Empowering Cloud-Resolving Models Through GPU and Asynchronous I/O

The 2014 Earth Science Technology Forum (ESTF2014)

Wei-Kuo TAO (PI)
Thomas L. Clune (Co-PI)
Shujia Zhou (Co-PI)
Toshihisa Matsui (Co-I)
Xiaowen Li (Co-I)
Xiping Zeng (Co-I)

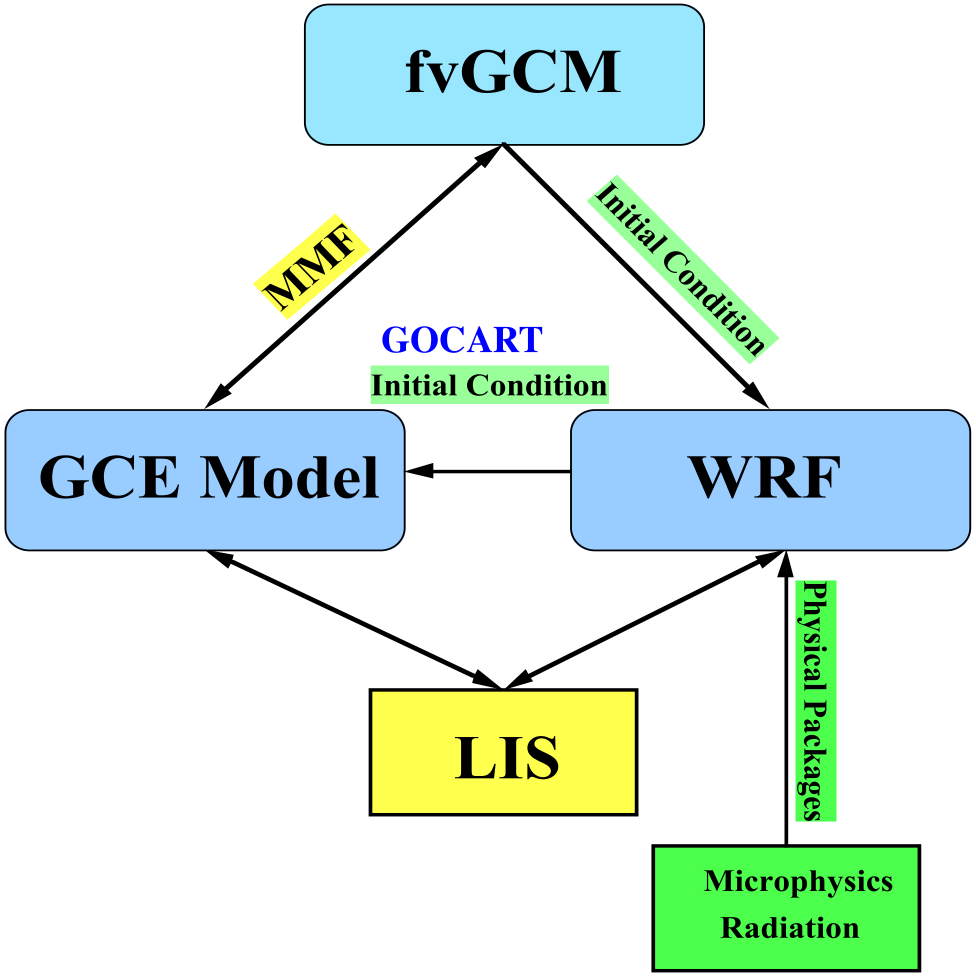
Earth Science Technology Office

Background

NASA Cloud Resolving Models

Recently, a multi-scale modeling system with unified physics was developed at NASA Goddard. It consists of **(1) the Goddard Cumulus Ensemble model (GCE), a cloud-resolving model (CRM), (2) the NASA unified Weather Research and Forecasting Model (WRF), a region-scale model, and (3) the coupled fvGCM-GCE, the GCE coupled to a general circulation model (or GCM known as the Goddard Multi-scale Modeling Framework or MMF).** The same cloud microphysical processes, long- and short-wave radiative transfer and land-surface processes are applied in all of the models to study explicit cloud-radiation and cloud-surface interactive processes in this multi-scale modeling system. This modeling system has been coupled with a multi-satellite simulator for comparison and validation with NASA high-resolution satellite data. The left figure shows the multi-scale modeling system with unified physics. The GCE and WRF share the same microphysical and radiative transfer processes (including the cloud-interaction) and land information system (LIS). The same GCE physics will also be utilized in the Goddard MMF.

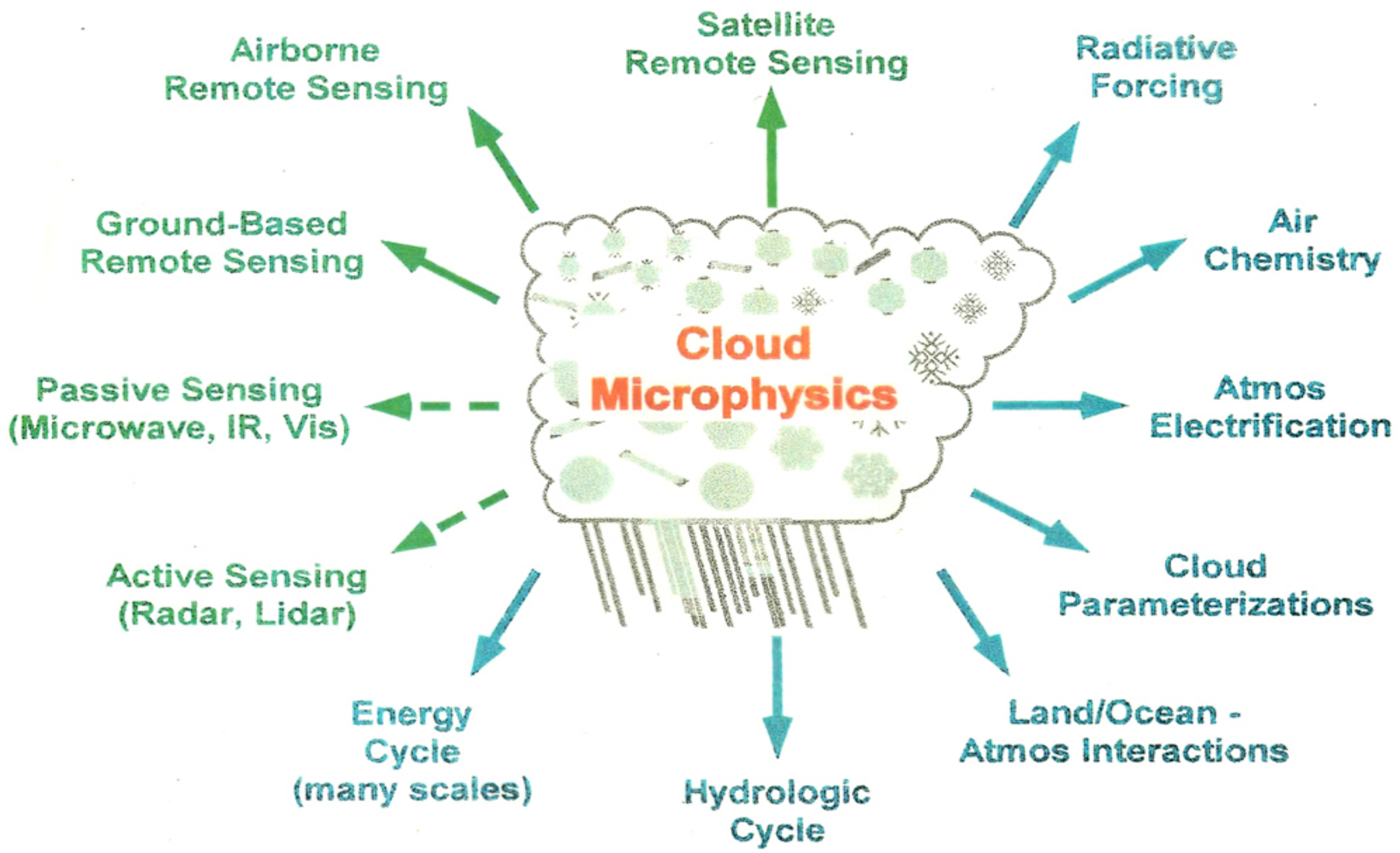
The idea to have a multi-scale modeling system with unified physics is to be able to propagate improvements made to a physical process in one component into other components smoothly and efficiently.



MMF: Multi-Scale Modeling Framework
GCE: Goddard Cumulus Ensemble Model
WRF: Weather Research Forecast
LIS: Land Information System
GOCART: Goddard Chemistry Aerosol Radiation and Transport Model

Tao, W.-K., D. Anderson, J. Chern, J. Entin, A. Hou, P. Houser, R. Kakar, S. Lang, W. Lau, C. Peters-Lidard, X. Li, T. Matsui, M. Rienecker, M. R. Schoeberl B.-W. Shen, J.-J. Shi, and X. Zeng, 2009: Goddard Multi-Scale Modeling Systems with Unified Physics, *Annales Geophysics*, **27**, 3055-3064.

CLOUD MICROPHYSICS IN EARTH SYSTEM SCIENCE



Weather and climate models are using explicit microphysics schemes developed by CRM for their higher resolution forecast/simulation

CPU for radiation and microphysics

	Total CPU hours	Total CPU number	Radiation	Microphysics	Dynamics
Bulk (radiation every 10 steps)	198	64	25.1%	10.0%	63.9%
Bulk (radiation every step)	658	64	77.2%	3.0%	19.5%
Bin (radiation every 10 step)	44,697	1024	0.12%	45.8%	54.1%
Bin (radiation every step)	47,138	1024	1.16%	45.8%	53.0%

CPU times for 3D GCE simulations for a convective case on the NASA Pleiades computer. The domain size is 256x256x41, total integration time is 24 hours with 3 seconds time step. Dynamics includes the advection of all variables as well as the pressure solver.

Bin scheme cost about x 326 CPU time compared with 1-M bulk run

I/O data requirements Microphysical Scheme

Estimations based on the domain size of 256 x 256 x 41 grid points, for a total 5-days integration time, using FORTRAN binary format

	single output	current output frequency	current data amount	desired output frequency	desired data amount
Dynamics	0.12 G	1 hr	14.4 G	5 min	0.17 T
Bulk microphysics	0.15 G	1 hr	18.0 G	5 min	0.22 T
Bin microphysics	5.4 G	1 hr	648 G	5 min	7.78 T
Statistics	0.4 G	simulation period	0.4 G	simulation period	0.4 G
Total	18 G	------	680.8 G	------	8.17 T

Goddard MMF: 5 TBs for 1 year run with hourly CRM output and 45% wall time for output

Project Goals

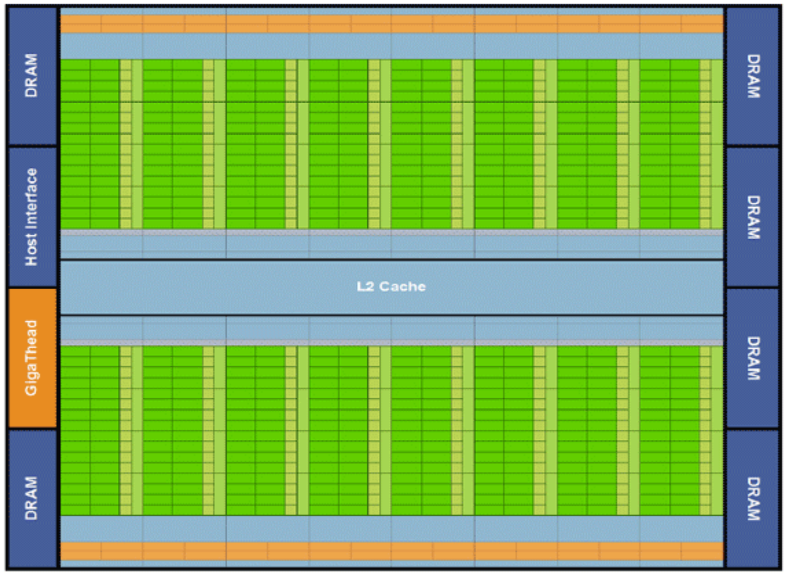
- Improve NASA cloud resolving models' computational performance by porting computationally-intensive components (Radiation and Microphysics) to Graphics Processing Units (GPUs)
- Develop an Asynchronous I/O tool to offload output data from compute node to reduce the idle time of computing processors
- Develop a data compression mechanism to further empower the Asynchronous I/O tool

Rationale for Reengineering: **Preparation for Future Supercomputers**

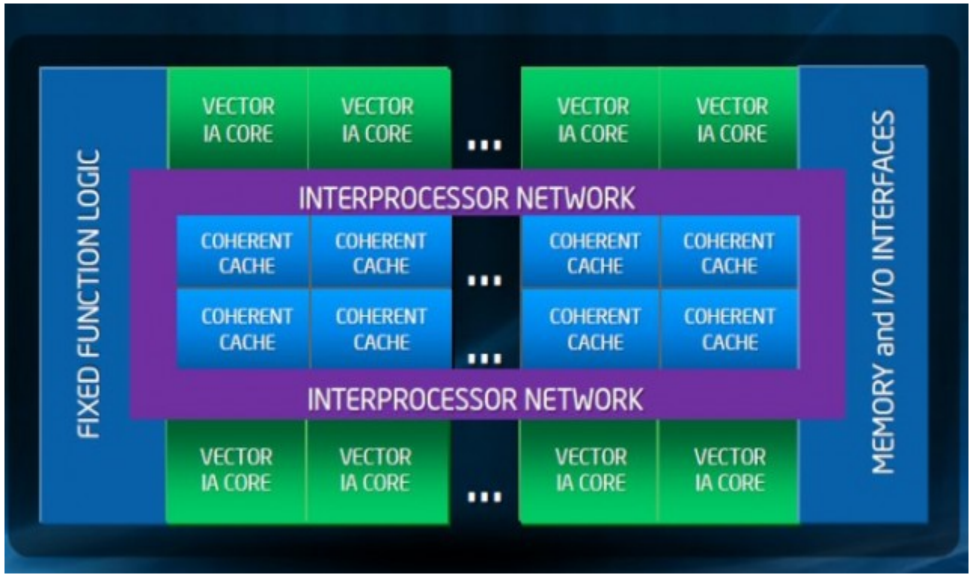
- Essentially all next-generation supercomputers are using HW accelerators (using SIMD or Vector processors) to reduce energy consumption
  - Several with NVIDIA GPUs and Intel Xeon Phi in Top500 Supercomputers as of November 2013
    - Tianhe-2 (No. 1) using Intel Xeon Phi
    - ORNL Titan (No. 2) using GPU K20x
  - NASA NCCS and NAS have installed MIC in its supercomputer and are expected to make further investments in upcoming procurements.
- NASA NCCS has encountered the constraint of energy consumption
  - Part of intel processors run in a low energy consumption mode

Strategy for Reengineering GCE/Nu-WRF: Current Accelerator Technologies

- NVIDIA Graphics Processing Units (GPUs)
  - Fermi
  - Kepler
- Intel Many Integrated Core Architecture (MIC)
  - Knights Corner with more than 50 cores per chip

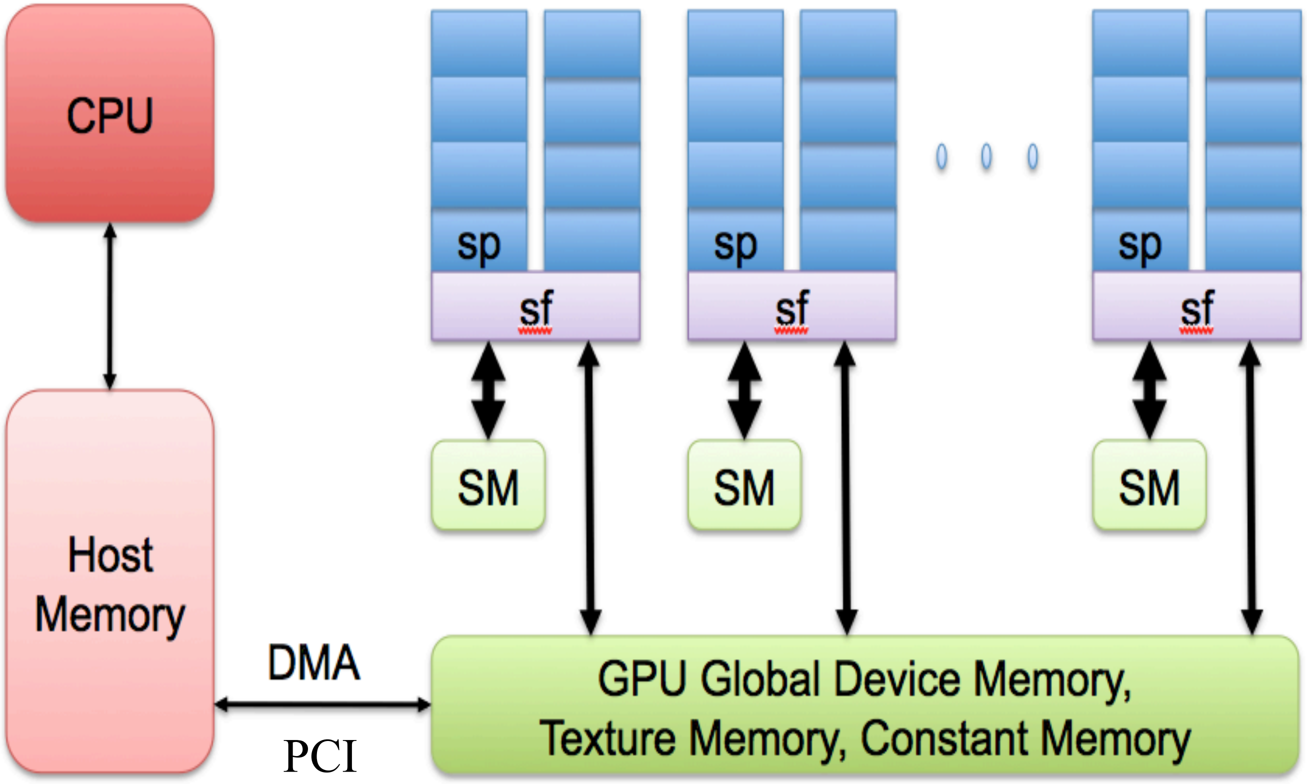


Fermi's 16 SM are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).



For GPUs some real-world applications have achieved good performance with considerable code changes. For MIC, it is reported for relative easier adoption. However, considerable code changes are still required to achieve good performance.

Porting an application to GPU with optimal performance involves three major memory systems: CPU host memory, GPU global device memory, and GPU local shared memory



Major components involved in GPU computing. Data transfers between CPU's host memory and GPU's global device memory through Direct-Memory Access (DMA). Each stream processor (sp) accesses data from local shared memory (SM), global device memory, texture memory, and constant memory. A special function unit (transcendental: sin, pow, etc.) is represented with sf.

Strategy for Reengineering GCE/NuWRF Current Programming Paradigms

- **NVIDIA CUDA**
  - Many-thread and multiple-memory programming model
    - The more performance, the more code changing
- **OpenACC**
  - Developed to ease programming for accelerators in a similar style to OpenMP
  - The current compiler 1.x is not mature. The 2.0 version has been released lately.
- We are using Portland Group (PGI) CUDA Fortran to improve the performance
  - Have limited options in reducing copy-in-and-out costs
    - Use code reorganization to reduce the costs in this project

Strategy for Reengineering GCE/NU-WRF: Current Solution

- Reduce copy-in-and-out cost
  - Put reusable variables onto GPU through Fortran CUDA module
  - Fuse identical codes to reduce temporary variables
    - Soluv() and Solir()
  - Move the column index out of kernels and into drivers
    - More code changes

Port Radiation, One-moment Physics, Two-moment Physics

Solar Radiation: Deledd() Performance Comparison

- Deledd() is compute-intensive as well as repeatedly called
- Deledd() takes ~26.9% of radiation, which is 5.5 times more than Cldflx()
  - Cldflx() takes ~5.5% of radiation
- For the configuration of 128x128 columns, performance comparison against one CPU core

Processor	Time (micro second)	Speedup
CPU	68850	
OpenACC with IO	14440	4.77X
CUDA Fortran with IO	11898	5.78X
CUDA Fortran without IO	220	321X

142X (U. Wisconsin)

Earth Science Technology Office

Performance: K20 vs Fermi

GCE configuration: 128 X 128 Columns, GPU Thread configuration: 32 X 4

	K20: CPU/GPU	Fermi: CPU/GPU	K20 GPU/Fermi GPU
Soluv() (Micro second)	19,406,336/ 2,450,730 = 7.92	20,091,133 / 2,717,804 = 7.39	2,717,804 / 2,450,730.0 = 1.11
Solir() (Micro second)	71,337,870 / 7,859,607 = 9.08	74,055,186 / 9,112,408 = 8.13	9,117,496/7,859,607 = 1.16

➢ *K20 is faster than Fermi*

- 1027 microsecond for deleddgpu() without copy into and out of GPU
- 1858 microsecond for an array, tst (128x128, 43), copy out of GPU with pitch option
- 7579 microsecond for tst copy out of GPU with PGI Fortran " = "option

➢ *Copy-in and -out is the performance bottleneck*

Long Wave Radiation

- Effort
  - Tablup() is the most compute intensive one. It uses look-up tables.
  - For Tablup(), used shared memory, constant memory, copied the actually-used table. However, performance does not improvement since the time for copy-into-and-out-of-GPU is dominant compared to computational time
- Approach
  - Rewrite the codes and port the whole code into GPU to reduce copy-in-and-out time
    - Pull the do loop for column out of kernels
  - Fit tables in formula
    - Too complicated to fit in a reasonable expression
  - Pre-calculate values from tablup() and store them into arrays
    - Not portable

Optimizing One Moment Microphysics Code for GPU

- Code improvements for GPU
  - Merge many loops into one
  - Change arrays into scalars
  - Modularlize the code
- Accuracy comparison in CPU
  - Same in binary outputs between the old and new codes
- Comparison in computational efficiency

– Original code running:	30.52 s
– New code running:	22.46 s
– Performance improvement:	35% (in CPU)

ESTO Earth Science Technology Office

Porting One-Moment Microphysics Code into GPU

- Difference in code structure

```
…
 if(tair(i,j).lt.t0) then
         y1(i,j)=max( min(tairc(i,j), -1.), -31.)
         it(i,j)=int(abs(y1(i,j)))
         y1(i,j)=rn12a(it(i,j))
         y2(i,j)=rn12b(it(i,j))
         y3(i,j)=rn13(it(i,j))
        psfw(i,j)=max(d2t*y1(i,j)*(y2(i,j)+r12r*qc(i,j))*qi(i,j),0.0)
        psfi(i,j)=y3(i,j)*qi(i,j)

        if(ilang.eq.-1)then                    !xp
          tmp=BergCon1(it(i,j))*qi(i,j)
     1      +BergCon2(it(i,j))*rrho(k)*rn25*exp(beta*(tair(i,j)-t0))
          psfi(i,j)=max(tmp*d2t,0.0)
        endif
…
```

Original code in arrays

```
…
 if(tair.lt.t0) then
         y1=max( min(tairc, -1.), -31.)
         it=int(abs(y1))
         y1=rn12a(it)
         y2=rn12b(it)
         y3=rn13(it)
        psfw=max(d2t*y1*(y2+r12r*qc)*qi,0.0)
        psfi=y3*qi

         tmp=BergCon1(it)*qi                            &
          +BergCon2(it)*rr0*rn25*exp(beta*(tair-t0))
!   1      +BergCon2(it)*rrho(k)*rn25*exp(beta*(tair-t0))
         psfi=max(tmp*d2t,0.0)

       endif
…
```

Final code in scalars

- Performance in GPU
  - New code in CPU: 22.46 s
  - Final code in GPU: 5.95 s
  - Efficiency of GPU: 3.8x Included I/O between CPU and GPU

    5.1x Compared to the original code

Porting Two-Moment Microphysics Code into GPU

- Profiled Morrison 2-moment scheme
- Identified the most compute-intensive component, sedimentation, which is 30% of total microphysics computing time
- Reengineered the code and porting it to GPU

Baseline Test of Code Merging

- Merged the new GPU radiation code into the baseline test system
- Merging and testing the new GPU one-moment microphysics code into the test system
- Merged the new CPU two-momentum microphysics code into the test system

Lesson Learnt

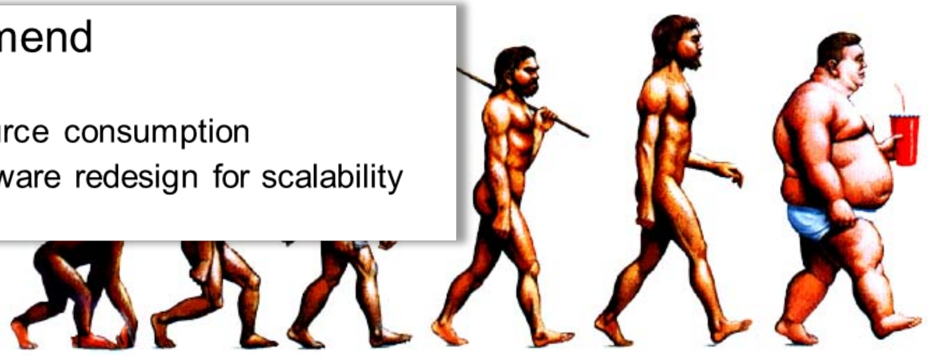
- Data transfer speed between CPU and GPU is the main performance bottleneck
  - Code reengineering is necessary to reduce unnecessary data transfer
- OpenACC is easy to use, however, its performance is not appealing.
- CUDA Fortran offers better performance, however, it requires more code reengineering and is challenging to debug

Summary: evolving to next-gen HPC

- Accelerators show promise but too little return on current hardware
- Prepping applications for next gen. hardware underway
  - Increasing concurrency
  - Increasing vectorization
  - Decreasing memory footprint
- More attention needs to be paid to fine-grained parallelism going forward
- WRF is heading in the wrong direction on memory use but no clear evidence it's hurting performance **... yet**

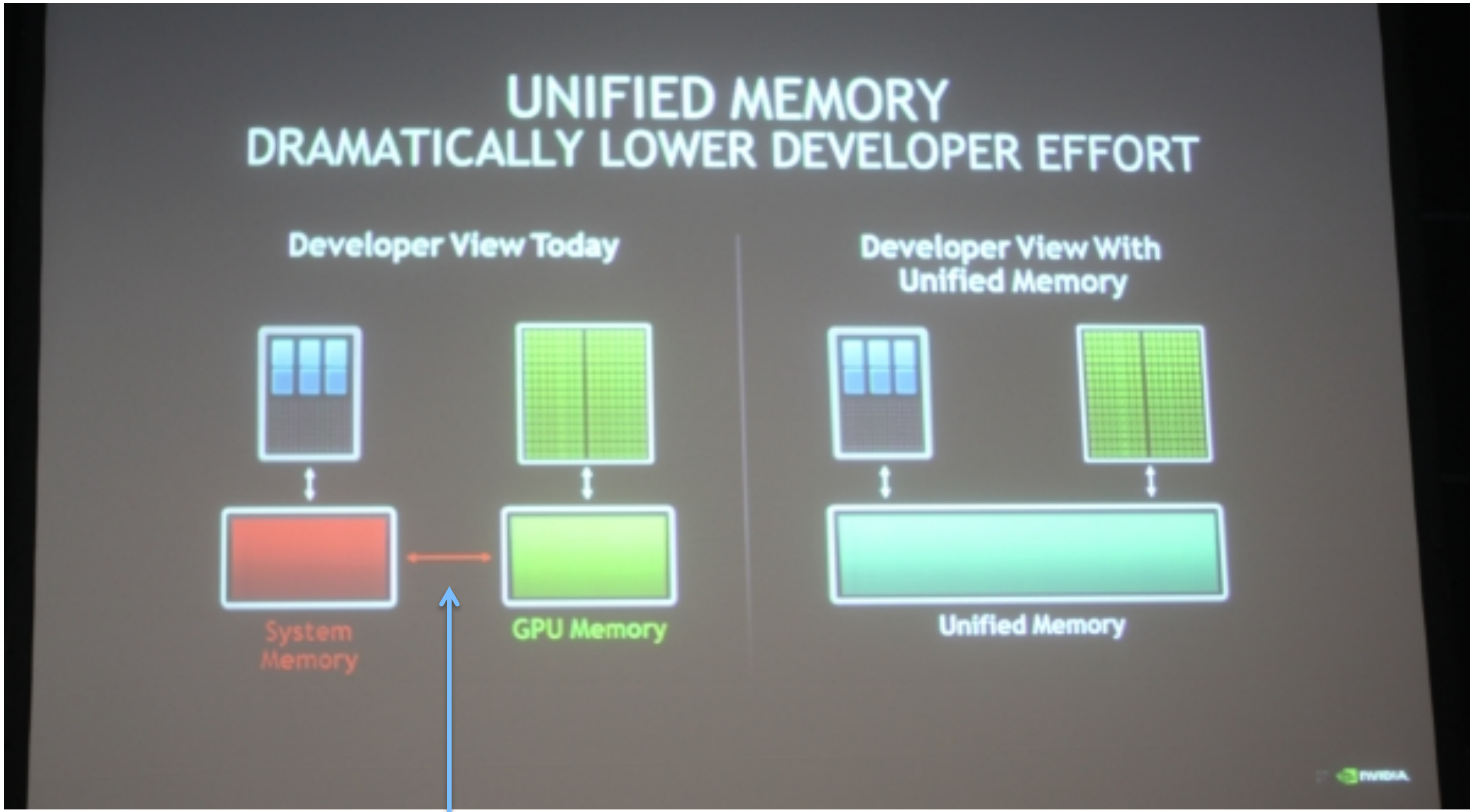
Recommend

- Further study
- Test for performance and resource consumption
- Consider requirements for software redesign for scalability



Courtesy of John Michalakes, NOAA/EMC

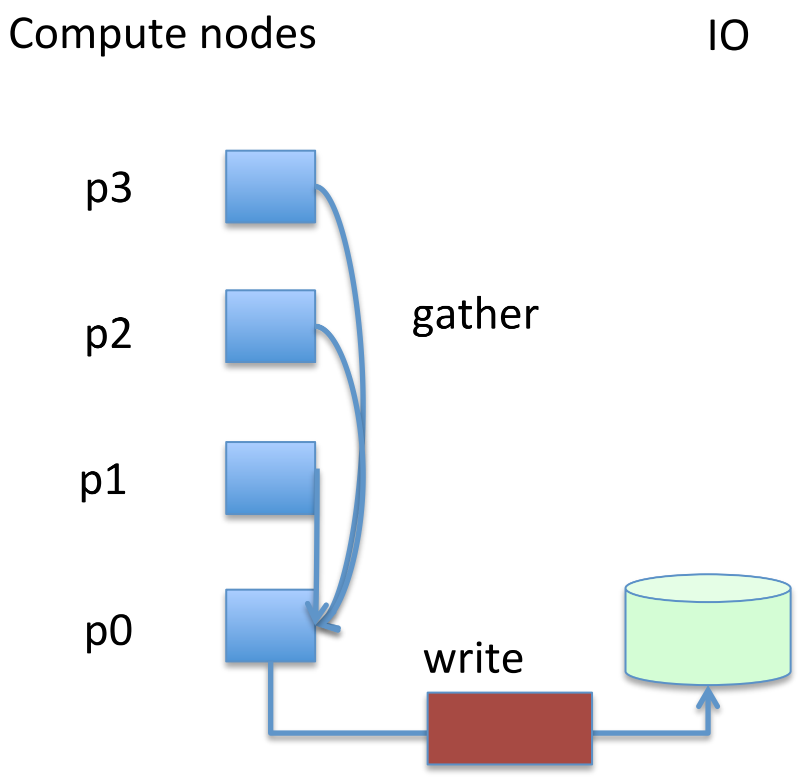
Next-Generation GPU: CPU-GPU Unified Memory and NVLink



NVLink: 80 GB/s

Extend Asynchronous I/O

Current IO Architecture

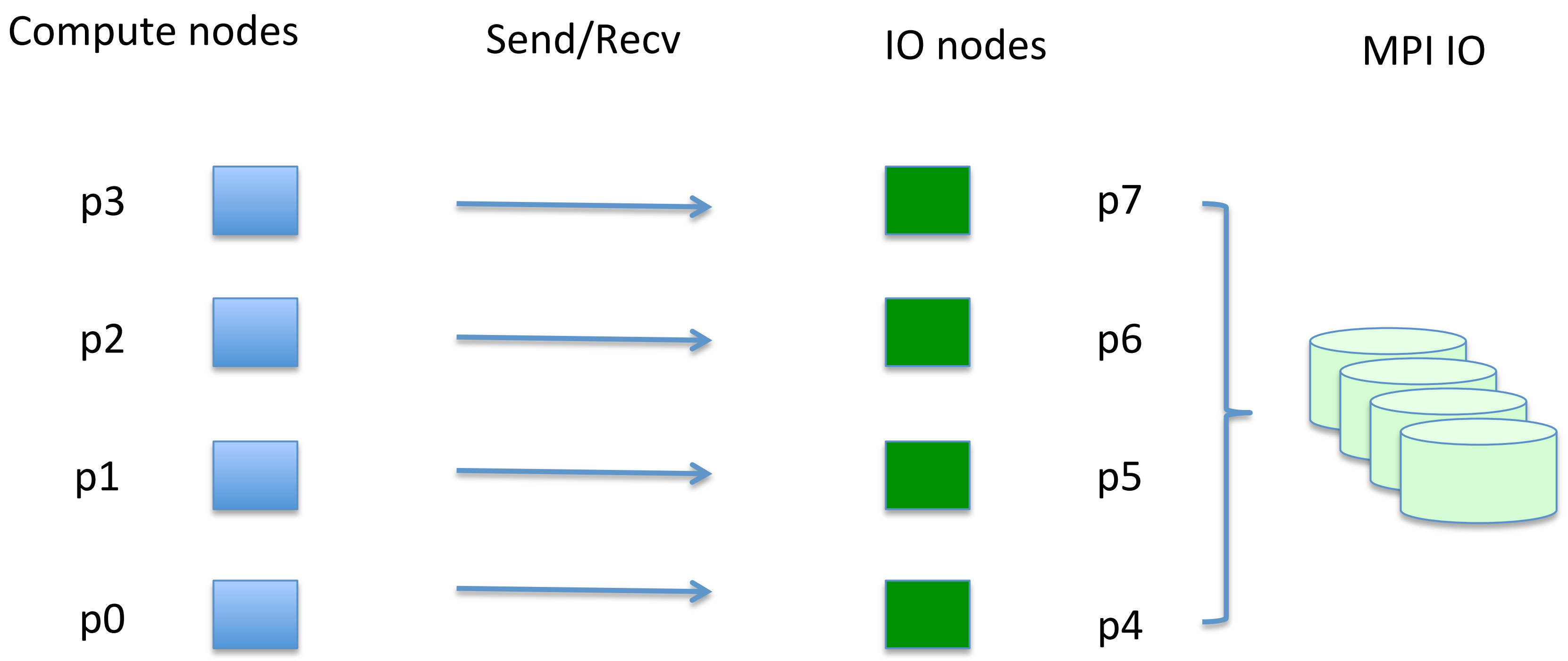


File system, disk system

- IO performs in a root process. It needs two operations:
  - Gather distributed data arrays to root process,
  - Write them out to disk at root process
- Issues
  - Gather can be time consuming
  - Memory size is limited in a single node

Total IO time = gather() + write ()

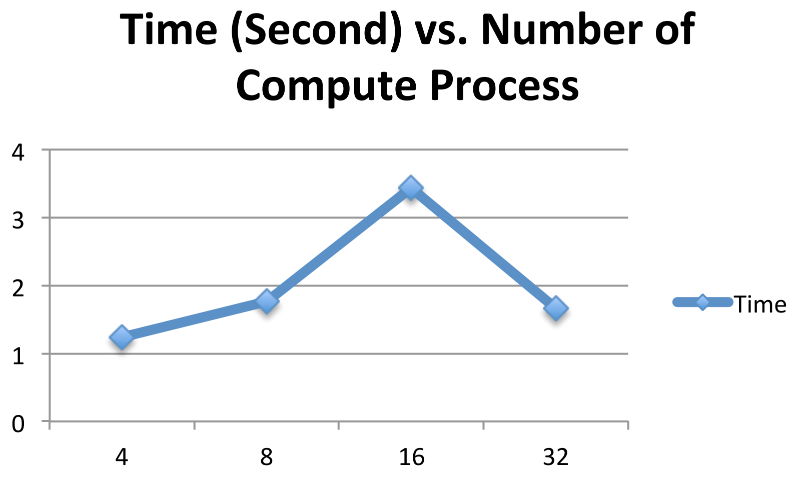
Current Parallel AsyncIO Implementation

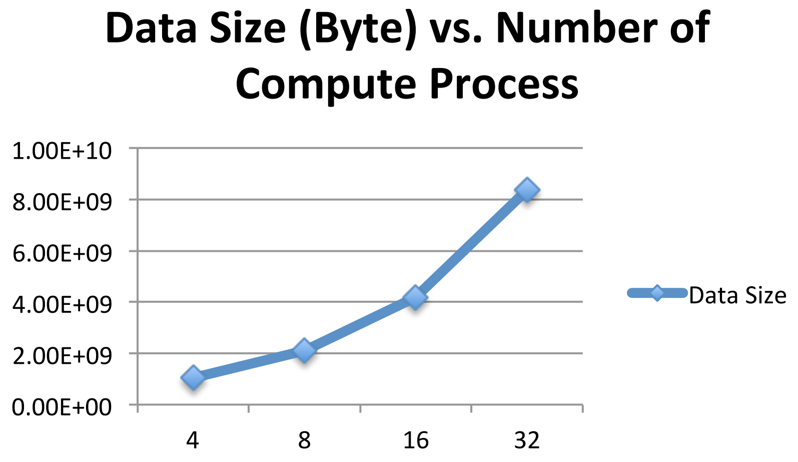


Total IO time = send()
Note: (1) For nonbloack send, no need for receive() to complete, (2) no need for write() to complete

Parallel AsyncIO Performance

- Output real arrays
  - Test at NCCS Discover
    - Intel Westere node (2 hex-core processors per node)
    - One MPI process per node
  - Data configuration
    - One array size on each process: 32*32*40
    - Do-Loop times: 1600
- With parallel AsyncIO, IO time depends on the interconnection speed rather than the number of processes





Summary

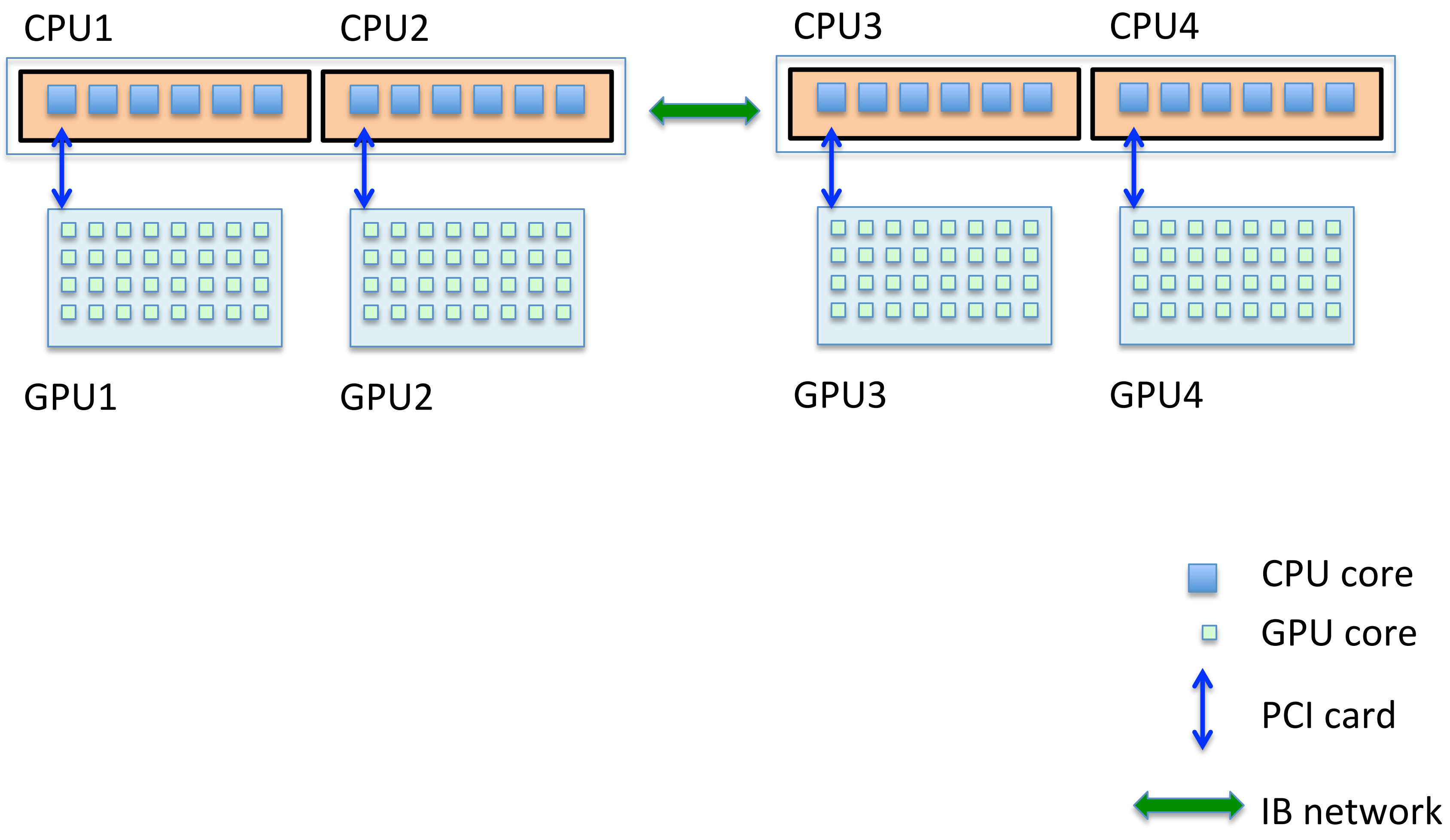
- Accomplished work
  - Explored two GPU porting approaches: OpenACC and CUDA Fortran. We chose CUDA Fortran due to its good performance. However, its coding and validation are challenging.
  - Ported radiation and integrated it into test baseline.
  - One-moment physics has been extensively reengineered and ported and is being validated
  - Two-moment physics has been reengineered and is being ported.
  - AsyncIO has been extended with parallel IO capability

Summary

- Work to be accomplished
  - Port microphysics (one-moment and two-moment) and integrate them into GCE and WRF along with radiation.
  - Enhance AsyncIO with NetCDF and HDF output support as well as compression and integrate it into GCE and MMF

backup

CPU-GPU System Diagram



CPU-GPU Implementation Approach

- For GPU, use CUDA Fortran
  - Plan to use OpenACC again if its performance is sufficiently good
- Between GPU and CPU, use CUDA API to copy data in and out
  - Currently use one CPU core to communicate with one GPU
    - CUDA 4.0
  - Plan to use all cores in a CPU to communicate with one GPU
    - CUDA 6.0
- Among CPU cores, use MPI to communicate

Updated NVIDIA GPU Roadmap

- Roadmap released on March 17, 2014
  - Unified Memory: This will make building applications that take advantage of what both GPUs and CPUs can do quicker and easier by allowing the CPU to access the GPU’s memory, and the GPU to access the CPU’s memory, so developers don’t have to allocate resources between the two.
  - NVLink: Today’s computers are constrained by the speed at which data can move between the CPU and GPU. NVLink puts a fatter pipe between the CPU and GPU, allowing data to flow at more than 80GB per second, compared to the 16GB per second available now
  - Pascal Module: NVIDIA has designed a module to house Pascal GPUs with NVLink. Pascal is due in 2016
- Impact
  - The performance bottleneck and coding/debugging issue related to copy in/out between CPU and GPU will disappear and acceleration with GPU will be more evident.

Solar Radiation: Compare CPU and GPU Results

For example, in solir() routine, all-sky flux (downward minus upward), flx, is

0.7947214478638445	without GPU
0.7949332959524538	with GPU

Solar Radiation: GPU Code Example

GPU kernel driver

```
  integer devnum
  integer dev_pitch
  call wrap_cudaGetDevice(devnum)
  call wrap_cudaSetupMemoryUsage(devnum)
  dev_pitch = 64

! input
! size m x np
  call wrap_cudaMallocPitch(dev_tautob, dev_pitch, m, np)

! output
  call wrap_cudaMallocPitch(dev_rrt, dev_pitch, m, np)

!-----memcpy 2D arrays
!     size m x np
      call wrap_cudaMemcpy2DHostToDevice(dev_tautob, dev_pitch, tautob, &
       m, m, np.)

call deleddKernel<<<dimGrid,dimBlock>>>(m, dev_pitch, np, &
           dev_cosz, &
           dev_tautob, dev_ssatob, dev_asytob, &
           dev_rrt, dev_ttt, dev_tdt)

!     size m x np
      call wrap_cudaMemcpy2DDeviceToHost(rrt, m, dev_rrt, dev_pitch, &
       m, np)

!     size m x np
      call wrap_cudaFree(dev_tautob, dev_pitch*np*8)
```

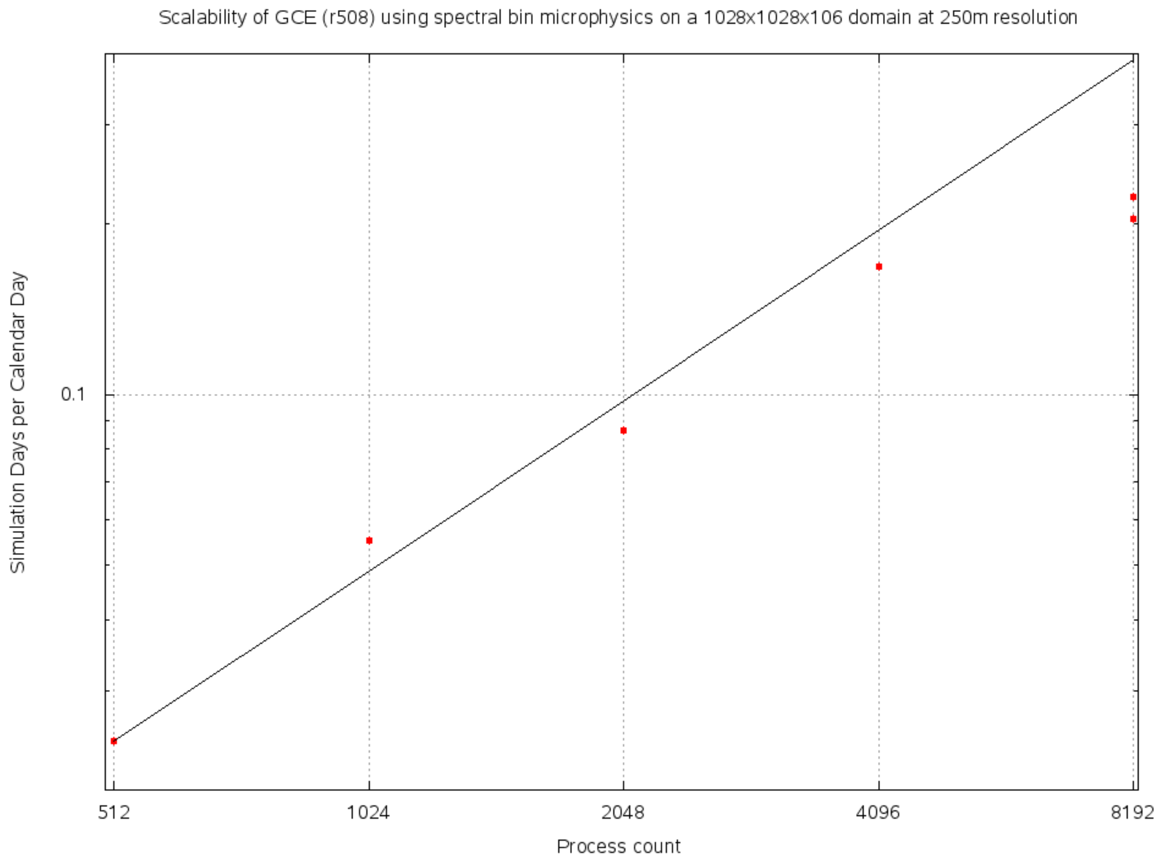
GPU kernel

```
    attributes(global) subroutine deleddKernel(m, dev_pitch, np, &
     dev_cza1,&
     dev_tau1, dev_ssc1, dev_g01,&
     dev_rr1, dev_tt1, dev_td1)

!-----input parameters
    integer, value :: m, dev_pitch,np
    real :: dev_cza1(dev_pitch)   ! 1D
!    input that are 2D module variables
    real :: dev_tau1(dev_pitch,np.)

!-----output parameters
    real :: dev_rr1(dev_pitch,np.)
```

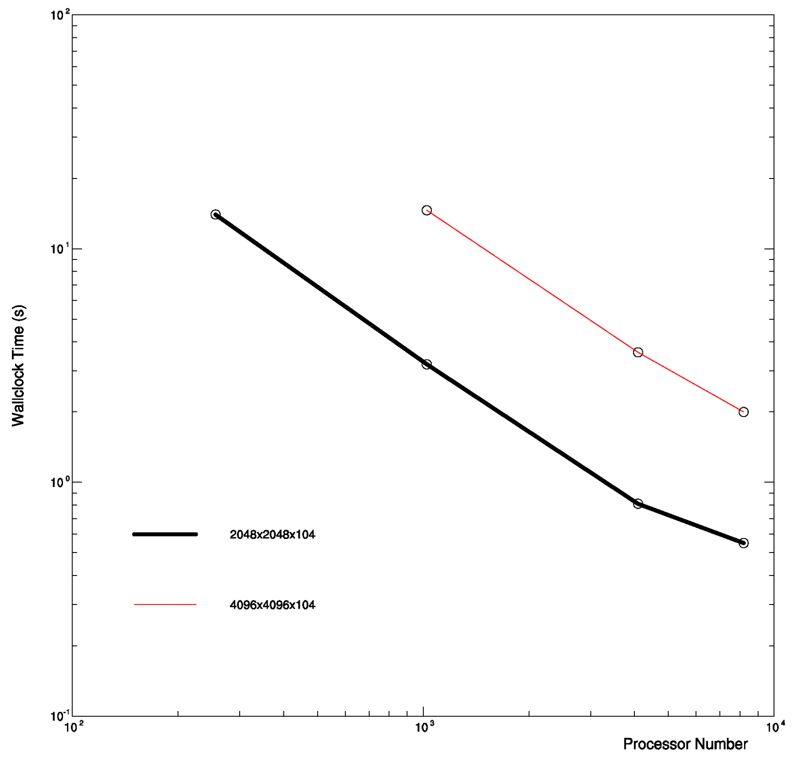
GCE Scalability with Bin Physics and Parallel IO



As the number of processes increases, the domain size decreases and the percentage of communication cost (halo update) increase. Consequently, scalability is not linear. Without parallel IO, 1028x1028x46 run fails

Simulations were carried out in NASA NAS Pleiades. Use ivybridge nodes with 16 ranks per node evenly distributed on the two sockets

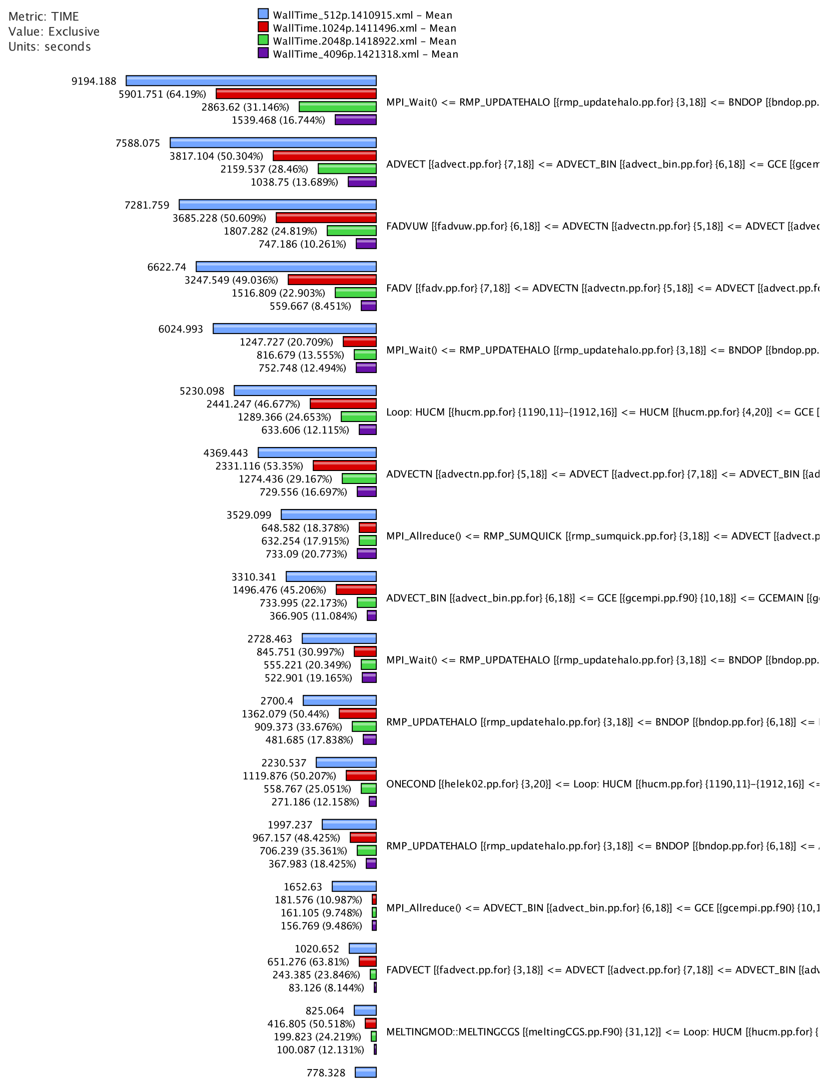
GCE Scalability with Bulk (One-moment) Physics and without IO



As the number of processes increases, the domain size decreases and the percentage of communication cost (halo update) increase. Consequently, scalability is not linear. However, when the domain size is fixed, it is close to liner. For example, the run with 2048x2048x104 with 4096 processes takes the similar walk clock time to 4096x4096x104 with 1024 processes.

Simulations were carried out in NASA NAS Pleiades. Westere processors are used.

GCE Scalability with Bin Physics and Parallel IO



This perform profile shows that halo update and advection are two major time consumers.

Spectral Bin Microphysical Scheme in GPU

Preliminary Estimations of Computing time and GPU Efficiency
(based on a 24-hour model run)

Main processeses in bin microphysics	Time (s)	% of total bin microphysics time	GPU Efficiency Estimation
Diffusion*	4799	28.8	excellent
Coagulation*	4837	29.1	challenging
Freezing/Melting	1355	8.2	excellent

* Need to increase calling frequencies because cloud/precipitation is fast process
* Diffusion needs up to x10 increase in calling frequency
* Coagulation needs x2 increase in calling frequency