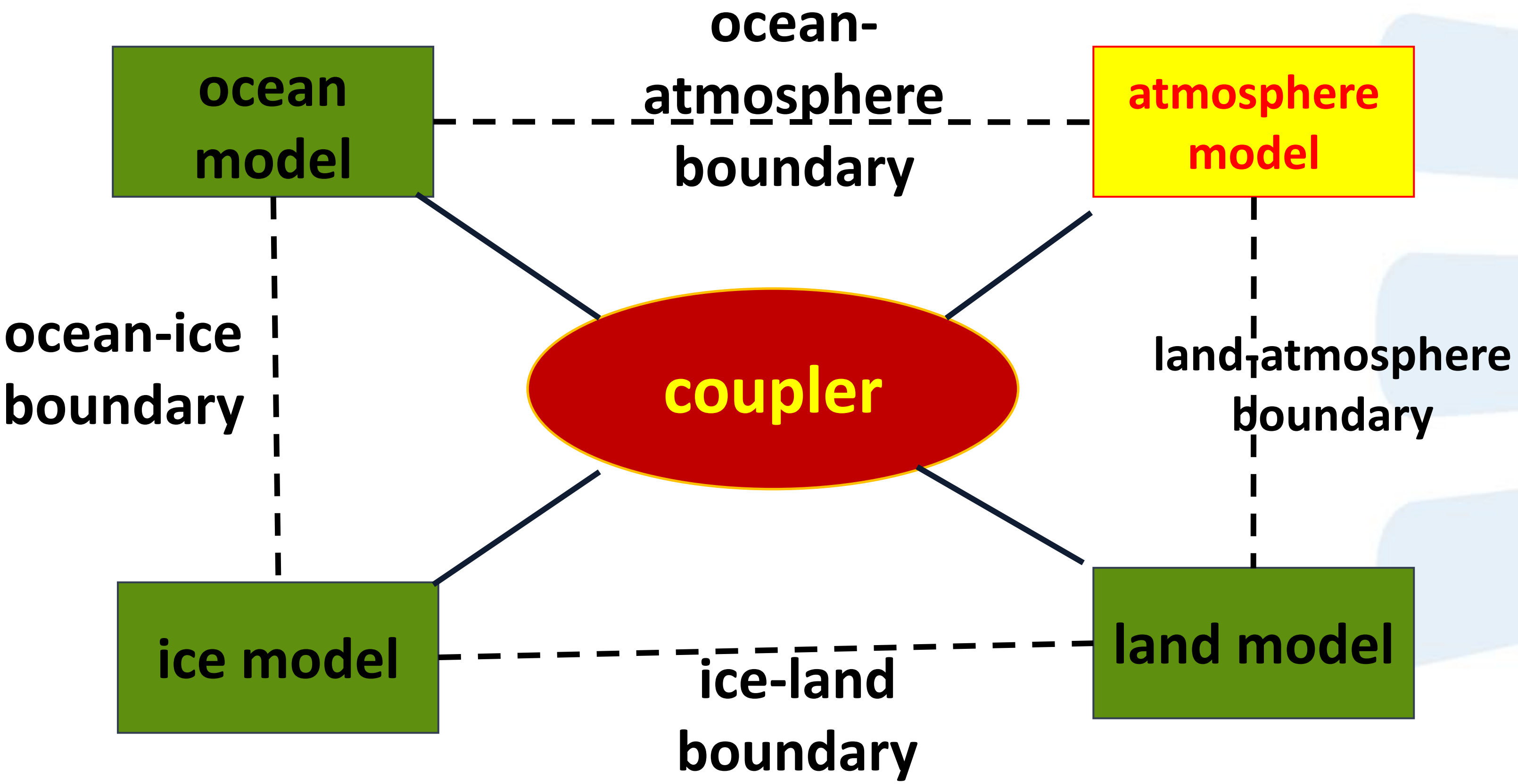
More and more component models



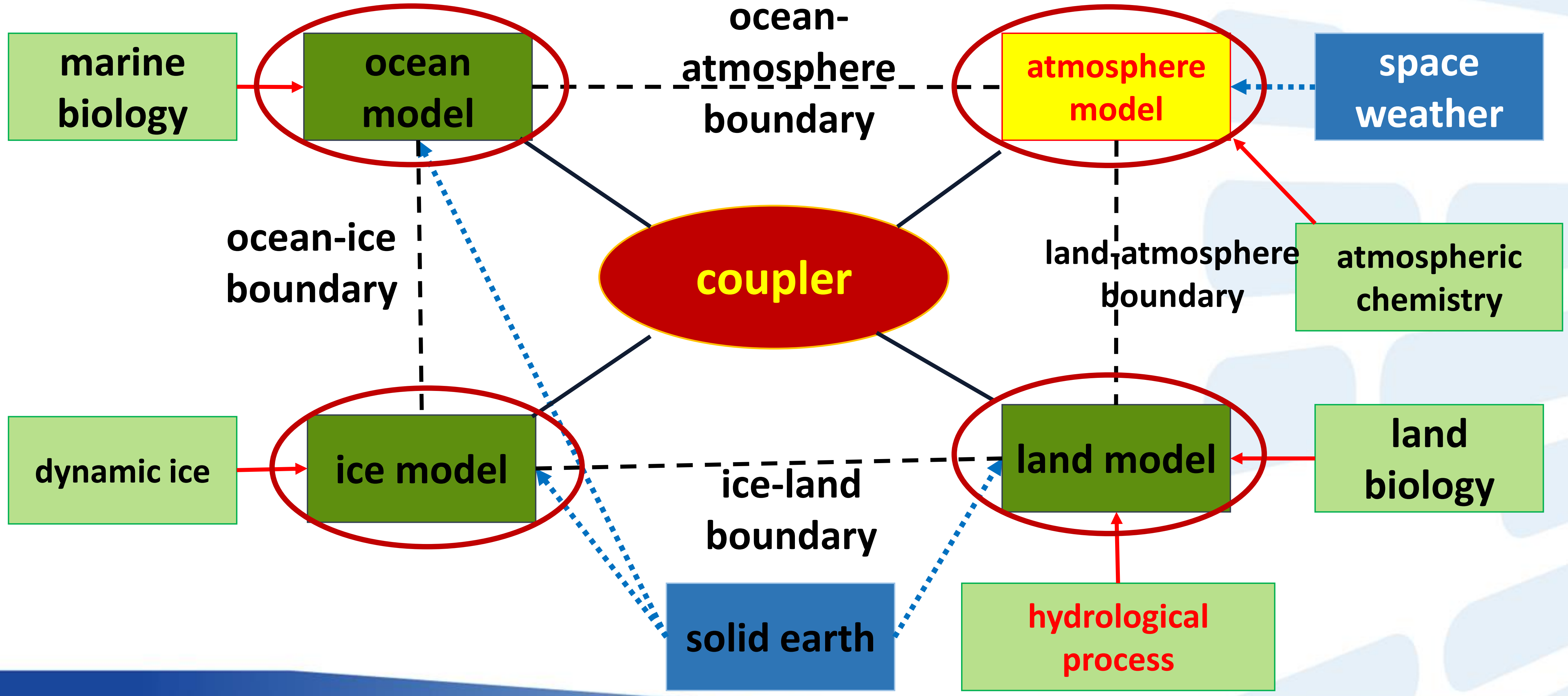
More and more component models



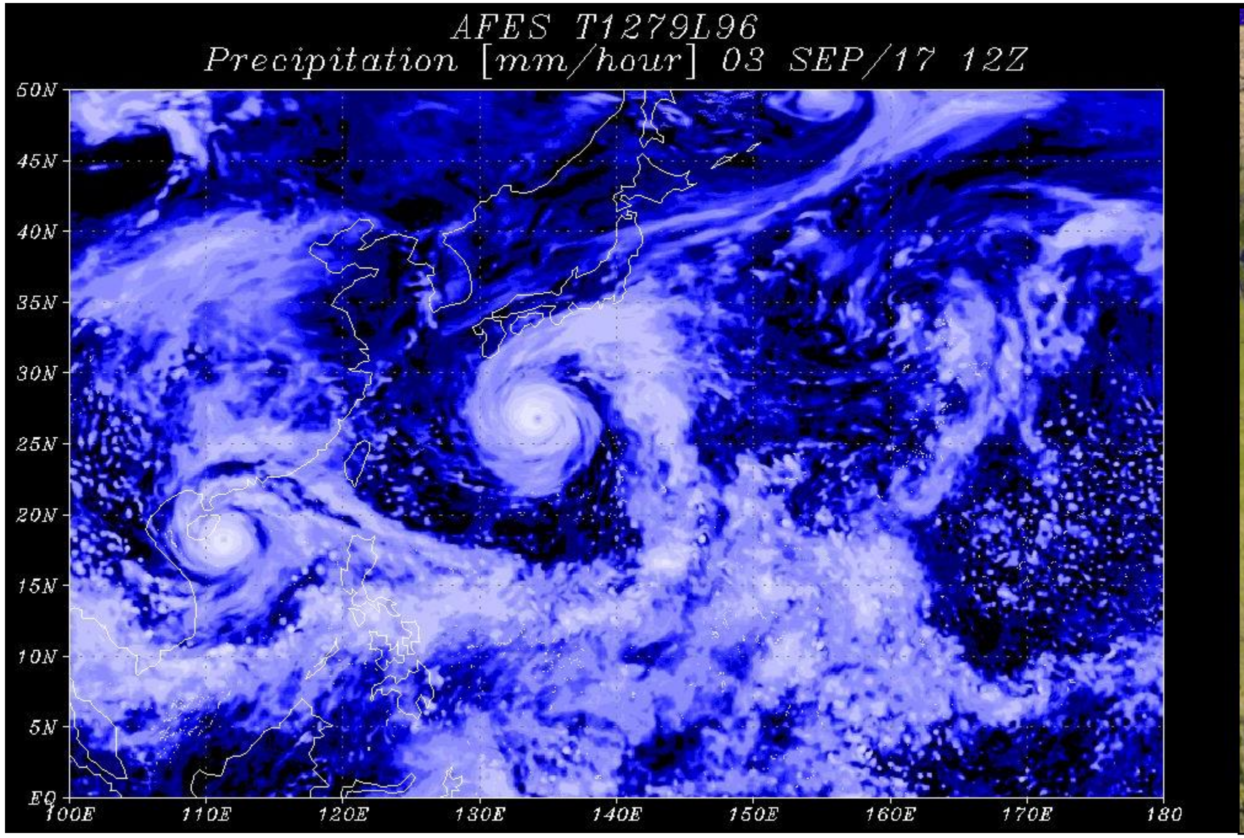
More and more component models

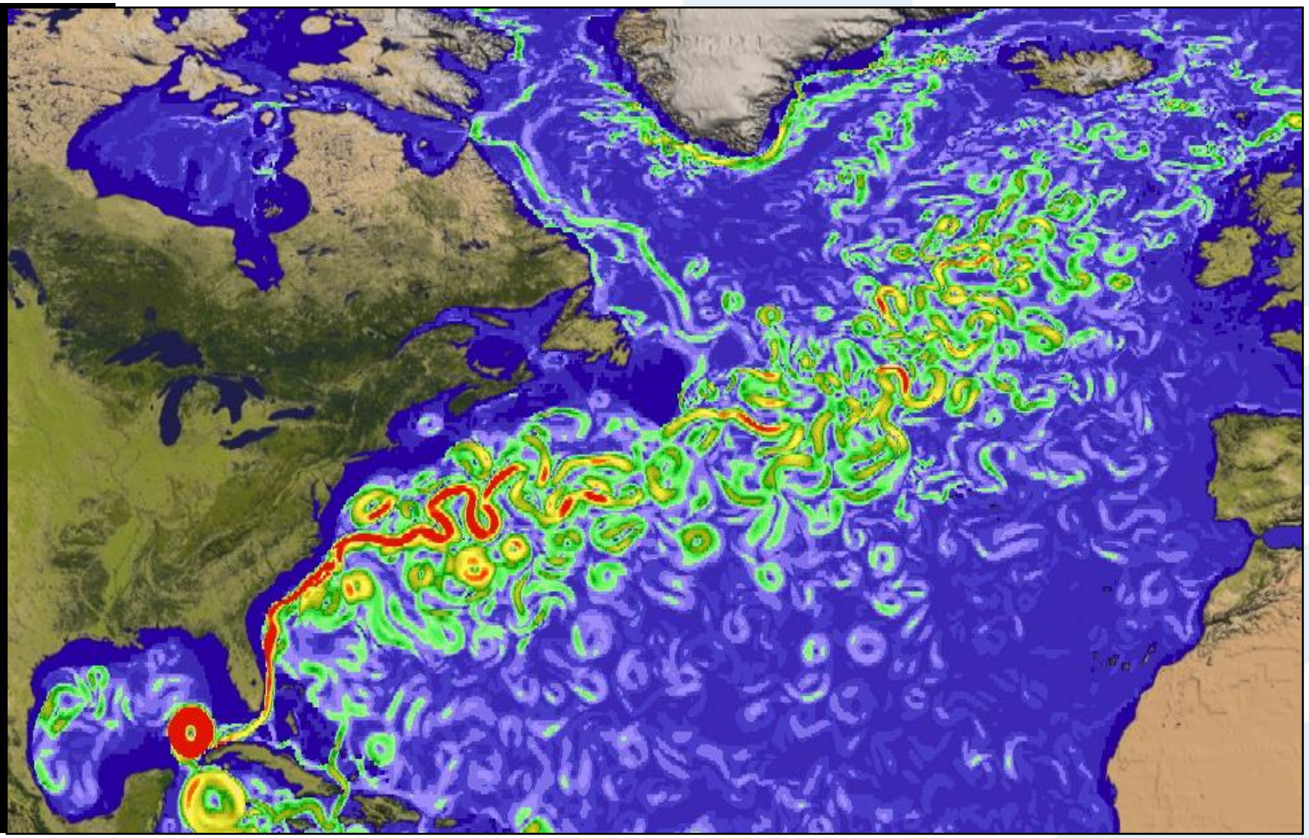


More and more component models

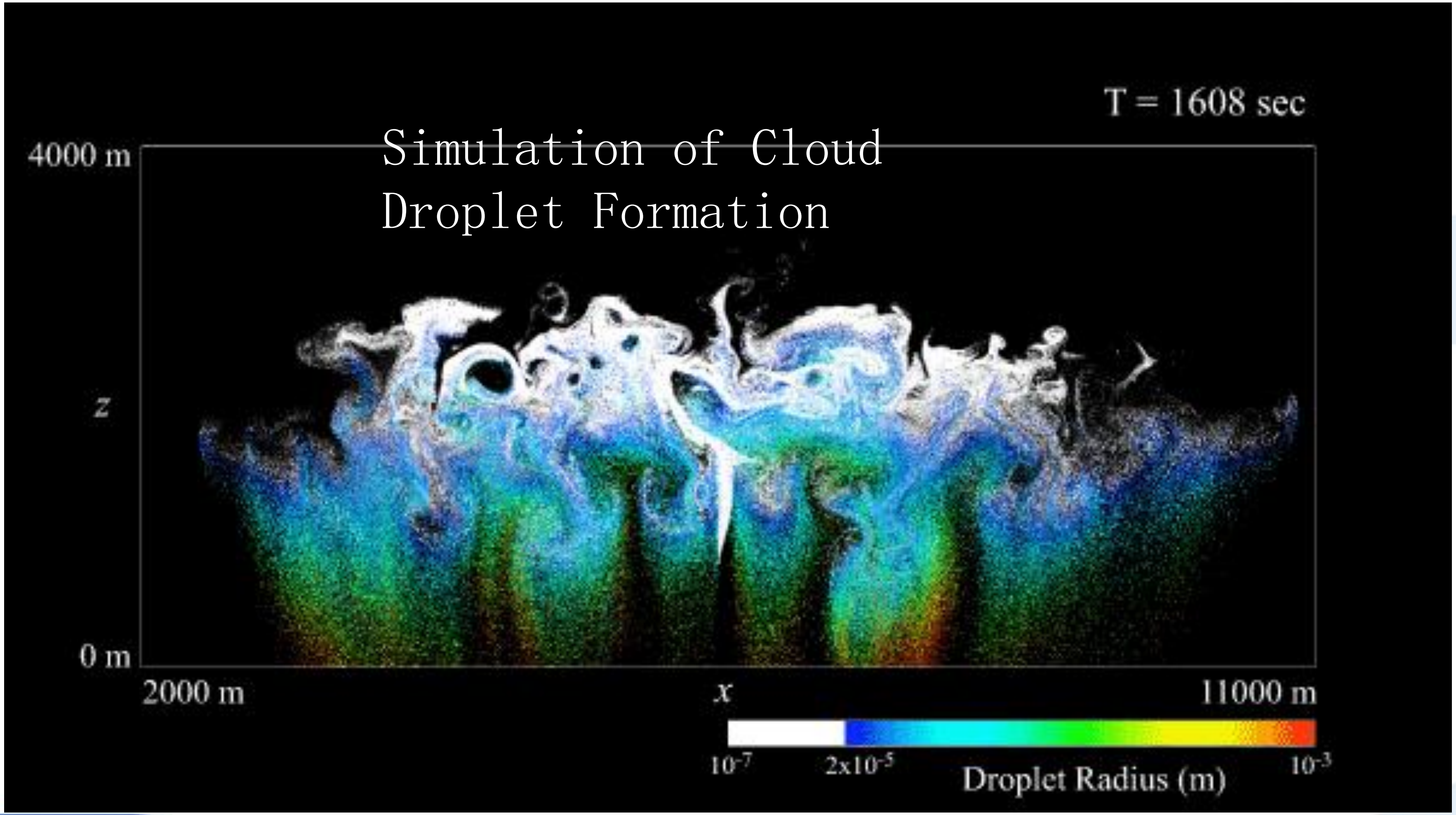


Increase in Spatial and Temporal Resolution to be Cloud-Resolving and Eddy-Resolving



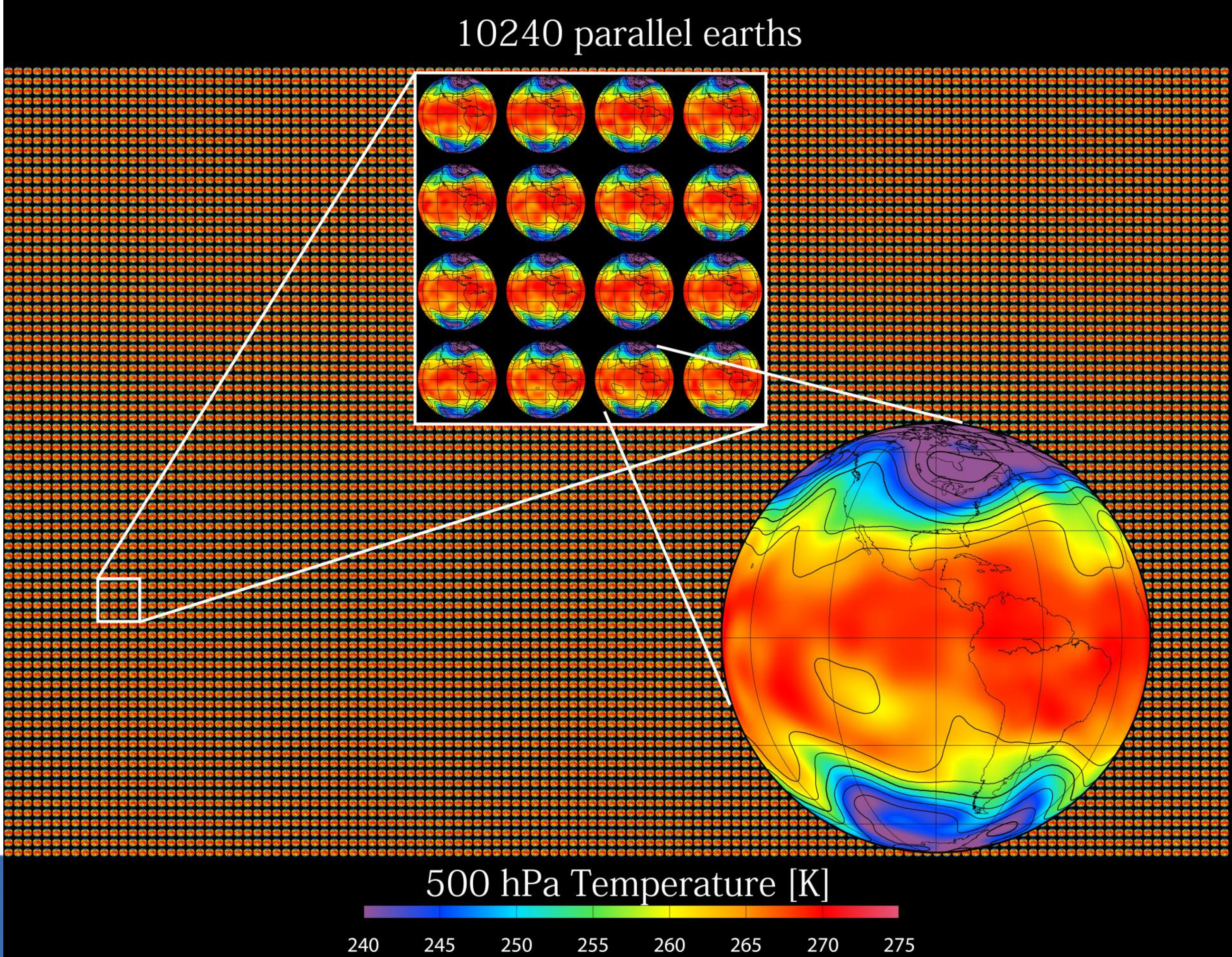


Simulation of more and more detailed physics processes



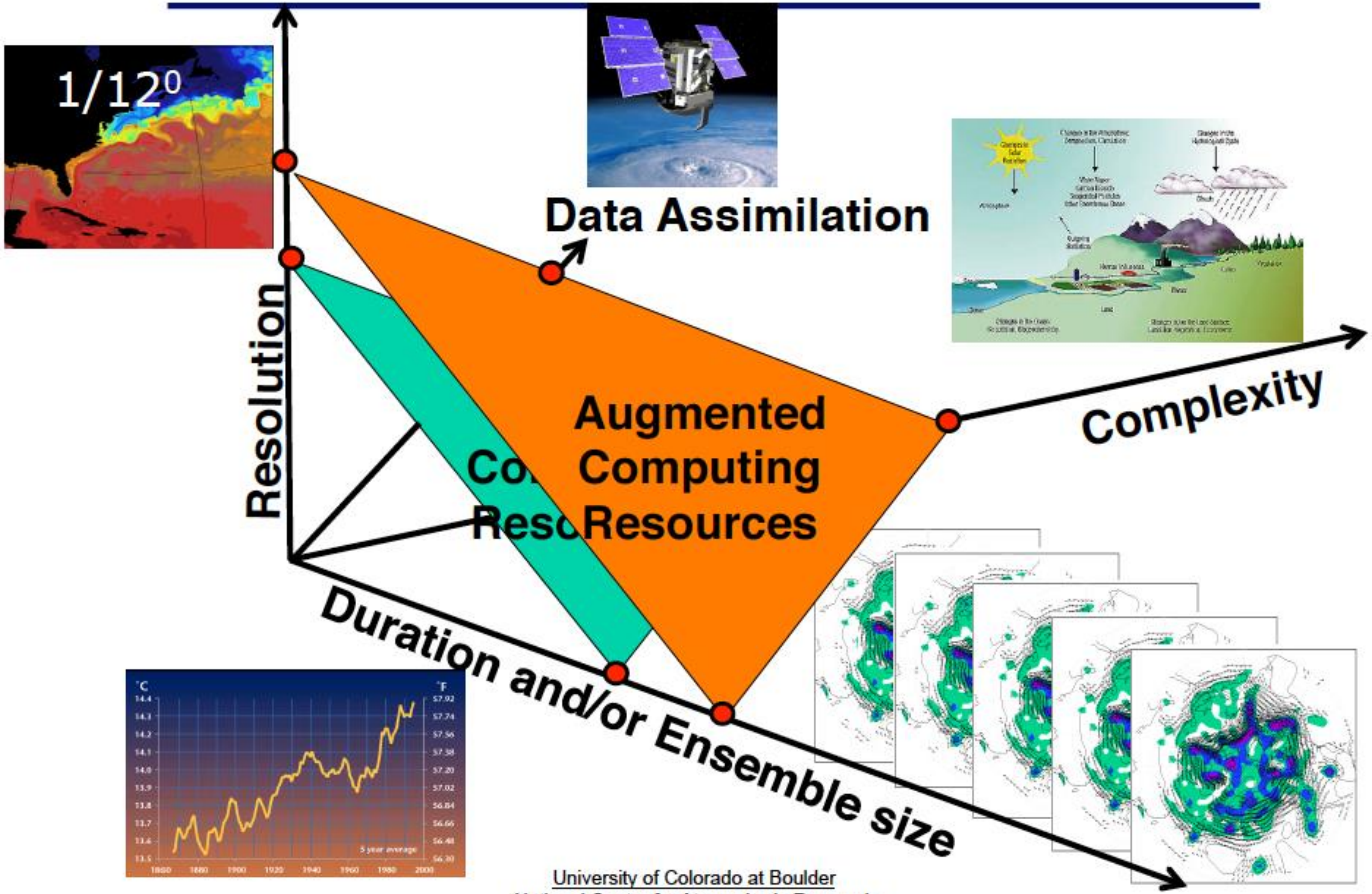
Online Ensembles

10240 NICAM Samples on K computer



Courtesy of Takemasa Miyoshi's talk at BDEC 2017, Wuxi.

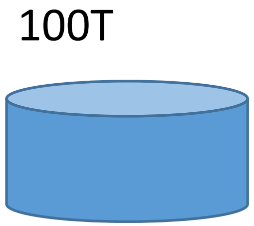
Balancing Science Goals with Computing Power



University of Colorado at Boulder
National Center for Atmospheric Research

The Gap between Software and Hardware

- millions lines of legacy code
- poor scalability
- written for multi-core, rather than many-core



China's models
- pure CPU code
- scaling to hundreds or thousands of cores

China's supercomputers
- heterogeneous systems with many-core chips
- millions of cores

Our Research Goals

- highly scalable framework that can efficiently utilize many-core processors
- automated tools to deal with the legacy code

100P

- millions lines of legacy code
- poor scalability
- written for m[illegible]e, rather than many-core

100T

China's models
- pure CPU code
- scaling to hundreds or thousands of cores

China's supercomputers
- heterogeneous systems with many-core chips
- millions of cores

Our Research Goals

- highly scalable framework that can efficiently utilize many-core processors
- automated tools to deal with the legacy code

- millions lines of legacy code
- poor scalability
- written for m[illegible], rather than many-core

100T

100P

China’s models
- pure CPU code
- scaling to hundreds or thousands of cores

China’s supercomputers
- heterogeneous systems with many-core chips
- millions of cores

Highly-Scalable Atmospheric Simulation Framework

cube-sphere grid or other grid

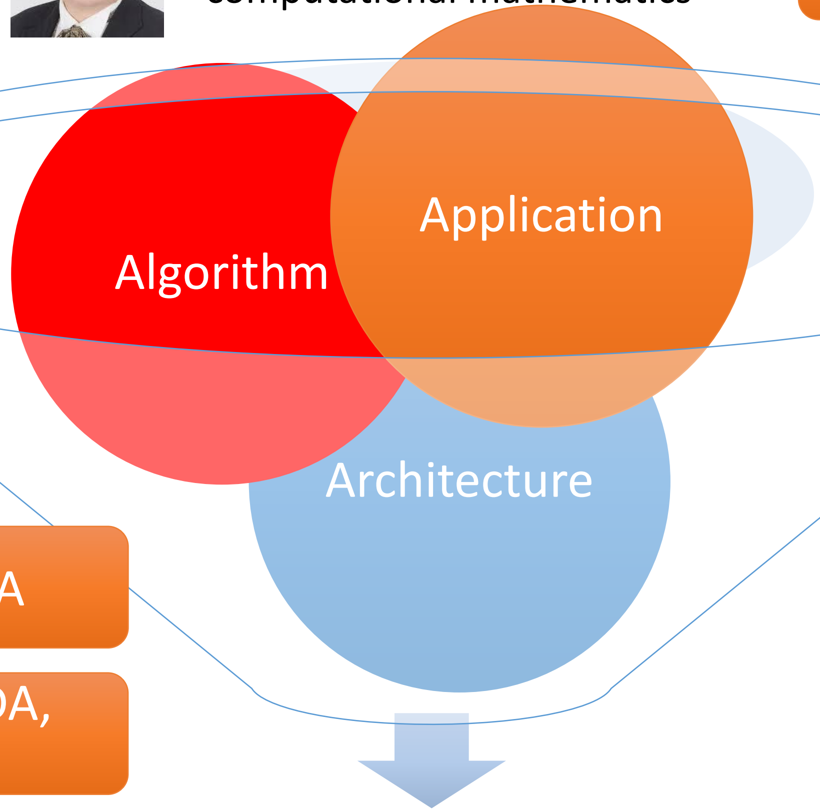
explicit, implicit, or semi-implicit method

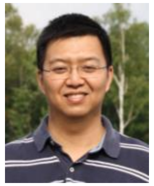
Yang, Chao
Institute of Software, CAS
computational mathematics

cloud resolving

Wang, Lanning
Beijing Normal University
climate modeling

Xue, Wei
Tsinghua University
computer science





Fu, Haohuan
Tsinghua University
geo-computing

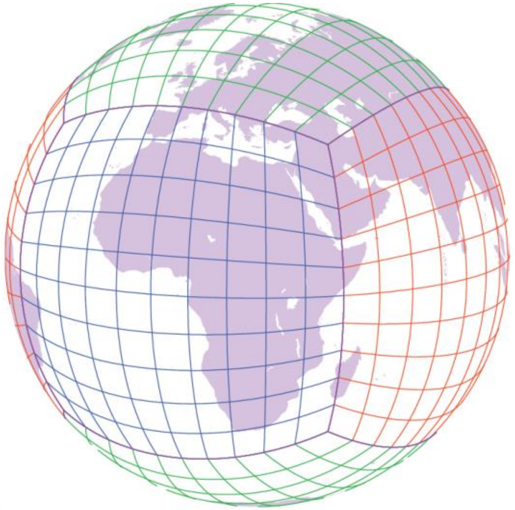
Sunway, GPU, MIC, FPGA

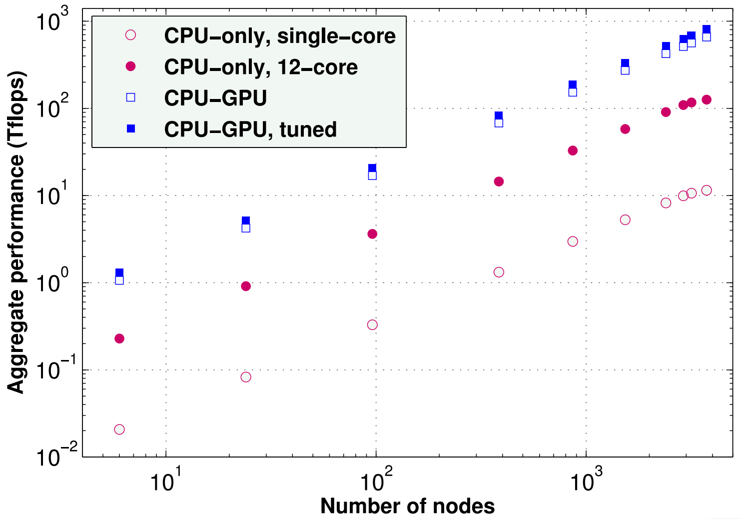
C/C++, Fortran, MPI, CUDA, Java, …

The “Best” Computational Solution

2012: 2D SWE Solver on Tianhe-1A

- Starting from shallow wave equation
  - cubed-sphere mesh grid
  - adjustable partition between CPU and GPU
  - scale to 40,000 CPU cores and 3750 GPUs with a sustainable performance of 800 TFlops

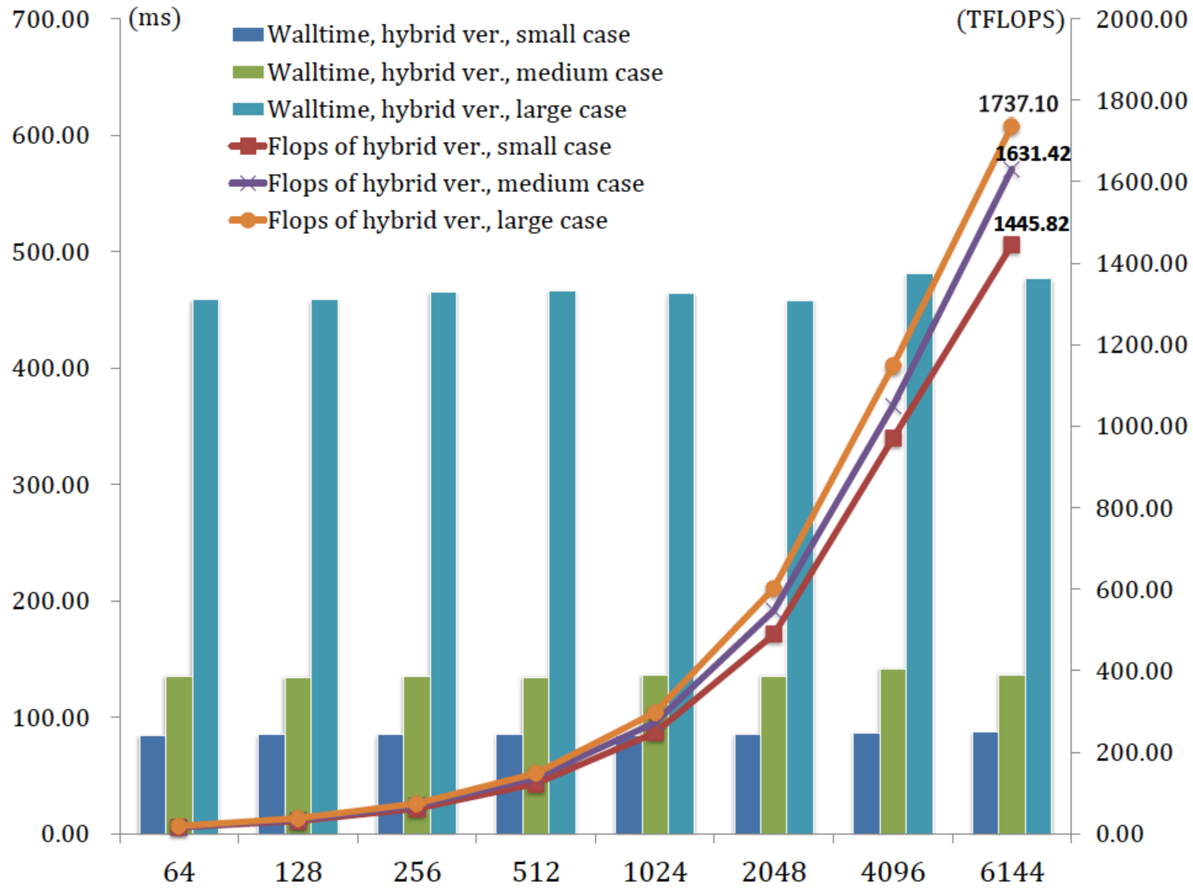




国家超级计算无锡中心
National Supercomputing Center in Wuxi

“A Peta-Scalable CPU-GPU Algorithm for Global Atmospheric Simulations”, in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pp. 1-12, Shenzhen, 2013. .

2013: 3D Euler Equation Solver on Tianhe-2



A Sustained Performance of 1.7 Pflops

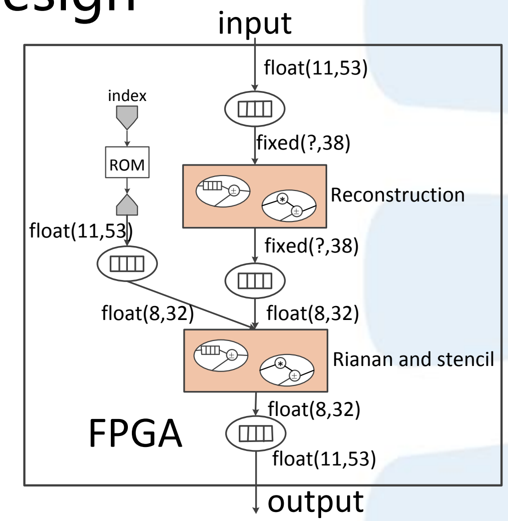
"Ultra-scalable CPU-MIC Acceleration of Mesoscale Atmospheric Modeling on Tianhe-2", IEEE Transaction on Computers.

2013: 2D SWE Solver on FPGA

- Customized mixed-preicion fully-pipelined hardware design

Algorithm 1 SWEs Algorithm

```
for patch 0 to patch 5 do
    Halo Updating
    for j ← 0 to n_j do
        for i ← 0 to n_i do
            Compute Local Coordinate                                  ▷ 58 FLOP
            Compute Left Intermediate Value, Including{
                State Reconstruction                                  ▷ 218 FLOP
                if i==0 then
                    Left Boundary computing 1                         ▷ 400 FLOP
                elsif i==1 then
                    Left Boundary computing 2                         ▷ 388 FLOP
                endif
                Rianann Operations }                                  ▷ 67 FLOP
            Compute Right Intermediate Value Including the Right Boundary
            Compute Bottom Intermediate Value Including the Bottom Boundary
            Compute Top Intermediate Value Including the Top Boundary
            Compute Upwind Stencil, h, hu^1 and hu^2                  ▷ 810 FLOP
        end for
    end for
end for
```



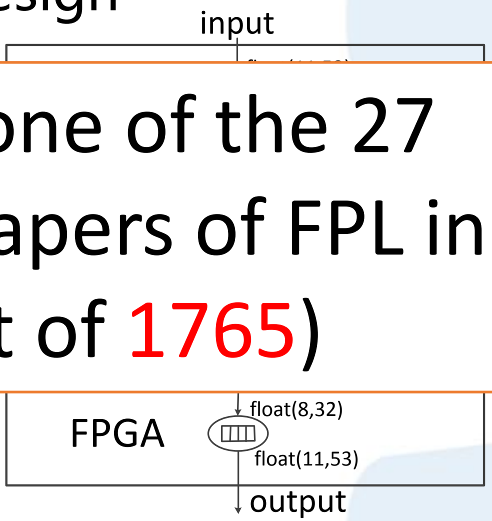
- 100x speedup over 6-core CPU, 5x speedup over GPU

Mesh size: $1024 \times 1024 \times 6$					
platform	performance (points/second)	speedup	power (Watt)	efficiency (points/(second·Watt))	power efficiency
6-core CPU	4.66K	1	225	20.71	1
Tianhe-1A node	110.38K	23x	360	306.6	14.8x
MaxWorkstation	468.11K	**100x**	186	2.52K	**121.6x**
MaxNode	1.54M	**330x**	514	3K	**144.9x**

"Accelerating Solvers for Global Atmospheric Equations Through Mixed-Precision Data Flow Engine", in Proceedings of the 23rd International Conference on Field Programmable Logic and Applications, 2013.

2013: 2D SWE Solver on FPGA

- -pipelined hardware design



Selected as one of the 27 Significant Papers of FPL in 25 Years (27 out of 1765)

- 100x speedup over 6-core CPU, 5x speedup over GPU

Mesh size: 1024 × 1024 × 6					
platform	performance (points/second)	speedup	power (Watt)	efficiency (points/(second·Watt))	power efficiency
6-core CPU	4.66K	1	225	20.71	1
Tianhe-1A node	110.38K	23x	360	306.6	14.8x
MaxWorkstation	468.11K	**100x**	186	2.52K	**121.6x**
MaxNode	1.54M	**330x**	514	3K	**144.9x**

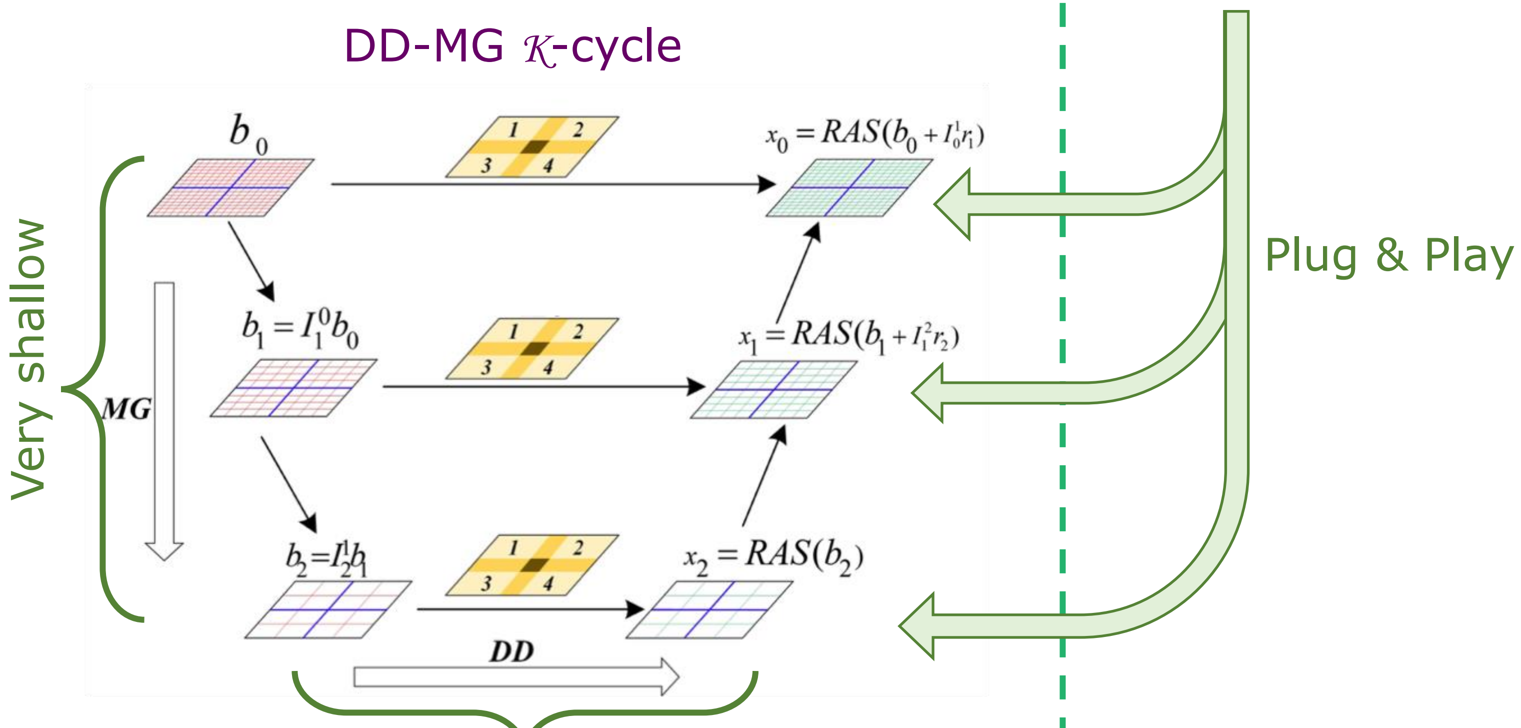
"Accelerating Solvers for Global Atmospheric Equations Through Mixed-Precision Data Flow Engine", in Proceedings of the 23rd International Conference on Field Programmable Logic and Applications, 2013.

163,840 processes | 65 threads

racks × *chips* × *core-groups* × *cores* = *total number of cores*

40 × 1,024 × 4 × 65 = 10,649,600

DD-MG $\mathcal{K}$-cycle



Uniform DD

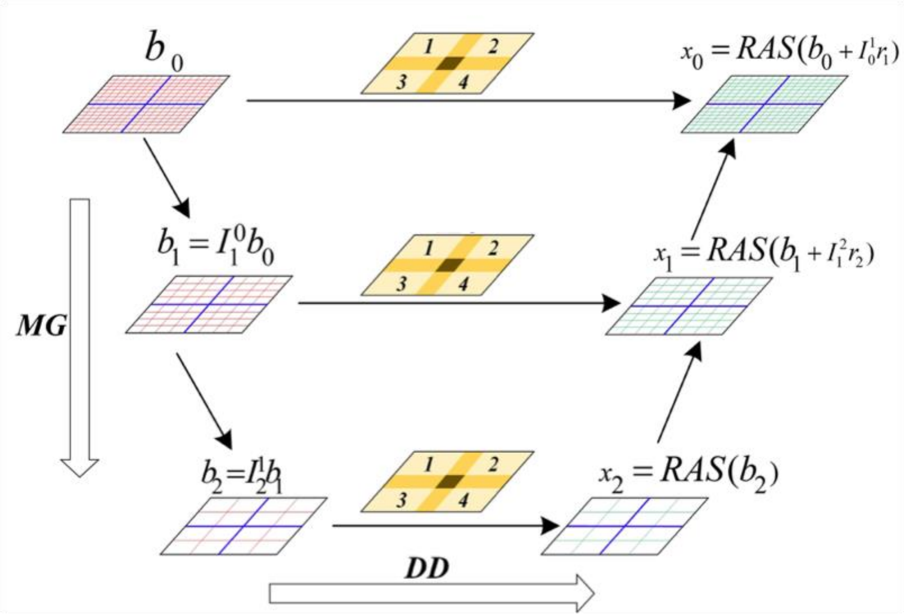
Now let's find a way to design a subdomain solver.

163,840 processes | 65 threads

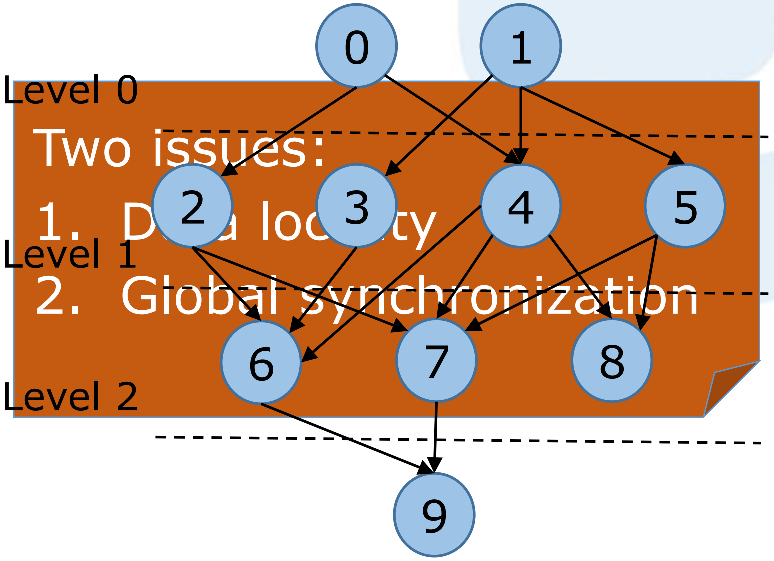
racks × *chips* × *core-groups* × *cores* = *total number of cores*

40 × 1,024 × 4 × 65 = 10,649,600

DD-MG $\mathcal{K}$-cycle



Parallel ILU with level-scheduling

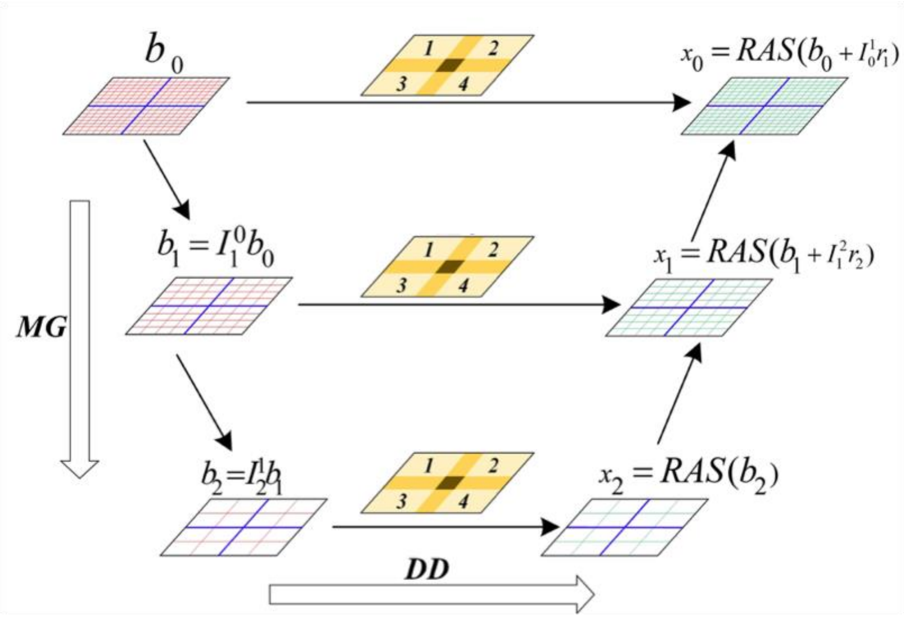


163,840 processes | 65 threads

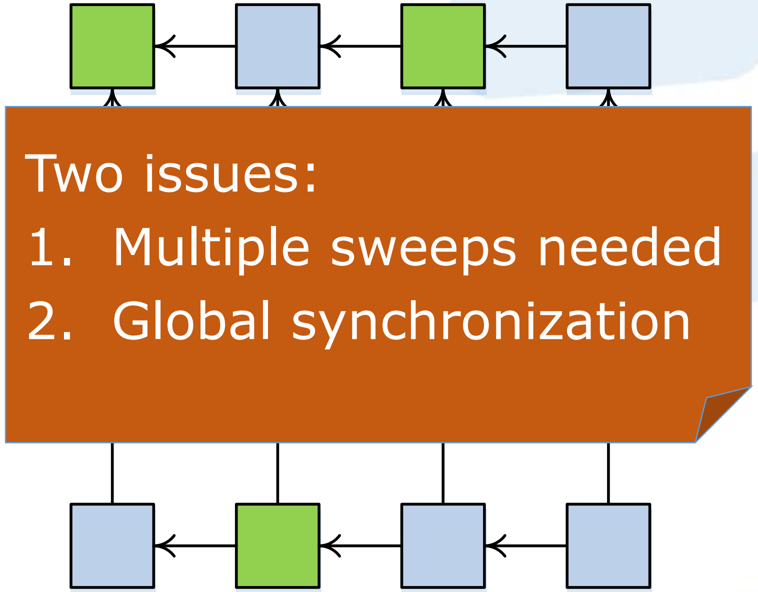
racks × *chips* × *core-groups* × *cores* = *total number of cores*

40 × 1,024 × 4 × 65 = 10,649,600

DD-MG $\mathcal{K}$-cycle



Async. parallel ILU of [Chow et al. SISC'15]



Two issues:

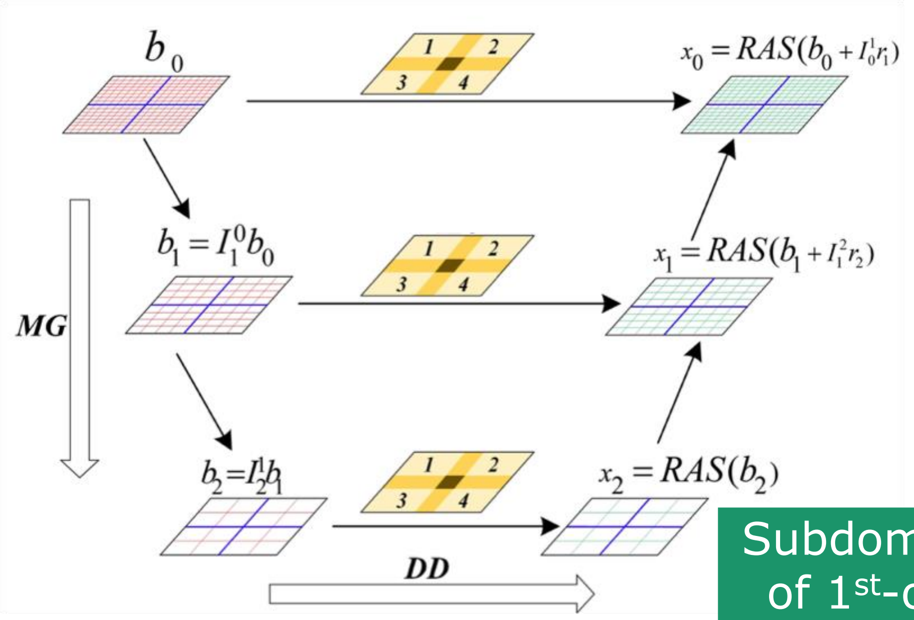
1. Multiple sweeps needed
2. Global synchronization

163,840 processes | 65 threads

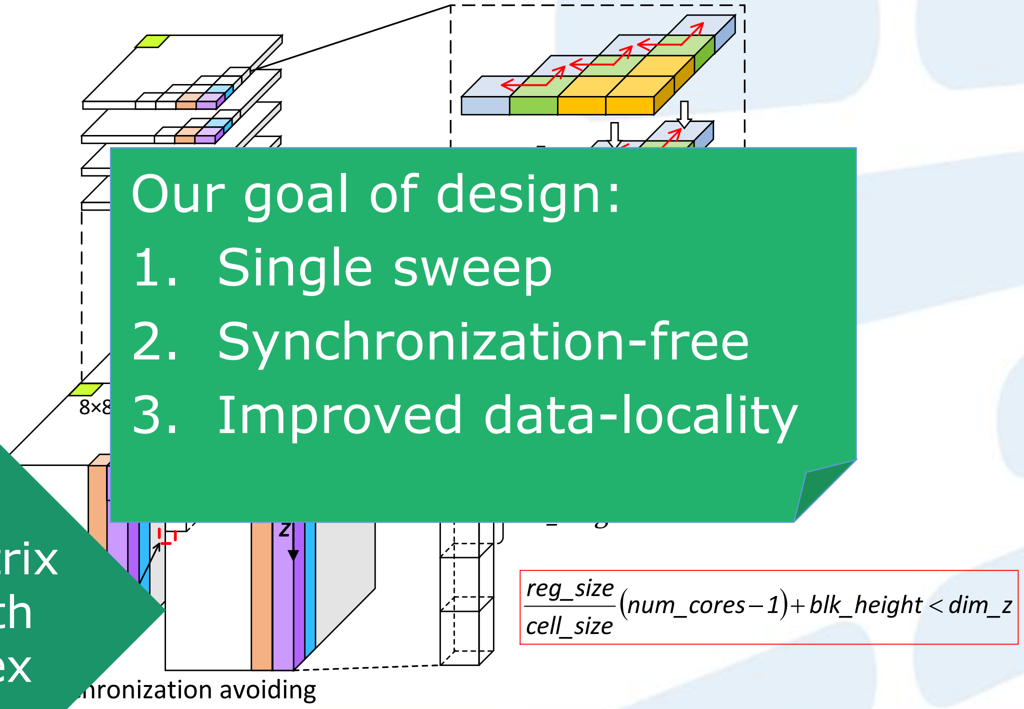
racks × *chips* × *core-groups* × *cores* = *total number of cores*

40 × 1,024 × 4 × 65 = 10,649,600

DD-MG $\mathcal{K}$-cycle



Geometry-based pipelined ILU (GP-ILU)



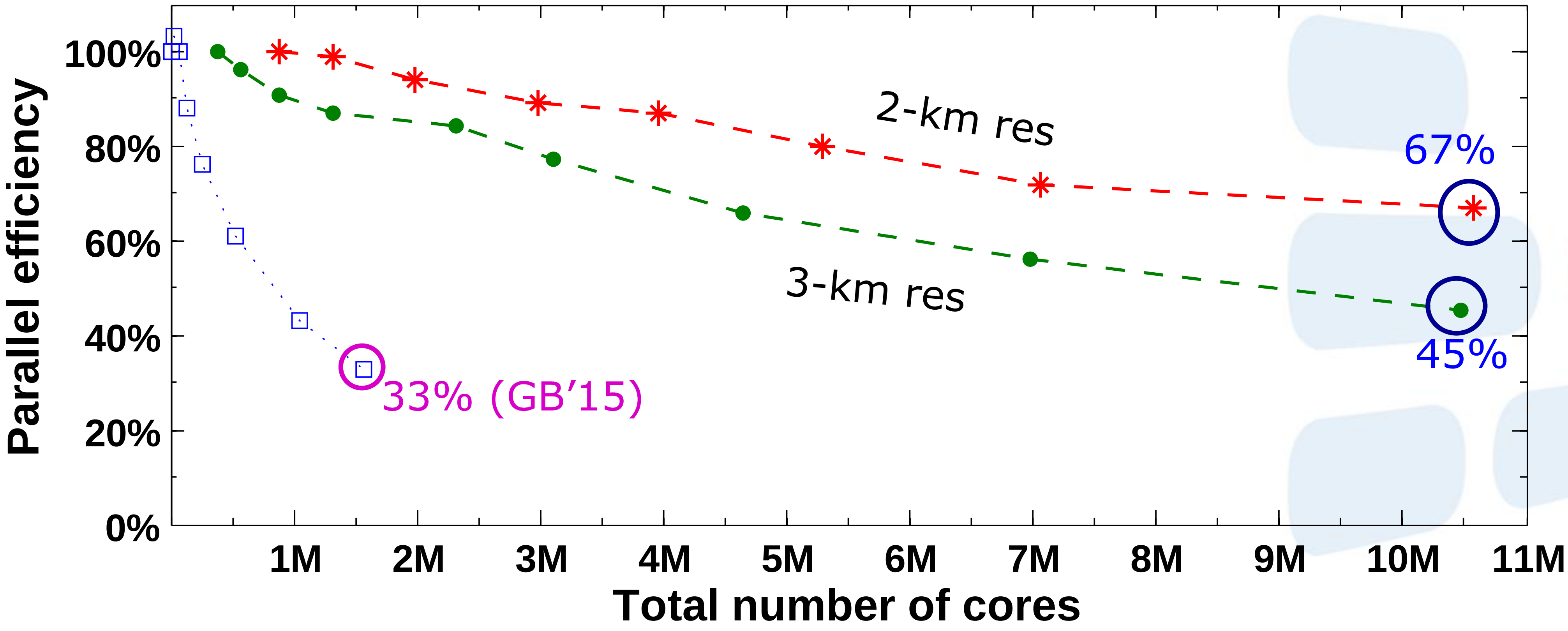
Our goal of design:

1. Single sweep
2. Synchronization-free
3. Improved data-locality

Subdomain matrix of 1st-order with geometric index

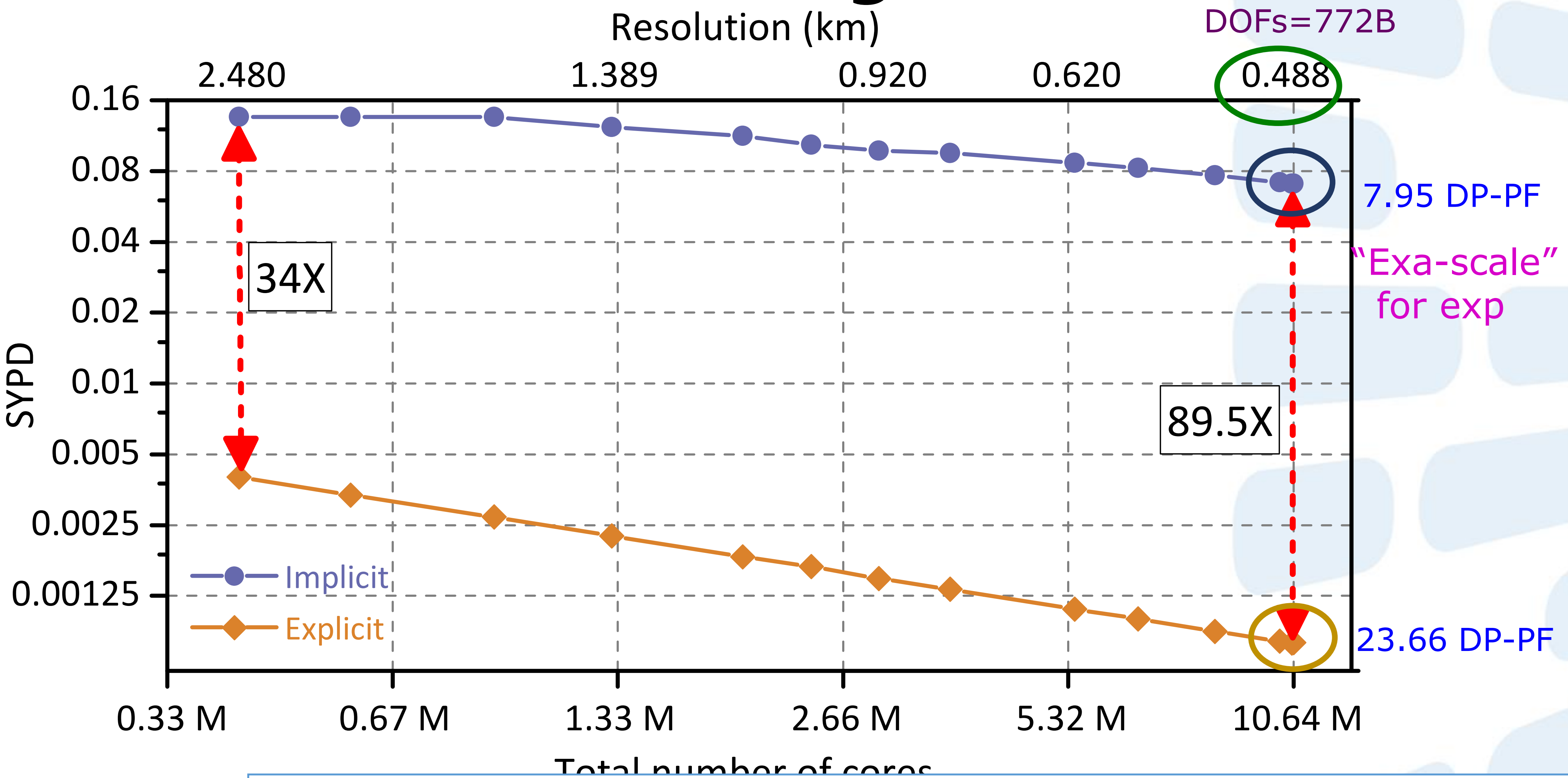
国家超级计算无锡中心
National Supercomputing Center in Wuxi

Strong-scaling results



The 3-km res run: 1.01 SYPD with 10.6M cores, dt=240s, I/O penalty <5%

Weak-scaling results



The 488-m res run: 0.07 SYPD, 10.6M cores, dt=240s, 89.5X speedup over explicit

Our Research Goals

- highly scalable framework that can efficiently utilize many-core processors
- automated tools to deal with the legacy code

- millions lines of legacy code
- poor scalability
- written for m[illegible]e, rather than many-core

100T

100P

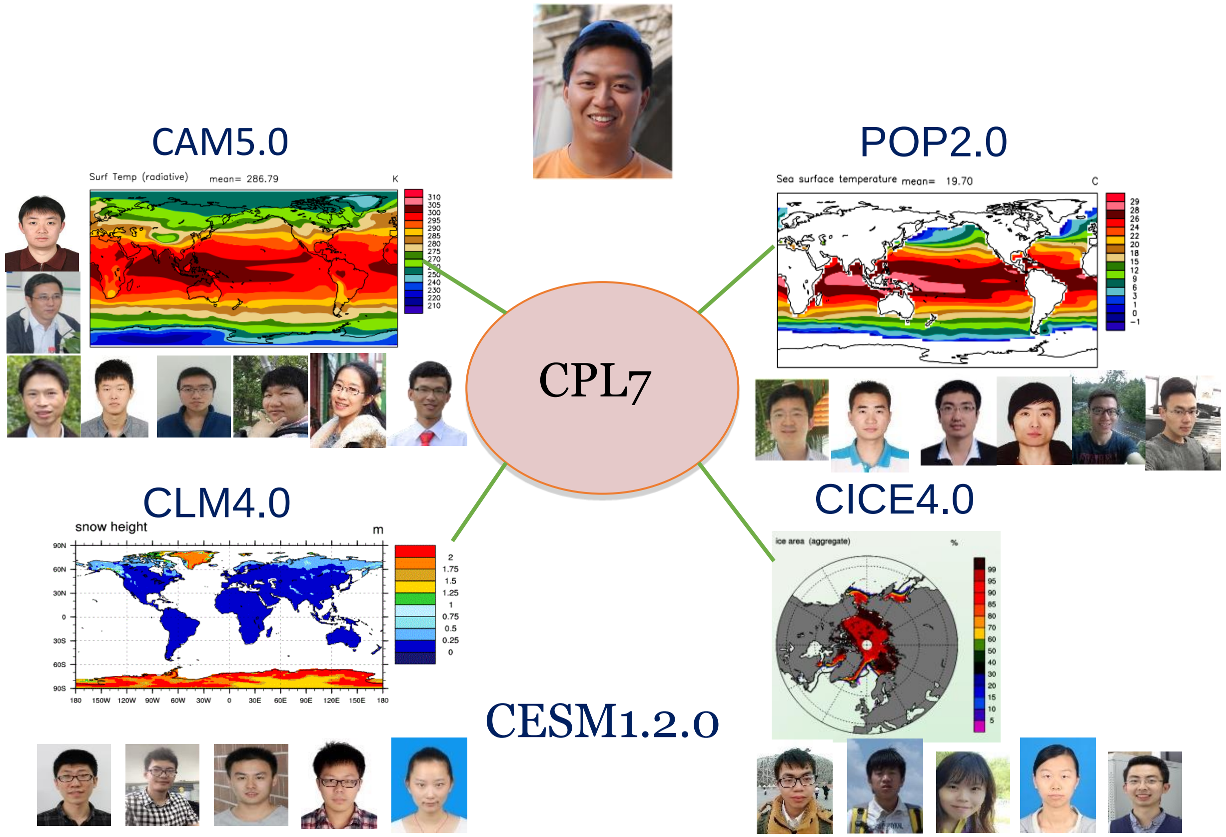
China's models

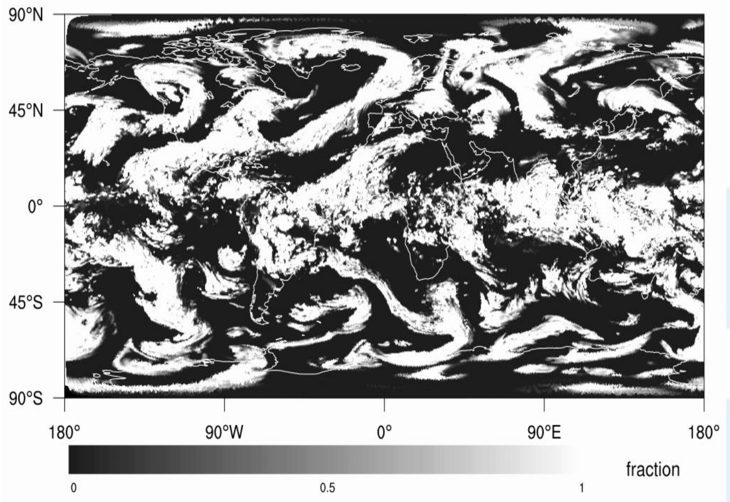
- pure CPU code
- scaling to hundreds or thousands of cores

China's supercomputers

- heterogeneous systems with many-core chips
- millions of cores

The CESM Project on Sunway TaihuLight





Tsinghua + BNU 30+ Professors and Students

- Four component models, millions lines of code
- Large-scale run on Sunway TaihuLight
  - 24,000 MPI processes
  - Over one million cores
- 10-20x speedup for kernels
- 2-3x speedup for the entire model

"Refactoring and Optimizing the Community Atmosphere Model (CAM) on the Sunway TaihuLight Supercomputer", in Proceedings of SC 2016.

Major Challenges

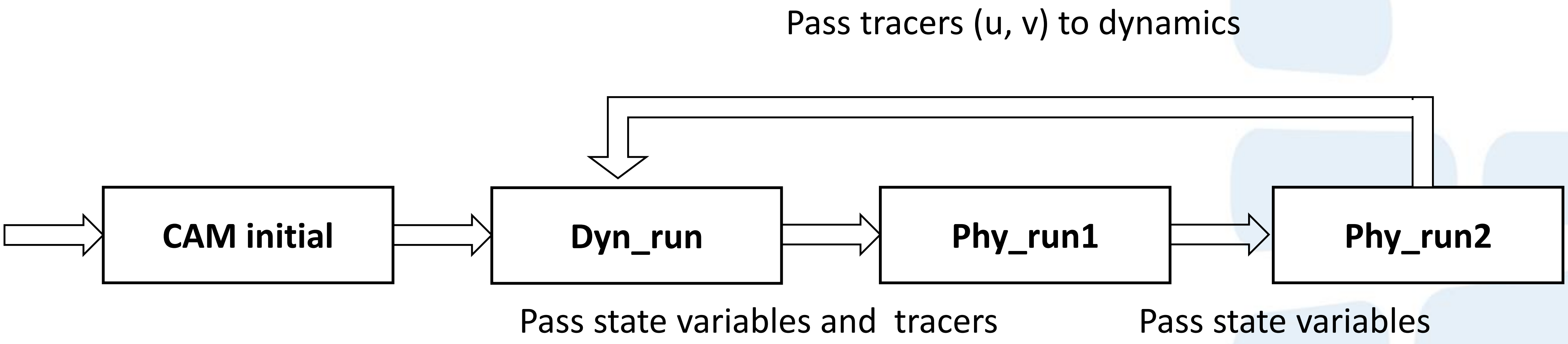
a high complexity in application, and a heavy legacy in the code base (millions lines of code)

an extremely complicated MPMD program with no hotspots (or hundreds of hotspots)

misfit between the in-place design philosophy and the new architecture

lack of people with interdisciplinary knowledge and experience

Workflow of CAM



After initialization, the physics and the dynamics are executed in turn during each simulation time-step.

Porting of CAM: General Idea

- Entire code base: 530, 000 lines of code

- Components with regular code patterns
  - e.g. the CAM-SE dynamic core
  - manual OpenACC parallelization and optimization on code and data structures

- Components with irregular and complex code patterns
  - e.g. the CAM physics schemes
  - loop transformation tool to expose the right level of parallelism and code size
  - memory footprint analysis and reduction tool

Refactoring the Euler Step

Euler_step:

```
do ie = nets, nete
   compute Q min/max values for lim8
   compute Biharmonic mixing term f
end do
```

```
do ie = nets, nete
    2D advection step
    data packing
end do
```

```
Bonundary exchange
```

```
Data extracting
```

1

```
do ie = nets, nete
   do k = 1, nlev
      dp(k) = func_1()
      do q = 1, qsize
         Qtens(k,q,ie) = func_2(dp(k))
      end do
   end do
end do

do ie = nets, nete
   do k = 1, nlev
      do q = 1, qsize
         qmin(k,q,ie) = …
         qmax(k,q,ie) = …
      end do
   end do
end do
```

```
do ie = nets, nete
   do q = 1, qsize
      do k = 1, nlev
         ….
      end do
   end do
end do

do ie = nets, nete
   do k = 1, nlev
      dp0 = func_3()
      dpdiss = func_4()
      do q = 1, qsize
         Qtens(k,q,ie) =   func_5(dp0, dpdiss)
      end do
   end do
end do
```

```
do ie = nets, nete
   do k = 1, nlev
      dp(k) = func_5()
      Vstar(k) = func_6()
   end do

   do q = 1, qsize
      do k = 1, nlev
         Qtens(k,q,ie) = func_7(dp(k), Vstar(k))
       end do
```

```
    do k = 1, nlev
        dp_star(k) = func_8(dp(k))
    end do

     do k = 1, nlev
          Qtens(k,q,ie) = func_9(dp_star(k))
      end do
   end do
   Data packing
end do
```

2

Refactoring the Euler Step

```
do ie = nets, nete
  do k = 1, nlev
    dp(k) = func_1()
    do q = 1, qsize
      Qtens(k,q,ie) = func_2(dp(k))
    end do
  end do
end do

do ie = nets, nete
  do k = 1, nlev
    do q = 1, qsize
      qmin(k,q,ie) = …
      qmax(k,q,ie) = …
    end do
  end do
end do

do ie = nets, nete
  do q = 1, qsize
    do k = 1, nlev
      ….
    end do
  end do
end do

do ie = nets, nete
  do k = 1, nlev
    dp0 = func_3()
    dpdiss = func_4()
    do q = 1, qsize
      Qtens(k,q,ie) =
            func_5(dp0,
dpdiss)
    end do
  end do
```

```
do ie = nets, nete
  do k = 1, nlev
    dp(k) = func_5()
    Vstar(k) = func_6()
  end do

  do q = 1, qsize
    do k = 1, nlev
      Qtens(k,q,ie) =
   func_7(dp(k), Vstar(k))
     end do
```

```
    do k = 1, nlev
      dp_star(k) = func_8(dp(k))
    end do

    do k = 1, nlev
      Qtens(k,q,ie) =
           func_9(dp_star(k))
    end do
  end do
  Data packing
end do
```

2

```
do ie = nets, nete
  do k = 1, nlev
    do q = 1, qsize
      Qtens(k,q,ie) =
     func_2(func_1())
    end do
  end do
end do

do ie = nets, nete
  do k = 1, nlev
    do q = 1, qsize
      qmin(k,q,ie) = …
      qmax(k,q,ie) = …
    end do
  end do
end do

do ie = nets, nete
  do q = 1, qsize
    do k = 1, nlev
      ….
    end do
  end do
end do

do ie = nets, nete
  do k = 1, nlev
    do q = 1, qsize
      Qtens(k,q,ie) =
           func_5(func_3(),func_4())
    end do
  end do
end do
```

```
do ie = nets, nete
  do q = 1, qsize
    do k = 1, nlev
      Qtens(k,q,ie) =
            func_7(func_5(),func_6())
     end do
```

```
    do k = 1, nlev
      Qtens(k,q,ie) =

func_9(func_8(func_5()))
    end do
  end do
  Data packing
end do
```

3

Refactoring the Euler Step

```
do ie = nets, nete
   do k = 1, nlev
      do q = 1, qsize
         Qtens(k,q,ie) =
         func_2(func_1())
      end do
   end do
end do

do ie = nets, nete
   do k = 1, nlev
      do q = 1, qsize
         qmin(k,q,ie) = …
         qmax(k,q,ie) = …
      end do
   end do
end do
```

```
do ie = nets, nete
   do q = 1, qsize
      do k = 1, nlev
         ….
      end do
   end do
end do

do ie = nets, nete
   do k = 1, nlev
      do q = 1, qsize
         Qtens(k,q,ie) =
                func_5(func_3(),func_4())
      end do
   end do
end do
```

```
do ie = nets, nete
   do q = 1, qsize
      do k = 1, nlev
         Qtens(k,q,ie) =
              func_7(func_5(),func_6())
       end do
```

```
      do k = 1, nlev
         Qtens(k,q,ie) =
              func_9(func_8(func_5()))
      end do
   end do
   Data packing
end do
```

3

```
do ie = nets, nete
   do k = 1, nlev
      do q = 1, qsize
         qmin(k,q,ie) = …
         qmax(k,q,ie) = …
         Qtens(k,q,ie) = …
      end do
   end do
end do
```

```
do ie = nets, nete
   do k = 1, nlev
      do q = 1, qsize
         Qtens(k,q,ie) = …
       end do
   end do
end do
Data packing
```

4

```
do ie = nets, nete
   do q = 1, qsize
      do k = 1, nlev
         qmin(k,q,ie) = …
         qmax(k,q,ie) = …
         Qtens(k,q,ie) = …
      end do
   end do
end do
```

```
do ie = nets, nete
   do q = 1, qsize
      do k = 1, nlev
         Qtens(k,q,ie) = …
       end do
   end do
end do
Data packing
```

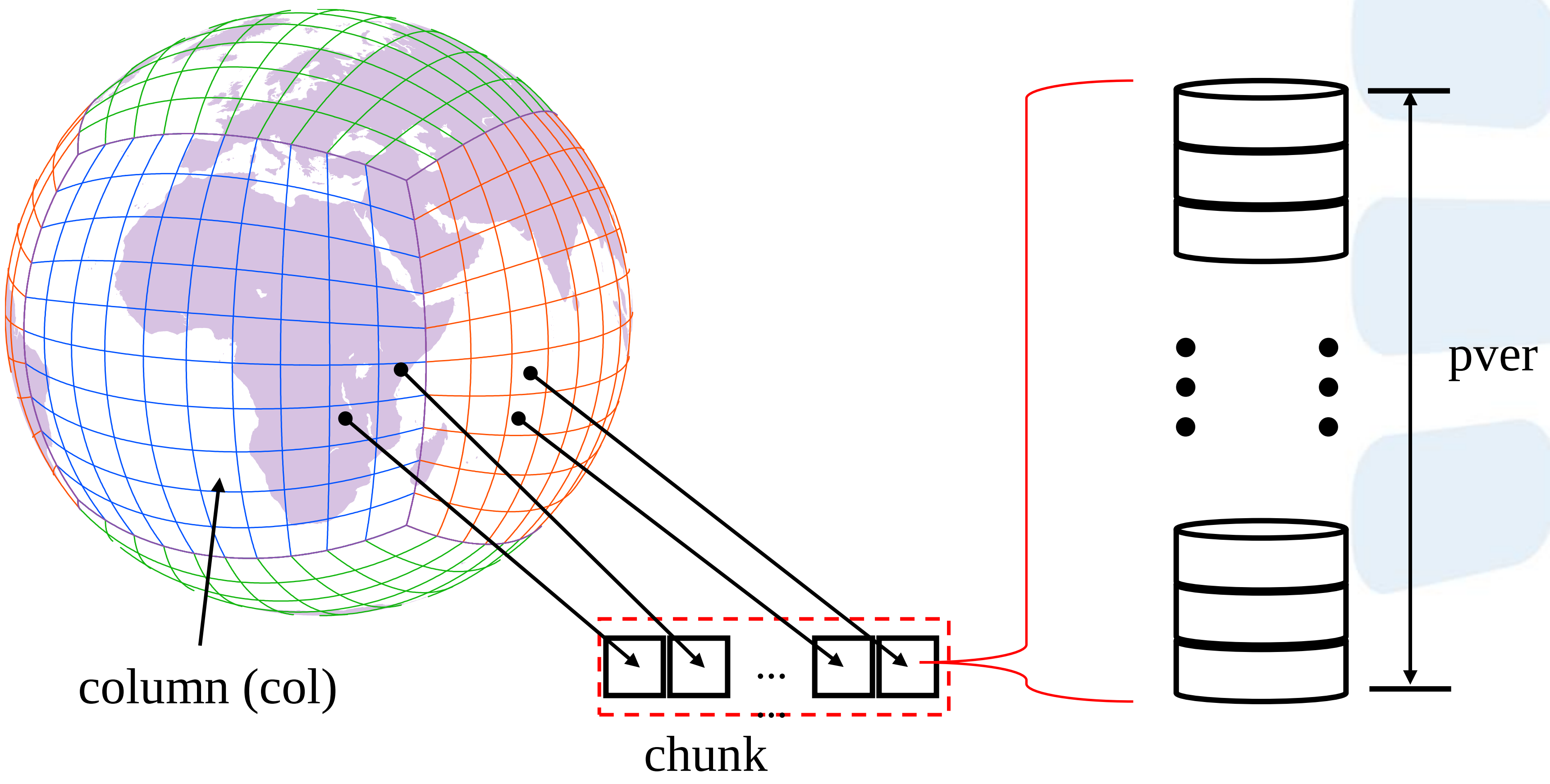
5

```
!$ACC PARALLEL LOOP
do ie_q = 1, qsize*(nete-nets)
   do k = 1, nlev
      q = func(ie_q)
      ie = func(ie_q)
      qmin(k,q,ie) = …
      qmax(k,q,ie) = …
      Qtens(k,q,ie) = …
   end do
end do
```

```
!$ACC PARALLEL LOOP
do ie_q = 1, qsize*(nete-nets)
   do k = 1, nlev
      q = func(ie_q)
      ie = func(ie_q)
      Qtens(k,q,ie) = …
   end do
end do
!$ACC PARALLEL LOOP
Data packing
```

6

Refactoring of the Physics Schemes



Loop Transformation for Phys_run1

```
do begin_chunk to end_chunk
  tphysbc()
  {
    convect_deep_tend(6.47%)
    convect_shallow_tend(15.57%)
    macrop_driver_tend(8.38%)
    microp_aero_run(4.29%)
    microp_driver_tend(7.13%)
    aerosol_wet_intr(4.29%)
    convect_deep_tend_2(0.51%)
    radiation_tend(54.07%)
  }
enddo
```
1

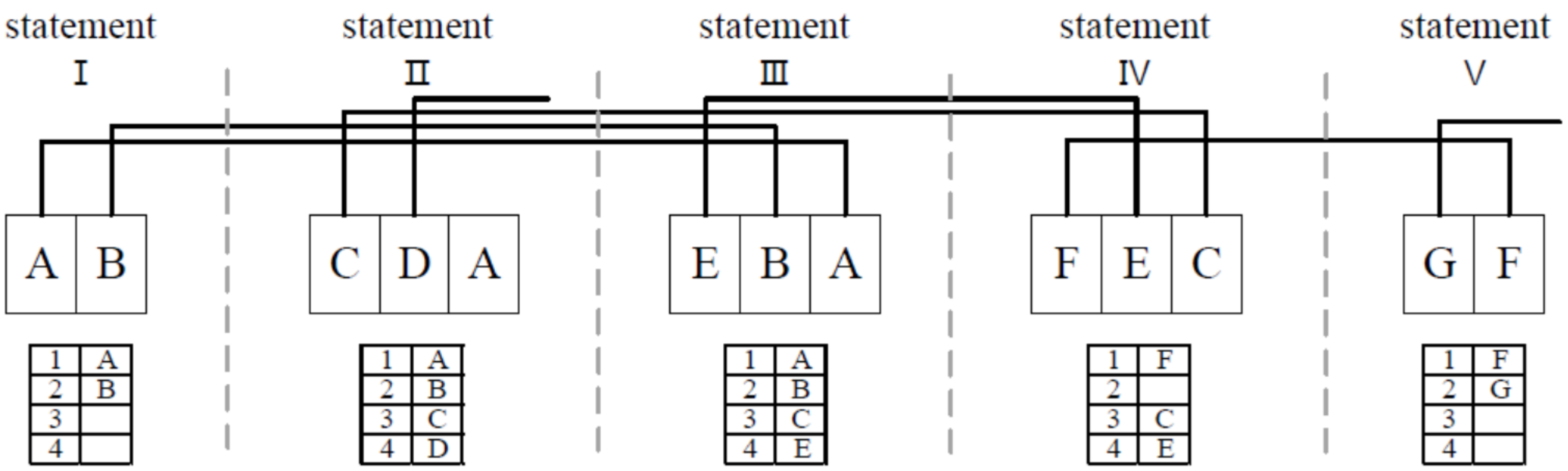
```
tphysbc()
{
  do begin_chunk to end_chunk
    convect_deep_tend(6.47%)
    convect_shallow_tend(15.57%)
    macrop_driver_tend(8.38%)
    microp_aero_run(4.29%)
    microp_driver_tend(7.13%)
    aerosol_wet_intr(4.29%)
    convect_deep_tend_2(0.51%)
    radiation_tend(54.07%)
  enddo
}
```
2

```
tphysbc()
{
  do begin_chunk to end_chunk
    convect_deep_tend(6.47%)
  enddo
  ……
  do begin_chunk to end_chunk
    microp_driver_tend(7.13%)
  enddo
  ……
  do begin_chunk to end_chunk
    radiation_tend(54.07%)
  enddo
}
```
3

```
do begin_chunk to end_chunk
  convect_deep_tend(6.47%)
  {
    zm_conv_tend(6.47%)
    {
      zm_convr(2.03%)
      zm_conv_evap()
      montran()
      convtranc(0.06%)
    }
  }
enddo
```
4

```
convect_deep_tend(6.47%)
{
  zm_conv_tend(6.47%)
  {
    do begin_chunk to end_chunk
      zm_convr(2.03%)
    enddo
    do begin_chunk to end_chunk
      zm_conv_evap()
    enddo
    do begin_chunk to end_chunk
      montran()
    enddo
    do begin_chunk to end_chunk
      convtranc(0.06%)
    enddo
  }
}
```
5

Variable Storage Space Analysis and Reduction Tool



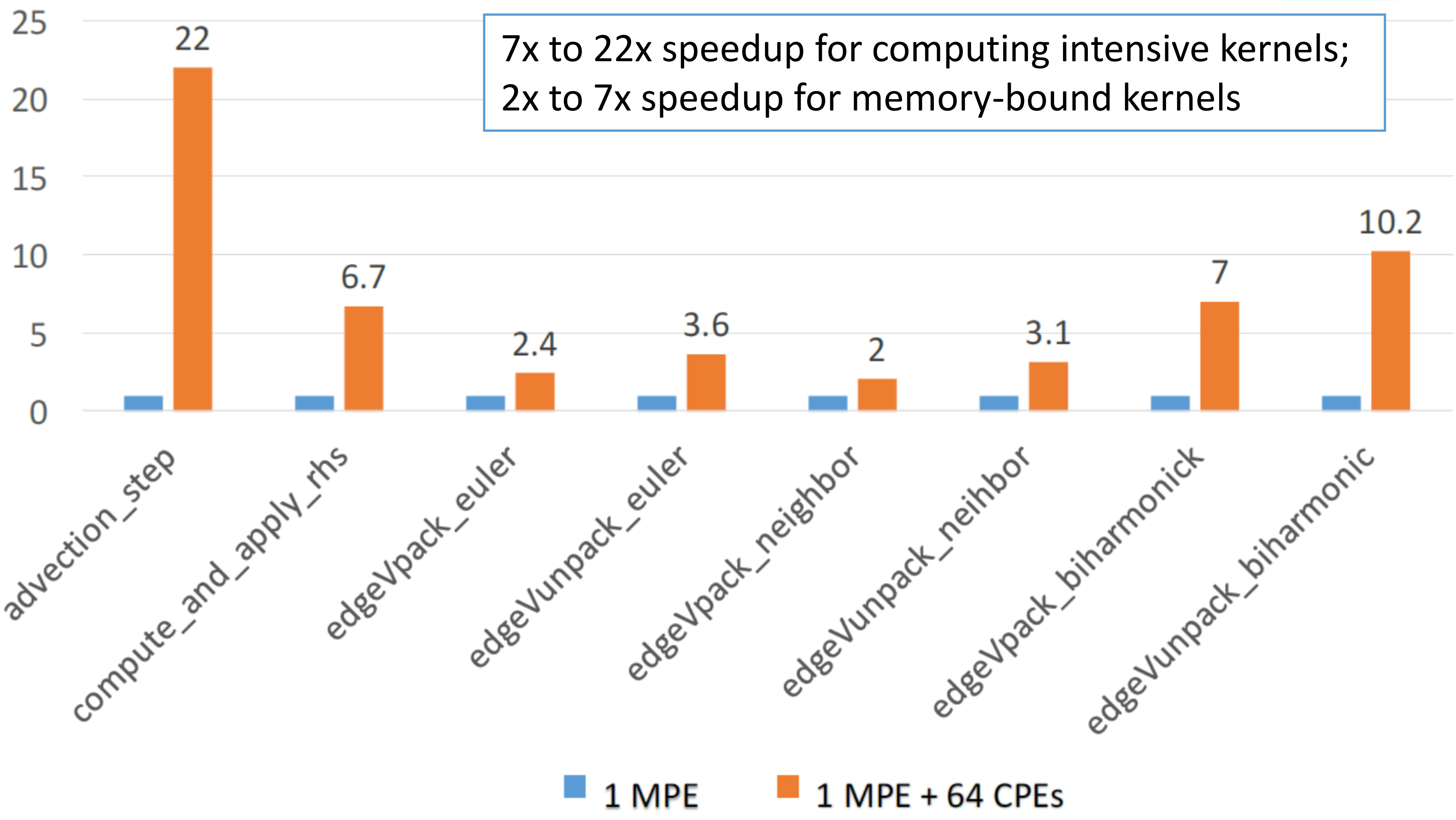
Basic functions

- Estimate the storage requirements of the variable and arrays
- Identify the lifespan of the variables and arrays
- Determine whether the variables and arrays of each CPE thread can fit into the 64KB SPM.

Example Explanation

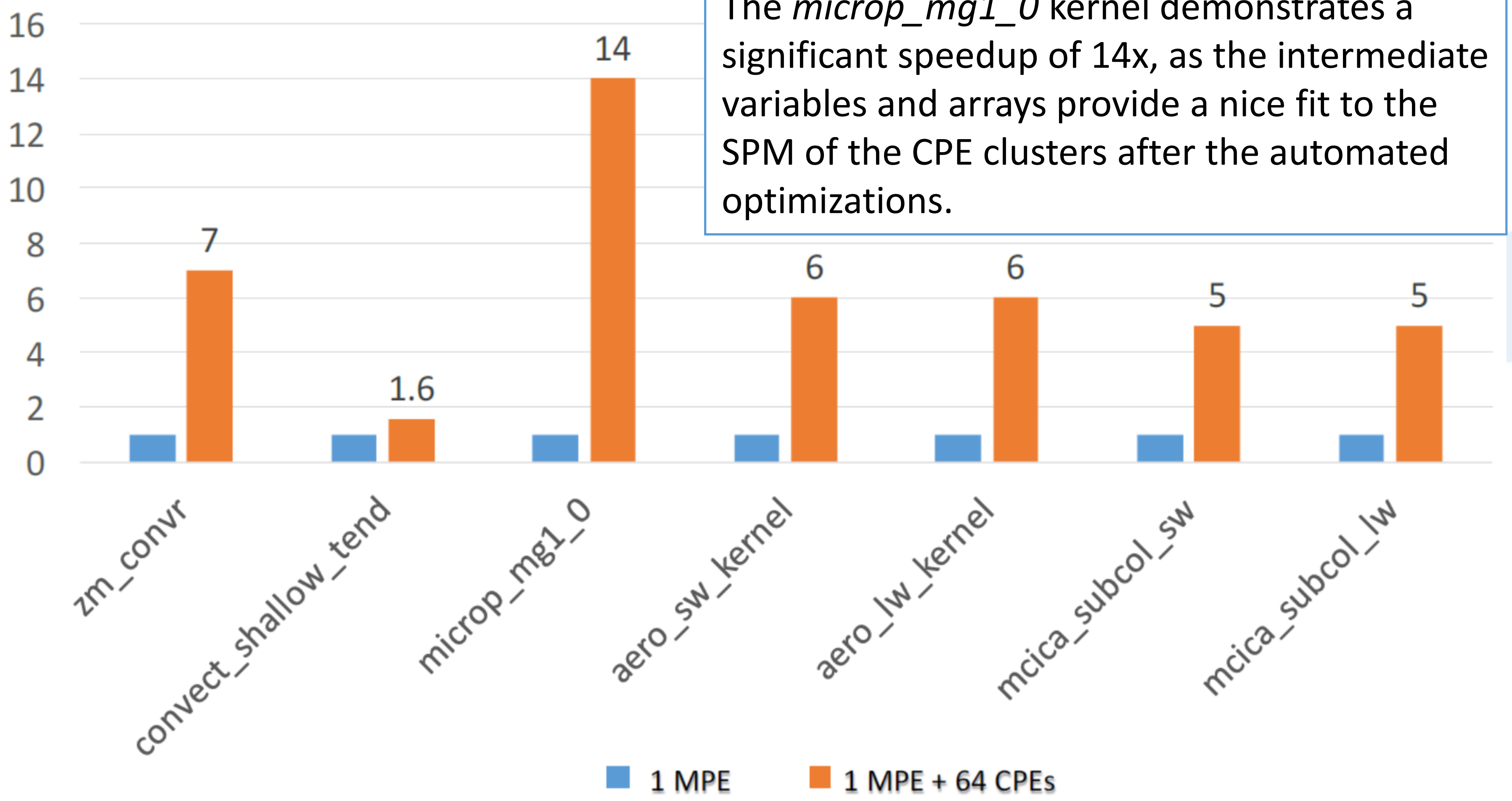
- The original Fortran function accesses 7 intermediate arrays (A to G) during the computation process. By analyzing the lifespan of these 7 arrays, which are annotated by the lines above these arrays, we can determine that 4 arrays would provide sufficient space to store these 7 arrays in different stages of the execution process.

Speedup of Major Kernels in CAM-SE



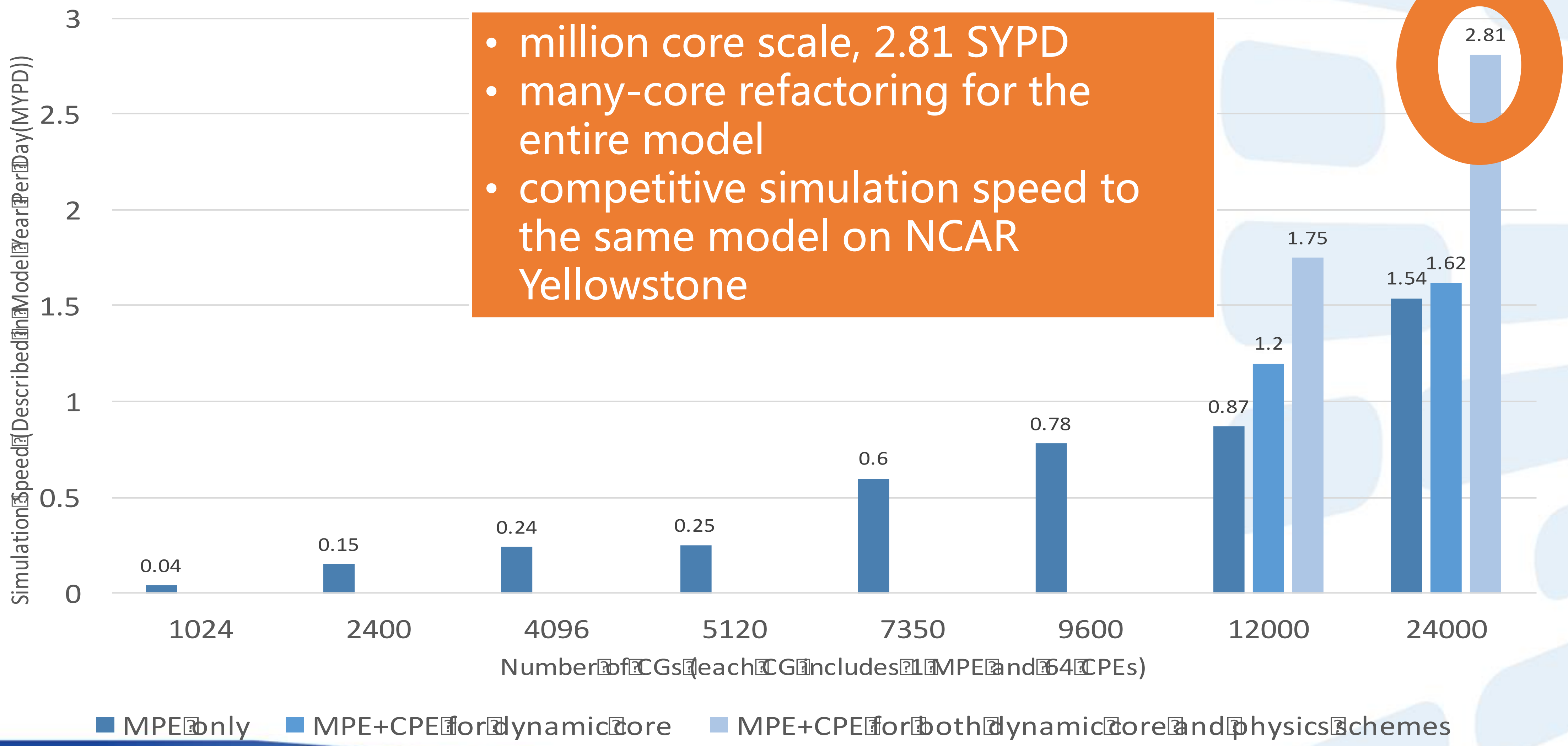
Speedup of Major Kernels in CAM-PHY

The *microp_mg1_0* kernel demonstrates a significant speedup of 14x, as the intermediate variables and arrays provide a nice fit to the SPM of the CPE clusters after the automated optimizations.



CAM model: scalability and speedup

- million core scale, 2.81 SYPD
- many-core refactoring for the entire model
- competitive simulation speed to the same model on NCAR Yellowstone



Library for Deep Learning (swDNN)

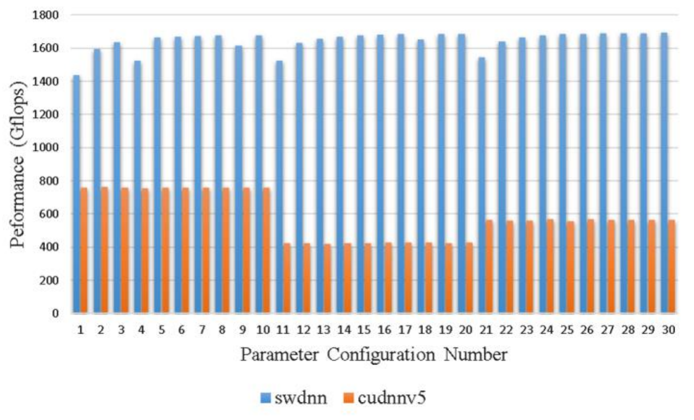
- swDNN: Provide interface for optimized basic operators
  - Fully-connected layer (BLAS); Pooling layer
  - Activation function; Batch Normalization
  - ***Convolutional Layer(90% time for CNN)**

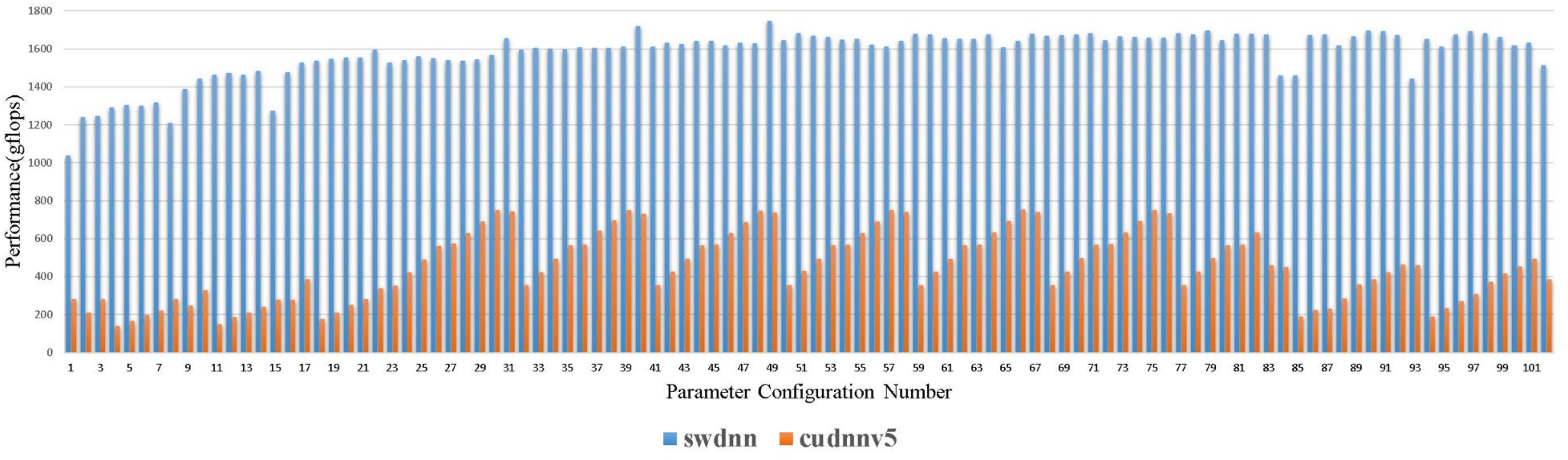
Related Works on other architectures

Work	Platform	Method
cuDNN(2014)	GPU	GEMM
fbtfft(2014)	GPU	FFT
Andrew Lavin (2015)	GPU	Winograd
Chen Zhang (2015)	FPGA	Direct Conv
swDNN	SW26010	Blocking GEMM

Library for Deep Learning (swDNN)

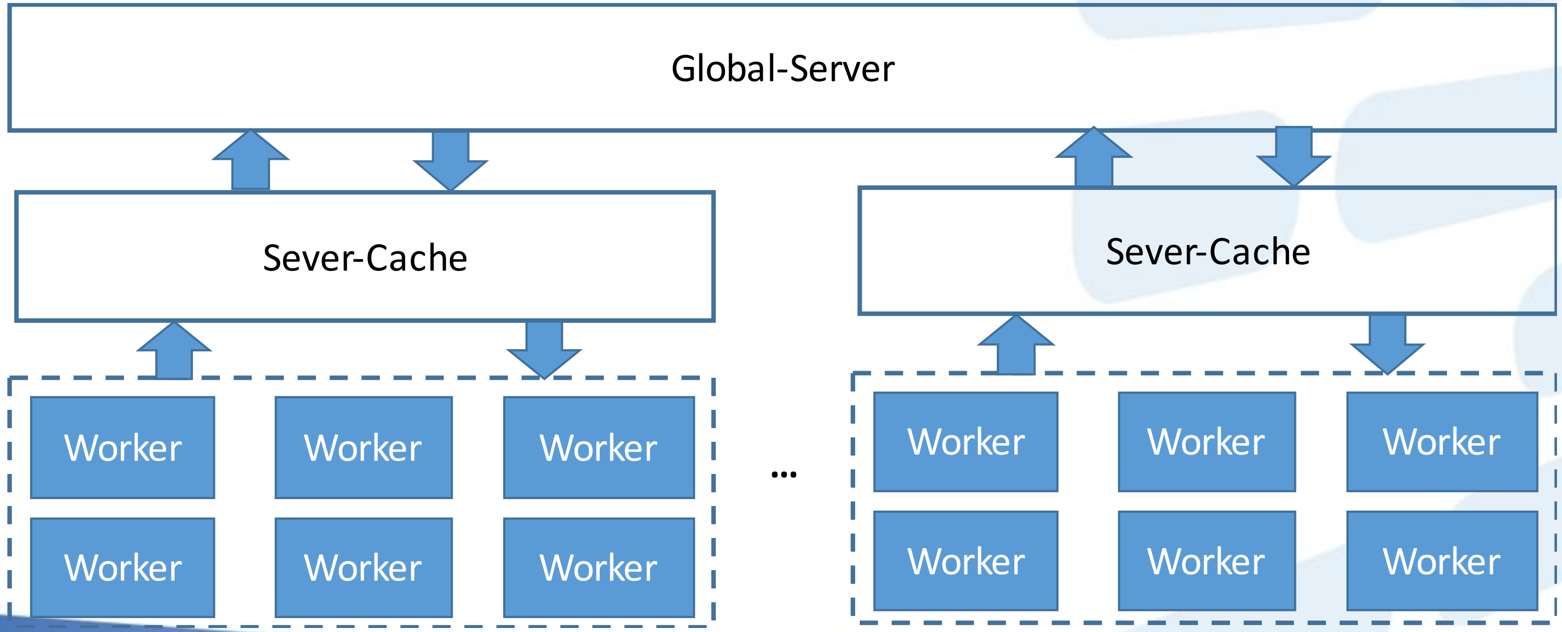
- **Performance**
  - Convolutional performance above **1.6 Tflops** with double-precision
  - Speedup ranging from **1.91x to 9.75x** compared with cudnnv5.1.



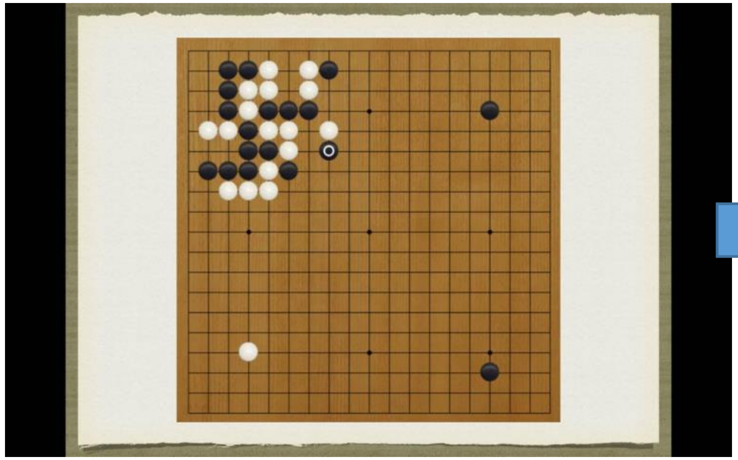


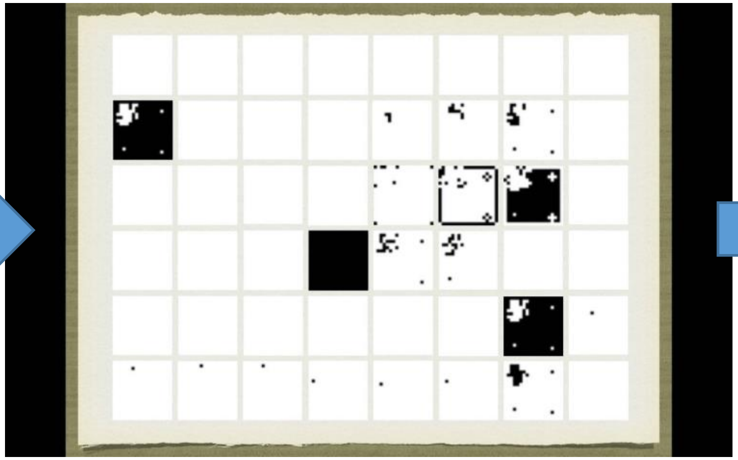
Framework for Deep Learning (under development)

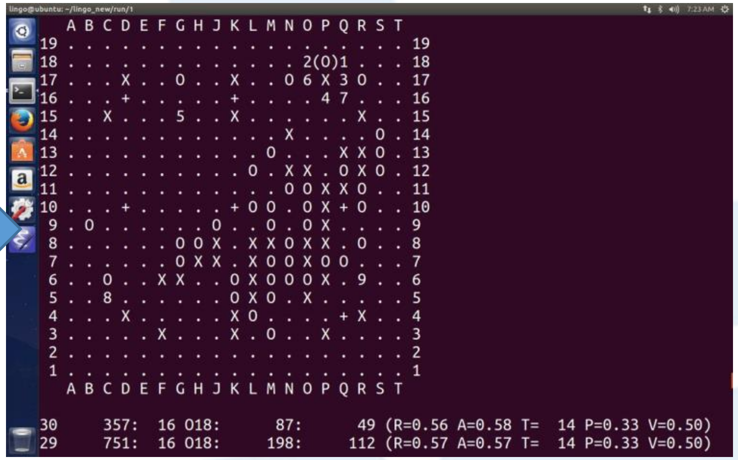
- **Distributed framework**
  - Customized from *Caffe* with less dependencies
  - Two-level Parameter Server Based-on MPI



swDNN Supported Project: Sunway-Lingo collaborated with Prof. Zhiqing Liu, BUPT







- Original go board to be processed
- Converted to a 48-channel image fed to deep CNN with essential go features such as liberties
- Order of probabilities of plausible moves as outputted by policy network

Long Term Plan

- Traditional HPC Applications (Science -> Service)
  - weather / climate service
  - seismic data processing service
  - CFD simulation framework for Advanced Manufacturing

- Deep Learning Related Applications
  - the swDNN framework
  - collaborating with face++ for face recognition applications
  - collaborating with Sogou for voice recognition and translation
  - customized DNN Sunway chip?

- Big Data Center
  - National Health and Medical Big Data Center at Nanjing

THANK YOU