Improved GPU/CUDA based parallel weather and research forecast (WRF) Single Moment 5-class (WSM5) cloud microphysics

Article *in* IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing · August 2012

DOI: 10.1109/JSTARS.2012.2188780

Jarno Mielikainen
University of Wisconsin–Madison
80 PUBLICATIONS **2,060** CITATIONS
SEE PROFILE

Mitch Goldberg
National Oceanic and Atmospheric Administration
228 PUBLICATIONS **12,286** CITATIONS
SEE PROFILE

Improved GPU/CUDA Based Parallel Weather and Research Forecast (WRF) Single Moment 5-class (WSM5) Cloud Microphysics

Jarno Mielikainen, Bormin Huang , Allen H.-L. Huang, Mitchell D. Goldberg

ABSTRACT

The Weather Research and Forecasting (WRF) model is an atmospheric simulation system which is designed for both operational and research use. WRF is currently in operational use at the National Oceanic and Atmospheric Administration (NOAA)'s national weather service as well as at the air force weather agency and meteorological services worldwide. Getting weather predictions in time using latest advances in atmospheric sciences is a challenge even on the fastest super computers. Timely weather predictions are particularly useful for severe weather events when lives and property are at risk. Microphysics is a crucial but computationally intensive part of WRF. WRF Single Moment 5-class (WSM5) microphysics scheme represents fallout of various types of precipitation,

Drs. J. Mielikainen, B. Huang an A. Huang are with Cooperative Institute for Meteorological Satellite Studies, University of Wisconsin-Madison (bormin@ssec.wisc.edu). Dr. M. Goldberg is with NESDIS Center for Satellite Applications and Research, National Oceanic and Atmospheric Administration.

condensation and thermodynamics effects of latent heat release. Therefore, to expedite the computation process, Graphics Processing Units (GPUs) appear an attractive alternative to traditional CPU architectures. In this paper, we accelerate the WSM5 microphysics scheme on GPUs and obtain a considerable speedup thereby significantly reducing the processing time. Such high performance and computationally efficient GPUs allow us to use higher resolution WRF forecasts. The use of high resolution WRF enables us to compute microphysical processes for increasingly small clouds and water droplets. To implement WSM5 scheme on GPUs, the WRF code was rewritten into CUDA C, a high level data-parallel programming language used on NVIDIA GPU. We observed a reduction in processing time from 16928 ms on CPU to 43.5 ms on a Graphics Processing Unit (GPU). We obtained a speedup of 389x without I/O using a single GPU. Taking I/O transfer times into account, the speedup obtained is 206x. The speedup was further increased by using 4 GPUs, speedup being 1556x and 357x for without I/O and with I/O respectively.

Keywords: WRF, cloud microphysics, WSM5, GPU, parallel processing, CUDA

1. INTRODUCTION

The science of meteorology explains observable weather events. The Greek philosopher Aristotle wrote a book called Meteorologica back in 340 B.C which had an account of weather and climate at that time. Topics covered included clouds, rain, snow, hail, thunder and hurricanes. However, only after the invention of weather instruments in the late 18th century, meteorology emerged as a genuine natural science [1]. One application of meteorology is weather forecasting, which can trace its roots back millennia. Weather forecasts require quantitative data about the current state of the atmosphere. The first weather observing network was established in 1654 by Ferdinando II del Medici [2]. At that time, atmospheric measurements included rain, wind and humidity. Today, radiosondes are used to get measurements from upper air data. Numerical weather forecasting was first proposed by Bjernes in 1904. However, numerical weather forecasting has become feasible only after the development of computers. Since atmosphere is a fluid, numerical weather prediction samples the state of the fluid at a given time and forms an estimate of the future state of the fluid using equations of fluid dynamics and thermodynamics. Weather models divide the planet into 3D grid. Laws of physics, chemistry and fluid motion are used to model weather using differential equations. These mathematical models are used to calculate winds, solar radiation, humidity, heat transfer and surface hydrology for each grid cell. In addition, the interactions with neighboring cells are calculated to predict atmospheric properties in the future.

Some of the most powerful supercomputers in the world are used for numerical weather forecasting. With the advent of Graphics Processing Unit (GPU) architectures, inherently parallel problem of weather forecasting can be effectively solved using GPUs. GPUs have hundreds of parallel processor cores for execution of tens of thousands of parallel threads. Unlike traditional CPUs, GPUs are not optimized for a single thread performance. Instead, they are optimized for executing a large number of threads simultaneously. Therefore, a single thread performance on a GPU is lower than that on a contemporary CPU. This difference is because the processor architectures require legacy software to be rewritten for efficient parallel execution on GPUs [3].

GPUs have been used very successfully for numerous different computational problems. Variety of GPU application domains can be found in [4] and [5]. The covered industries include scientific, electronic design automation, computer vision, finance, medicine, imaging, engineering, gaming, environmental science and green computing. GPUs have also been used in applications such as synthetic aperture radar (SAR) simulation system [6] and computation of ray-traced troposphere delays [7]. A collection of the latest GPU research in Earth Observation and remote sensing can be found in [8]. In [9], a very comprehensive review of the recent developments in high performance computing for remote sensing is presented. The review covers papers is such areas as spectral signature (endmember) [10][11], target detection [12], radiative transfer models [13][14] and data compression [15]-[17].

The Weather Research and Forecasting (WRF) model is an atmospheric simulation system, which is designed for both operational and research use. This common tool aspect promotes closer ties between research and operational communities. WRF is currently in operational use at the National Oceanic and Atmospheric Administration (NOAA)'s national weather service as well as at the air force weather agency and meteorological services worldwide. Microphysics is a crucial but computationally intensive part of WRF. WRF Single Moment 5-class (WSM-5) microphysics scheme represents fallout of various types of precipitation, condensation and thermodynamics effects of latent heat release. In this paper, we will show the results of our WSM5 optimization efforts using Graphics Processing Units (GPUs). The use of GPUs will allow using higher resolution WRF forecasts. The use of high resolution WRF enables computing microphysical processes for increasingly small clouds and water droplets.

Since public agencies need more computing capacity to forecast dangerous weather events and analyze long-term climate change, there already exists a significant amount of work on parallel WRF model. Weather and climate models such as WRF exhibit fine-grained data parallelism, which has been exploited by vector processors [17] and the Single Instruction Multiple Data (SIMD) supercomputers of the 1990s [19][20]. Modern large microprocessor based clusters are unable to exploit parallelism much finer than one sub-domain, i.e., the geographic region(s) allocated to one processor for weather modeling. CPUs lack the memory bandwidth and functional units needed to exploit fine-grained parallelism. However, modern GPUs, like earlier SIMD systems, are designed to exploit massive fine-grained parallelism.

WRF contains a lot a different physics and dynamics options reflecting the experience and input of the broad scientific community. The WRF physics categories are microphysics, cumulus parametrization, planetary boundary layer, land-surface model and radiation. Explicitly resolved water vapor, cloud and precipitation processes are included in microphysics [21]. Earlier GPU work on WRF modules includes an older version of WSM5 microphysics portion of WRF model [22]. The Long-Wave Rapid Radiative Transfer Model (RRTM) component of the WRF code was ported to the GPU using CUDA Fortran [23]. Work has also been performed on GPUs to accelerate other weather forecasting models besides WRF. In [24], a next-generation Japanese production weather code was demonstrated to have a speedup of 80x using 528 GPUs. Dynamics routine of High Resolution Local Area Model (HIRLAM) has been accelerated by an order of magnitude in [25]. Dynamics portion of Non-hydrostatic Isosahedral Model (NIM), a next-generation weather model was converted to run on a GPU and a speedup of 34x was achieved in [26]. GPU acceleration of radiance transfer computation, both for single profile and multi-profile processing were shown in [27] and [28] respectively. In this paper, we will accelerate a newer version of WSM5 on GPUs.

The rest of the paper is organized as follows. Section 2 describes WSM5 microphysics scheme. Section 3 explains the basics of CUDA computing engine for NVIDIA GPUs. In Section 4, the

GPU-accelerated WSM5 microphysics scheme is explained. Section 5 has comparison of GPU results to the original Fortran code. Finally, Section 6 concludes the paper.

2. WRF WSM5 CLOUD MICROSPHYSICS SCHEME

The WSM5 scheme predicts five categories of hydrometrics: water vapor, cloud vapor, cloud ice, rain and snow. Figure 1 depicts the different categories and production rates, which are explained in Table 1. WSM5 allows supercooled water to exist and a gradual melting of snow falling below the melting layer. The details of computational processes are described in [29][30].

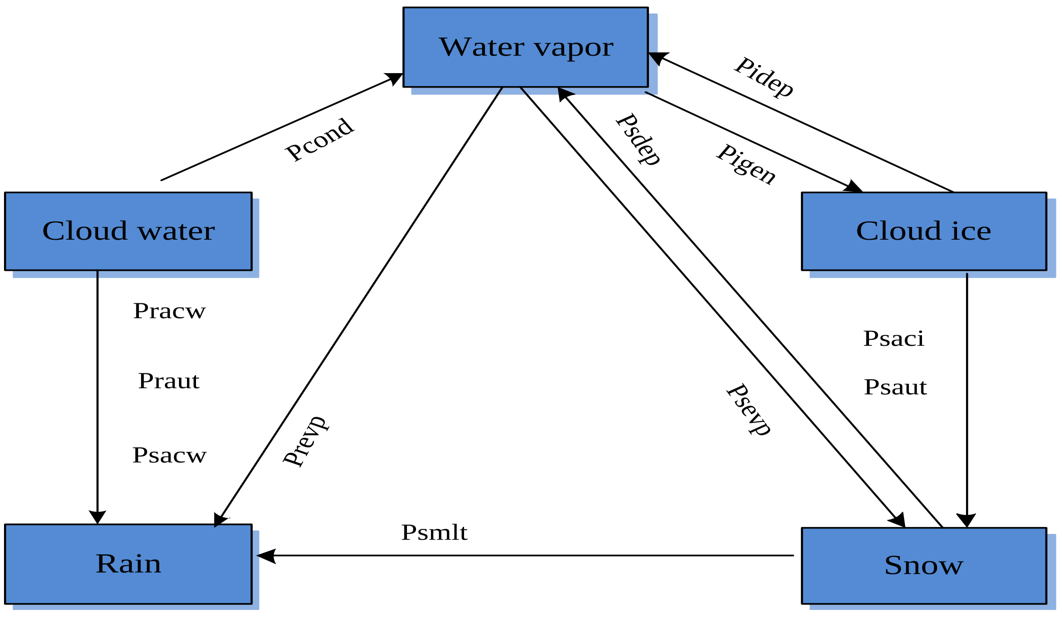


Fig. 1. Flowchart of the microphysics processes in the WSM5 scheme.

Table 1. List of symbols in Fig. 1

Symbol	Description
P_{COND}	Production rate for condensation-evaporation of cloud water
P_{IDEP}	Production rate for deposition sublimation rate of ice
P_{IGEN}	Production rate for generation of ice from water vapor
P_{RACW}	Production rate for accretion of cloud water by rain
P_{RAUT}	Production rate for autoconversion of cloud water to form rain
P_{REVP}	Production rate for accretion cloud ice by snow
P_{SACI}	Production rate for accretion of rain by snow
P_{SACW}	Production rate for accretion cloud water by snow
P_{SAUT}	Production rate for autoconversion of cloud ice to form snow
P_{SDEP}	Production rate for deposition-sublimation rate of snow
P_{SEVP}	Production rate for evaporation of melting snow
P_{SMLT}	Production rate for melting of snow from cloud water

Condensation of water vapor to cloud water is expressed as

$$P_{COND} = (q - q_{sw})\left[\Delta t\left(1 + \frac{L_v^2}{C_{pm} R_V T^2}\right)\right],$$

where cloud water evaporates, if $q < q_{SW.}$, which is saturated value with respect to cloud water, Δt is time increment, L_v is latent heat of condensation, C_{pm} is specific heat of moist air at constant pressure, R_V is gas constant for water vapor and T is temperature.

When air supersaturated with respect to ice, the growth rate of ice crystals by deposition of water vapor is

$$P_{IDEP} = \frac{4D_I(S_I - 1)N_I}{A_I + B_I},$$

where D_I is cloud ice diameter, S_I is saturation ratio over ice, A_I and B_I are thermodynamic terms.

When temperature, T, is smaller than reference temperature, T_0, and the air is supersaturated with respect to ice, the initiation rate of cloud ice from water vapor is

$$P_{IGEN} = min[(q_{I0} - q_I)/\Delta t, (q - q_{SI})/\Delta t],$$

where q_{Io} is mixing ratio of ice nuclei, q_I is mixing ratio of ice crystal, q is mixing ratio of water vapor and q_{SI} is saturated value with respect to ice.

The growth rate of raindrop by accretion of cloud water is

$$P_{RACW} = \frac{\pi a_R E_{RC} n_{0R}\, q_c\, \Gamma(3+b)}{4\lambda_r^{3+b}} \left(\frac{\rho_o}{\rho}\right)^{1/2},$$

where a_R is *841.9 m/s*, $E_{RC}=1$ is rain water collection efficiency, $n_{0R} = 8 \cdot 10^6$ is intercept parameter of rain, q_c is mixing ratio of cloud water, Γ is gamma function, λ_r is a slope of rain size distribution, ρ_0 is air density at reference state, and ρ is air density.

Production rate for auto conversion of cloud water to form rain

$$P_{RAUT} = \frac{0.104 g E_C \rho_0^{4/3}}{\mu (N_C \rho_W)^{1/3}} \cdot q_C^{7/3} H(q_C - q_{C0}),$$

where *H()* is Heaviside step function, *g=9.8* is gravitational acceleration, E_C is mean collection efficiency, $\mu=1.718 \cdot 10^{-5}$ is dynamic viscosity of air, N_C is concentration of cloud water droplet, ρ_W is water density, q_c is mixing ratio of cloud water and q_{c0} is critical mixing ratio of cloud water.

The evaporation rate for rain is

$$P_{REVP} = \frac{2\pi N_{0R}(S_W - 1)}{(A_W + B_W)} \cdot \left[\frac{0.78}{\lambda_r^2} + \frac{0.31\,\Gamma\left(\frac{b_r+5}{2}\right) a^{1/2}}{\lambda_r^{-(b_r+5)/2}} \left(\frac{\mu_k}{D}\right)\left(\frac{a_R \rho}{\mu_k}\right)^{1/2} \left(\frac{\rho_o}{\rho}\right)^{1/4} \right],$$

where S_W is saturation ratio with respect to water, A_W and B_W are thermodynamic terms, $a_R=841.9$ and μ_k is kinematic viscosity.

The rate of accretion of cloud ice by snow is:

$$P_{SACI} = \frac{\pi q_I E_{SI} n_{os} \left|V_S - V_I\right|}{4} \cdot \left[\lambda_S^{-3} + 2 D_I \lambda_S^{-2} + D_I^2 \lambda_S^{-1} \right],$$

where E_{SI}, the collection efficiency of the snow for cloud ice, V_s is mass weighted fall speed of snow, V_I is mass weighted fall speed of cloud ice, λ_S is a slope of snow size distribution and D_I is cloud ice diameter.

The accretion rate of cloud water by snow is

$$P_{SACW} = \frac{\pi a_s E_{SC} n_{os}\, q_c\, \Gamma(3+b_s)}{4\lambda_s^{3+b_s}} \left(\frac{\rho_o}{\rho}\right)^{1/2},$$

where E_{SW} =1 is the collection efficiency of the snow for cloud water and b_s is *0.41*. When $T > T_0$, P_{SACW} will contribute to form rain drop.

The aggregation rate of ice crystals to form snow is:

$$P_{SAUT} = 1.0 \times 10^{-3}\, \mathrm{e}^{0.025T_c} (q_i - q_{Io}).$$

The depositional growth rate of snow is

$$P_{SDEP} = \frac{4n_{0S}(S_i - 1)}{A_I + B_I} \cdot \left[0.65\lambda_s^{-2} + 0.44\Gamma \frac{\Gamma\left(\frac{b_s+5}{2}\right)}{\lambda_s^{-(b_S+5)/2}} \cdot \left(\frac{\mu_k}{D}\right)^{1/3} \left(\frac{a_s\rho}{\mu_k}\right)^{1/4} \left(\frac{\rho_o}{\rho}\right)^{1/4} \frac{\Gamma\left(\frac{b_s+5}{2}\right)}{\lambda_s^{-(b_S+5)/2}} \right],$$

where A_I and B_I are thermodynamic terms.

Evap
orati
on of
melti

$$P_{SEVP} = \frac{4n_{0S}(S_i - 1)}{A_W + B_W} \cdot \left[0.65\lambda_s^{-2} + 0.44\Gamma \frac{\Gamma\left(\frac{b_s+5}{2}\right)}{\lambda_s^{-(b_S+5)/2}} \cdot \left(\frac{\mu_k}{D}\right)^{1/3} \left(\frac{a_s\rho}{\mu_k}\right)^{1/4} \left(\frac{\rho_o}{\rho}\right)^{1/2} \frac{\Gamma\left(\frac{b_s+5}{2}\right)}{\lambda_s^{-(b_S+5)/2}} \right].$$

ng snow is

Production rate for melting of snow from cloud water is

$$P_{SMLT} = \frac{2\pi n_{0S}}{L_f} K_u \left(T - T_0\right) \left[0.65\lambda_s^{-2} + 0.44\Gamma \frac{\Gamma\left(\frac{b_s+5}{2}\right)}{\lambda_s^{-(b_S+5)/2}} \right],$$

where L_f is latent heat of fusion and K_u is thermal conductivity of air.

3. GPU COMPUTING

CUDA is an extension to the C programming language which offers a direct programming of the GPUs. It is designed such that its constructs allow for natural expression of data-level parallelism. A CUDA program is organized into two parts: a serial program running on the CPU and a parallel part running on the GPU. The parallel part is called a kernel. A CUDA program automatically uses more parallelism on GPUs that have more processor cores. A C program using CUDA extensions distributes a large number of copies of the kernel into available multiprocessors (MP) and executes them simultaneously.

Figure 2 presents a schematic visualization of a GTX 590 GPU device. Each multiprocessor has *32* CUDA cores and it executes in parallel with the other multiprocessors. Each multiprocessor has two groups of *16* CUDA cores. All sixteen cores in a group execute in a Single Instruction Multiple Data (SIMD) fashion with all cores in the same multiprocessor executing the same instruction at the same time. Each GPU has *1.5 GB* of global memory, which has a higher bandwidth than the DRAM memory in the CPUs.

The CUDA code consists of three computational phases: transmission of data into the global memory of the GPU, execution of the CUDA kernel and transmission of results from the GPU into the memory of CPU.

Figure 3 presents a schematic visualization of a multiprocessor (MP). There are four special function units (SFU) inside each MP for single-precision floating-point transcendental functions. Each MP has 16 load (LD) / store (ST) units for memory access. Each MP also has *32768 32*-bit registers which can be accessed in a single clock cycle. A nice introduction to the CUDA programming model can be found in [31].

CUDA takes a bottom-up point of view of parallelism in which a thread is an atomic unit of parallelism. Threads are organized into a three-level hierarchy. The highest level is a grid, which consists of thread blocks. A grid is a three-dimensional array of thread blocks. Thread blocks implement coarse-grained scalable data parallelism and they are executed independently, which allows them to be scheduled in any order across any number of cores. This allows the CUDA code to scale with the number of processors.

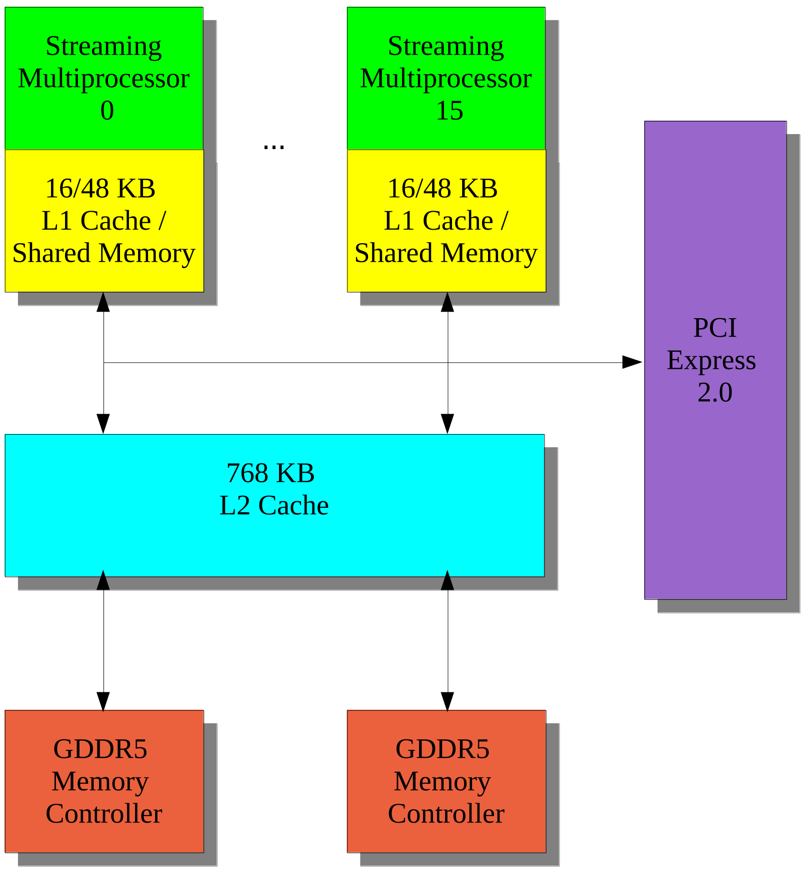

Streaming Multiprocessor 0
16/48 KB L1 Cache / Shared Memory
...
Streaming Multiprocessor 15
16/48 KB L1 Cache / Shared Memory
PCI Express 2.0
768 KB L2 Cache
GDDR5 Memory Controller
GDDR5 Memory Controller

Fig. 2. Schematic visualization of a GPU device.

Threads are executed in groups of *32* threads called warps. A CUDA core group inside a MP issues the same instruction to all the threads in a warp. When the threads take divergent paths, multiple passes are required to complete the warp execution. At each clock cycle, the MP schedules a suitable warp for execution. The scheduling favors those threads whose next instructions are not waiting for a long-latency instruction such as global memory access. Overloading the MP with a lot of active threads allows the GPU to hide the latency of slow instructions. Global memory loads and stores by threads of a half-warp (*16* threads). Different global memory accesses are coalesced by the device in as few as one memory transaction when the starting address of the memory access is aligned and the threads access the data sequentially. An efficient use of the global memory is one of the essential requirements for a high performance CUDA kernel. The other main issue to consider is to find enough data parallelism to keep all CUDA cores busy.

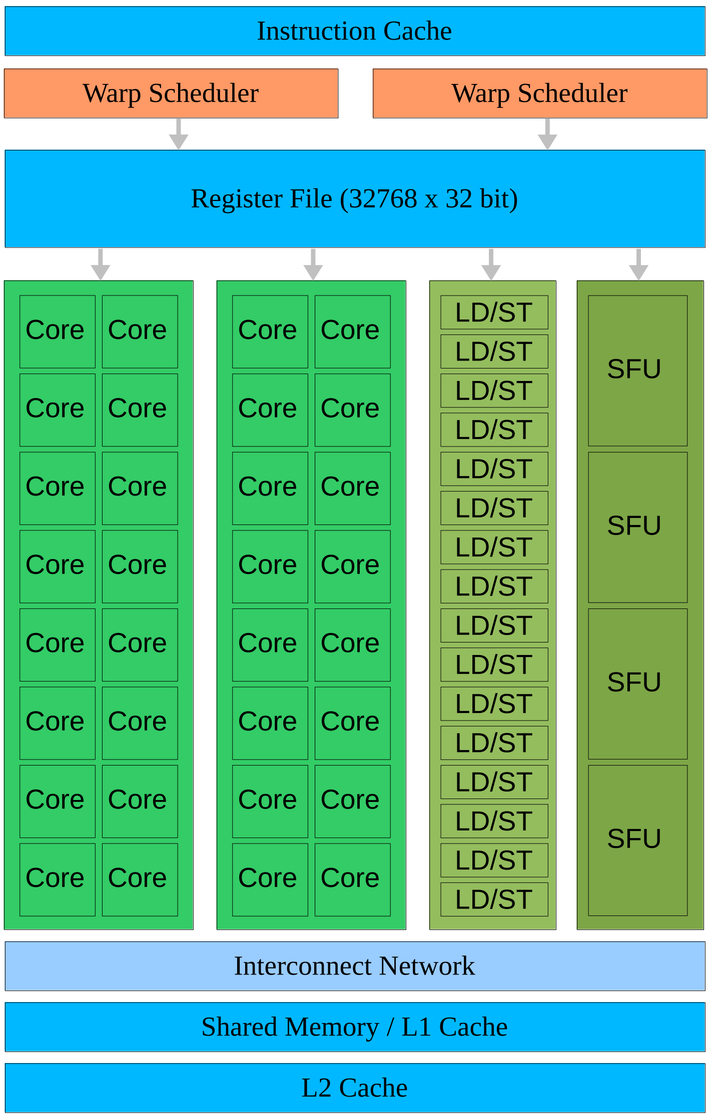


Fig. 3. Schematic visualization of a multi processor.

4. GPU ACCELERATED WSM5

4.1 The original Fortran code of WSM5 microphysics scheme

Profiling graph of the original Fortran code on a CPU is shown in Fig. 4. Graphs nodes contain information about the total processing time in percent of a sub-graph rooted at the node, processing time at the node and number of times the subroutine is executed. Directed graph edges have information about the percentage of the processing time spent on that specific subroutine call and the number of times the subroutine is called. In Table 2, the subroutines are explained.

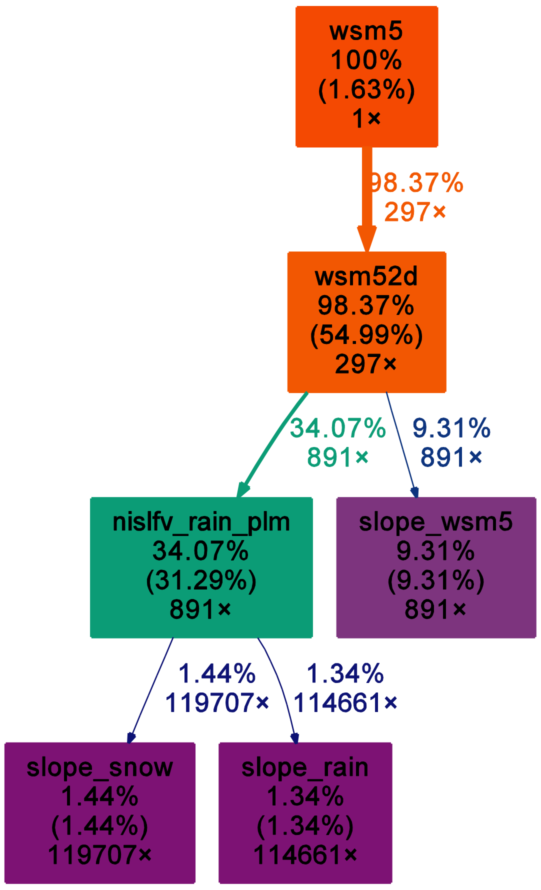


Fig. 4. Profiling graph of WSM5 microphysics scheme on a CPU. Graphs nodes contain information about the total processing time in percent of a sub-graph rooted at the node, processing time at the node and number of times the subroutine is executed. Directed graph edges have information about the percentage of the processing time spent on that specific subroutine call and the number of times the subroutine is called.

Table 2. Description of WSM5 microphysics subroutines

Subroutine name	Description
wsm5	Preparation for microphysics computation.
wsm52d	5-class mixed ice microphysics scheme of the Single-Moment MicroPhysics (WSMMP).

nislfv_rain_plm	Semi-Lagrangian forward advection for cloud with mass conservation and positive definite advection 2nd order interpolation with monotonic piecewise linear method.
slope_wsm5	Distribution slope parameter computation
slope_snow	Calculation of the distribution slope parameter, based upon conservation of the snow mass predicted by simulated microphysical processes.
slope_rain	Distribution slope parameter for rain.

4.2 Manual code translation of WSM5 from Fortran to standard C

As the original WRF code was written in Fortran 90, we ported the original WSM5 microphysics module from Fortran to CUDA C. As an intermediate conversion step, the original Fortran code was first rewritten using standard C. During Fortran to C conversion, some of the temporary arrays were replaced by scalar values, which are recomputed as needed. This was done keeping in mind that in GPUs it is faster to recompute values than to transfer them from relatively slower global memory. In other words, GPUs have higher computation to bandwidth capabilities than CPUs. The rewritten WSM5 code was verified against the Fortran code. After verification, C code was converted into CUDA C for data parallel execution on GPUs.

4.3 Converting C code into CUDA C

To test WSM5 we used a CONtinental United States (CONUS) benchmark data set for 12 km resolution domain for October 24, 2001 [32]. A WRF domain is a geographic region of interest discretized into a 2-dimensional grid parallel to the ground. Each grid point has multiple levels, which correspond to various vertical heights in the atmosphere. The size of the CONUS 12 km domain is *433 x 308* horizontal grid points with *35* vertical levels. As shown in Fig. 5, the test problem is a 12 km resolution 48-hour forecast over the Continental U.S. capturing the development of a strong baroclinic cyclone and a frontal boundary that extends from north to south across the entire U.S. For compiling Fortran code, we used default compiler options from WRF for gFortran: -O3 -ftree-vectorize -ftree-loop-linear -funroll-loops.

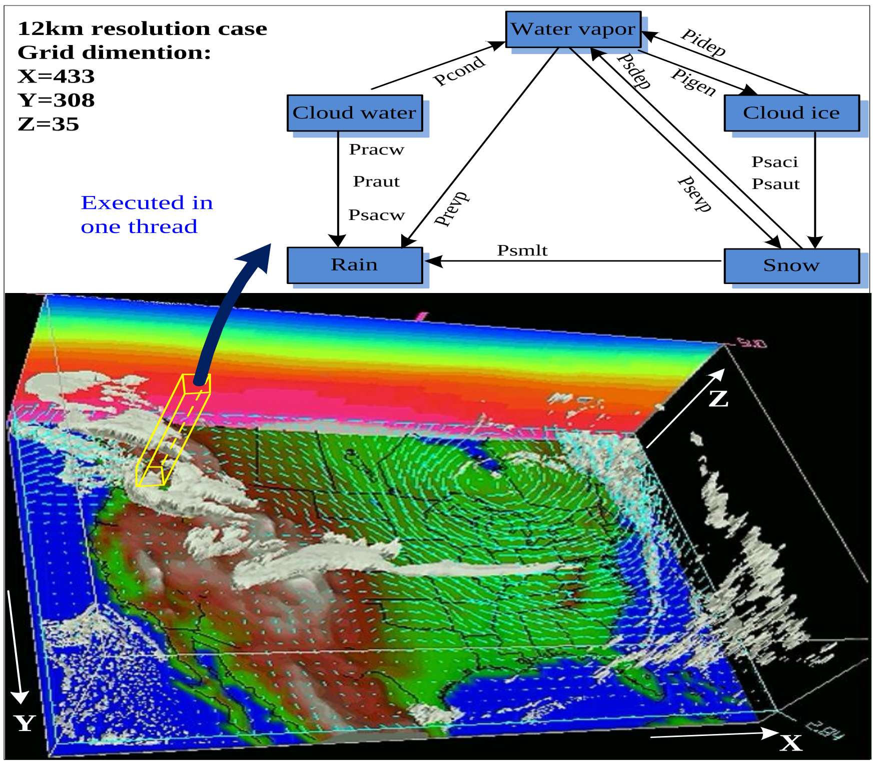


Fig. 5. U.S. Forecast for 48 hours using 12 km resolution. Frame 32 of Vis5D animation.

WSM5 microphysics computation is organized into a 3-dimensional grid where the microphysical process computations in each column (k-dimension) are processed independently. Thus, there are two dimensions of parallelism (i- and j-dimensions) to work with. Data-parallelism is achieved by giving each thread a unique global index in the i- and j-dimensions of the domain using thread and block indices. The result of this thread mapping is that multiple threads are carrying out the same computation on different data in parallel. The major changes made to CUDA C code for increasing its speed are as follows:

- On-chip fast memory was configured as 48KB of L1 cache and 16 KB of shared memory instead of default 16KB of L1 cache and 48 KB of shared memory.

- Restricted pointers were used for global variables in order to alleviate the aliasing problem, which inhibits compiler certain types of compiler optimizations. With __restrict__ keywords added to the pointer arguments, the compiler can reorder and do common sub-expression eliminations.
- Using register scalars instead of global memory arrays for storing temporary values. This process was applied to microphysical process variables listed in Table 1. In addition, several other temporary arrays were converted to scalars. Thus, their values would be recomputed using values in registers as needed instead loading a values from slower global memory.
- Compiler option *-use_fast_math* was utilized for faster but less accurate computation. The option forces GPU code to use intrinsic mathematical function. Also, denormalized numbers are flushed to zero. In addition, less precise division and square root operations are used.
- Three separate instances of slope function were written instead of using the same function for three separate function calls, which used only some of the output variables.

A CUDA C program distributes a large number of copies of the kernel into available multiprocessors to be executed simultaneously. Therefore, A CUDA program automatically uses more parallelism on GPUs that have more processor cores. This means that a lot of parallel threads have to be launched on GPU for effective parallel implementation. In order to achieve a high level of parallelism on a GPU we replaced the two outer loops by index computations using thread and block indices. Thus, the domain decomposition is such that each thread will compute values for all levels in one spatial position. This domain decomposition is illustrated in Fig. 6.

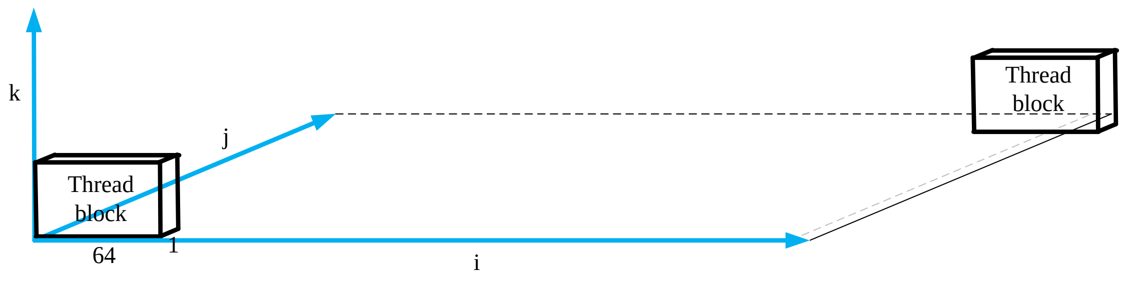


Fig. 6. Schematic visualization of the WSM5 microphysics domain decomposition.

Data-parallel nature of microphysics computation is illustrated in Fig. 7, which shows a block diagram of accretion of cloud water by rain (pracw). The same microphysics operation is performed for all the domain values at the same time in parallel. The C code of the process can be seen in Fig. 8.

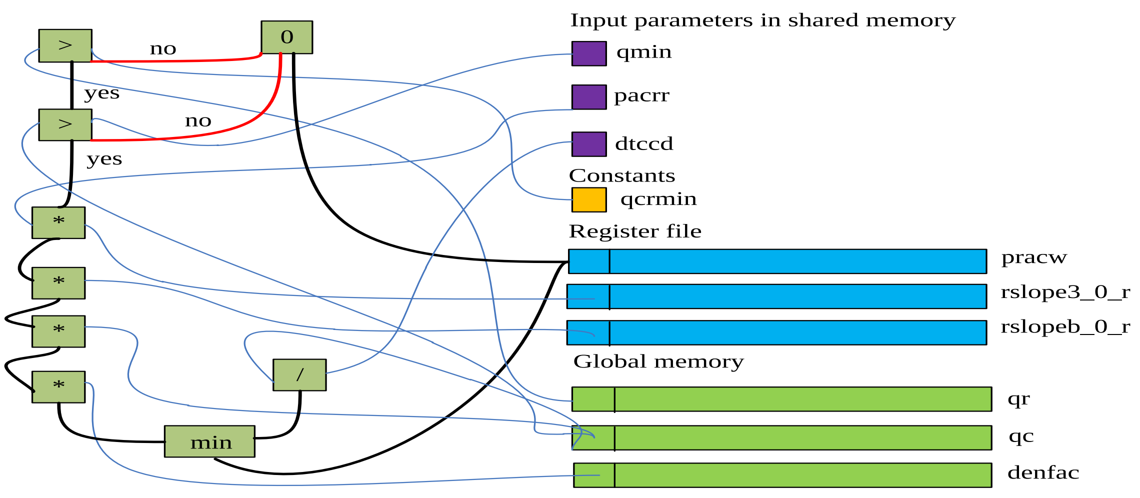


Fig. 7. Block diagram of a data-parallel kernel snippet of accretion of cloud water by rain (pracw).

```
if(qr[I3(i,k,j)] > qcrmin && qc[I3(i,k,j)] > qmin)
   pracw = min(pacrr * rslope3_0_r * rslopeb_0_r * qc[I3(i,k,j)] * denfac[I3(i,k,j)], qc[I3(i,k,j)] / dtcld);
else
   pracw = 0.0f;
```

Fig. 8. C code for accretion of cloud water by rain (pracw).

To evaluate the performance of the GPU accelerated WSM5, we ran our CUDA C code on GTX 590. Specifications can be seen in Table 3. Each MP has a data cache, which is shared among the CUDA cores. The data cache can be configured as 16 KB software managed cache called shared memory and 48 KB hardware cache (L1) or the other way around. There is also 768 KB of level 2 (L2) cache, which is shared among MPs. The CPU used in the tests was Intel i7 970 CPU running at 3.20 GHZ. A reduction of processing time from *16928 ms* on CPU to *52.8 ms* without I/O on a GPU means that a speedup of *321x* can be achieved. We used thread block size *64*. The CUDA C compiler was version 4.1 release candidate 1. Also, the code was compiled for 32-bit architecture for faster execution.

Table 3. Hardware specifications

Number of CUDA cores per GPU	512
Global memory bandwidth per GPU	164 GB/s
Global memory per GPU	1.5 GB
Frequency of CUDA cores	1.215 GHz
Number of 32-bit registers per multiprocessor	32768
L1 cache	16 KB / 48 KB
L2 cache size per GPU	768 KB

The achieved instruction per byte ratio for the kernel is 2.07. This ratio is the total number of instructions issued by the kernel divided by the total number of bytes accessed by the kernel from global memory. Balanced instruction ratio is defined as a ratio of the peak instruction throughput and the peak memory throughput of the CUDA device. For the GTX 590 the balanced instruction ratio is 3.80 instructions per byte, which is more than the achieved instructions per byte. Hence, the kernel is memory bandwidth limited.

4.4 Coalesced memory access

Recall that coalesced memory access happens when adjacent threads load or store data that in contiguous and aligned in memory. Because of CUDA’s coalescing rules, it is convenient to lay out memory in an array so that i-columns in the array start on 64 byte boundaries. Therefore, i-columns are padded with zeroes so that each i-column begins at a 64-byte boundary. Consequently, the full padded size of i-column dimension is increased from the original *idim* to *pitch*. The other two array dimensions are *kdim* and jdim. Therefore, the stride between adjacent elements in i- dimension is 1, stride between adjacent elements in k-dimension is *pitch* and the stride between adjacent elements in j-dimension is *pitch*kdim*. The memory layout of 3D arrays is illustrated in Fig. 9. The memory layout for 2D arrays is similar, only k-dimension is removed. With coalesced memory access, the processing time is reduced from *52.8 ms* to *43.5 ms*.

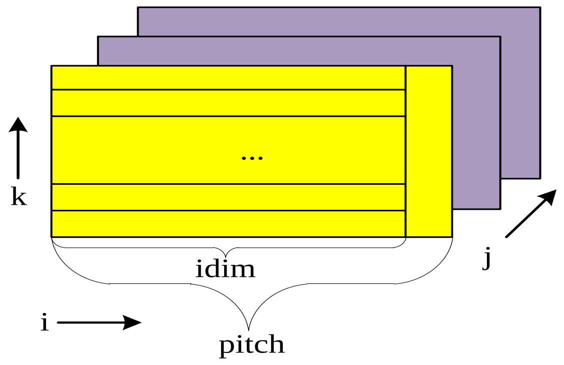


Fig. 9. The memory layout of 3D-arrays.

The first GPU/CUDA based WSM5 microphysics (in WRFv3.1 version) was developed by Michalakes *et al*. of NCAR with a nearly *10x* speedup on NVIDIA's 8800 GTX. Later a team in Alexandria University developed WRF v3.2 version of WSM5 microphysics [33] with a significant improvement in speedup. For fair comparison, we ran both our WSM5 GPU/CUDA code and Alexandria team's WSM5 GPU code on our NVIDIA GTX 590 GPU. As seen from speedup results in Fig. 10 our implementation of WSM5 on a GPU is faster than the other implementation because we scalarized more temporary arrays and used coalesced memory access pattern for global memory.

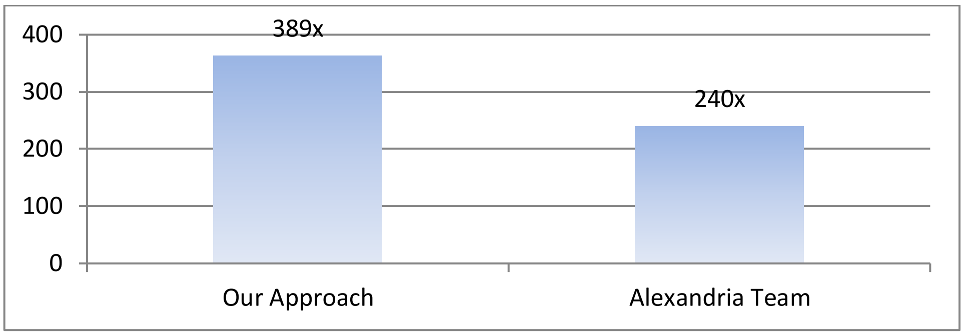
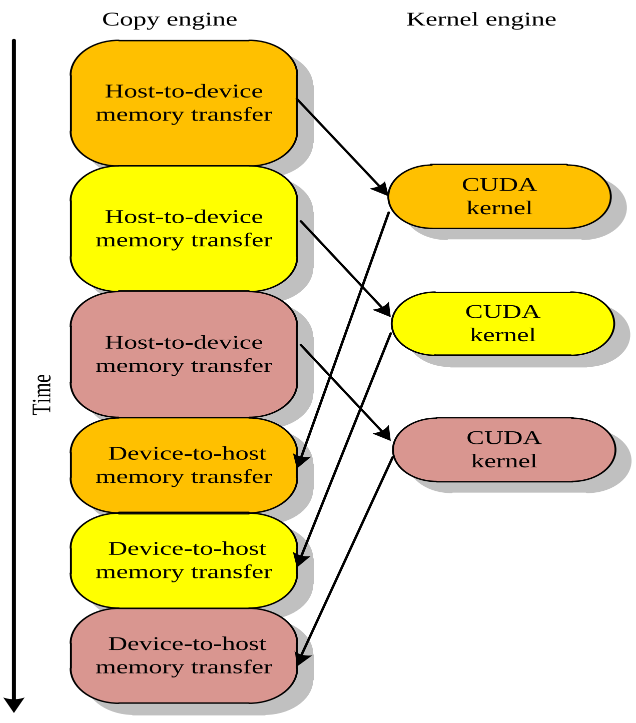


Fig. 10. Speedups for WSM5 without I/O.

4.5 Asynchronous data transfer

The other major difference between C and CUDA C version is the additional data transfer between CPU and GPU in CUDA C. In order to have simultaneous CUDA kernel execution and data transfer, we utilized streams. Stream can be seen as a command pipeline, which executes commands in first-in-first-out (FIFO) order. A diagram depicting the execution timeline of the WSM5 utilizing asynchronous data transfer to overlap CUDA kernel with data transfer is shown in Fig. 11.



Copy engine
Kernel engine
Time
Host-to-device memory transfer
Host-to-device memory transfer
Host-to-device memory transfer
Device-to-host memory transfer
Device-to-host memory transfer
Device-to-host memory transfer
CUDA kernel
CUDA kernel
CUDA kernel

Fig. 11. Execution timeline of the WSM5 computation

The three streams operating at the same time are colored in orange and yellow and purple, respectively. During each kernel call to CUDA kernel, several rows of data are processes by the kernel. In Fig. 12, processing times are shown are a function of the number of rows. The presented processing times are without coalesced memory access. With coalesced memory access, the runtime is *106.5 ms* compared to *95.1 ms* without coalesced memory access. This anomaly is related to the current state of the NVIDIA device drivers and the behavior of the memory transfers of data allocated for coalesced memory access can change in the future.

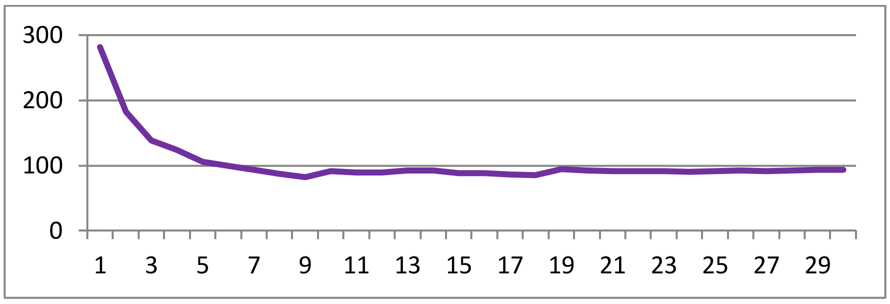


Fig. 12. Processing times [ms] for 1 GPU as a function of the number of rows transferred from CPU to GPU at a time.

The optimal number of rows is *9*. The speedups obtained for our implementation and the older GPU implantation are shown in Fig. 13. Our implementation is faster in this case as well.

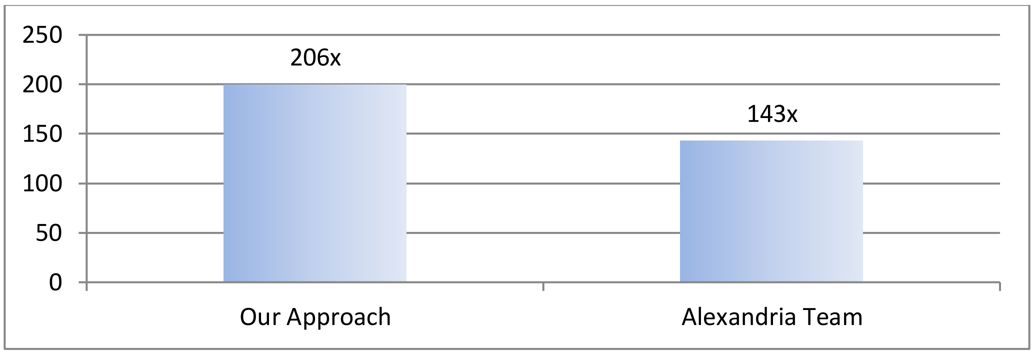


Fig. 13. WSM5 speedups with I/O.

4.6 Multi-GPU implementation

Multi-GPU implementation of WSM5 operates by computing several continuous j-dimension values in the arrays in the same GPU. For 2-4 GPUs the optimal number of j-dimension to transfer from CPU to GPU are *18*, *10* and *13*, respectively. This drop in the number of rows is due to the fact that PCIe bus is becoming more heavily utilized as more GPUs are using it. Thus, smaller amount of data transfer is more optimal.

A set-up for two GTX 590 cards is shown in Fig. 14. In this set-up, there are two GPUs on one GPU card for total of four GPUs. When two GPUs on the same GPU card are transferring data at the same time, the total maximum bandwidth is almost the same as if only one GPU is transferring data.

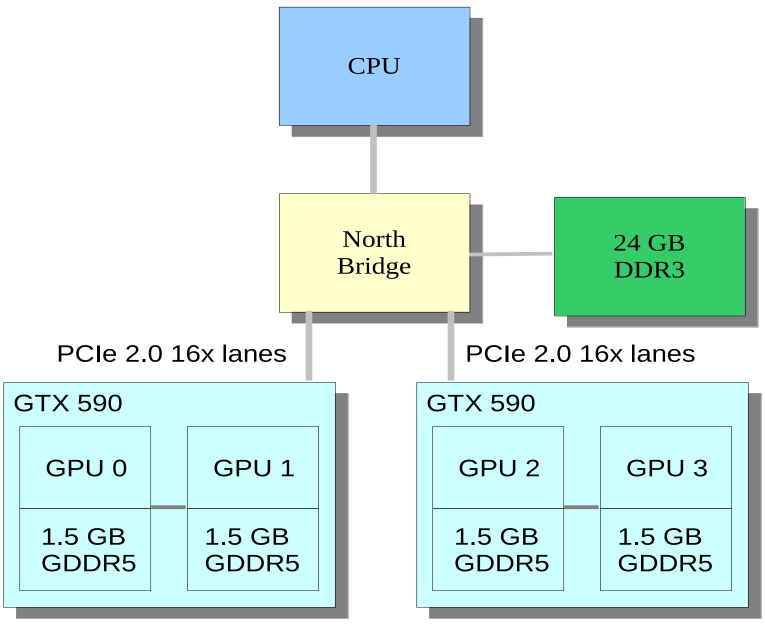


Fig. 14. Schematic diagram of multi-GPU setup

Table 4 lists the computation times for multi-GPUs. The corresponding speedups are depicted in Fig. 16. In the tests, when 2 GPU are used they are from the different GPU cards. Thus, the GPUs are using separate GPU links and are not saturating the same link. Without I/O, the speedup increases linearly with the number of GPU. Thus, 4 GPUs have a speedup of *1556x*.

Table 4. Processing times with I/O for multi-GPU setup

Number of GPUs	Processing time [ms]
1	82.1
2	50.3
4	47.3

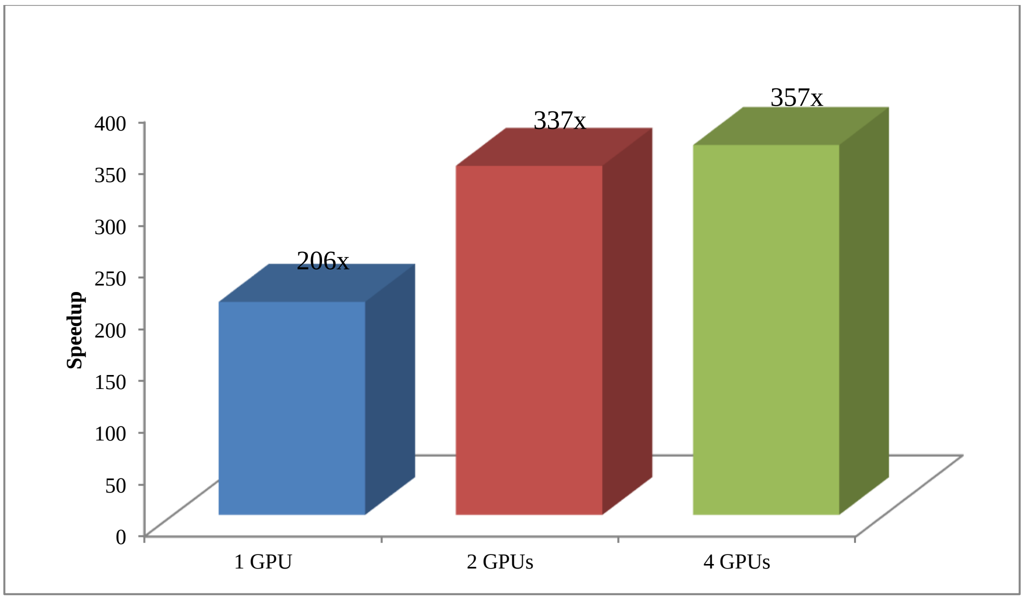


Fig. 15. WSM5 microphysics GPU speedups with I/O over serial CPU implementation.

Microphysics computation is just a small part of the whole WRF model. Given that we have very successfully GPU accelerated WSM5 microphysics scheme, we will continue to convert other WRF modules to CUDA C programming model. Once whole WRF has been rewritten using CUDA C, there will be no need to implement input/output transfers between CPU and GPU for each individual module. Instead, only inputs to the first WRF module have to be transferred from CPU to GPU. Similarly, only the outputs from the last WRF module will be transferred from GPU to CPU.

Therefore, we expect that the speedup for WSM5 in WRF GPU will be closer to results presented in this paper for speedups without I/O than with I/O. The future work includes evaluating Message Passing Interface (MPI) for connecting GPU workstations to form a GPU cluster for WRF computation. Also, other many-core programming environments in addition to NVIDIA's CUDA will be considered.

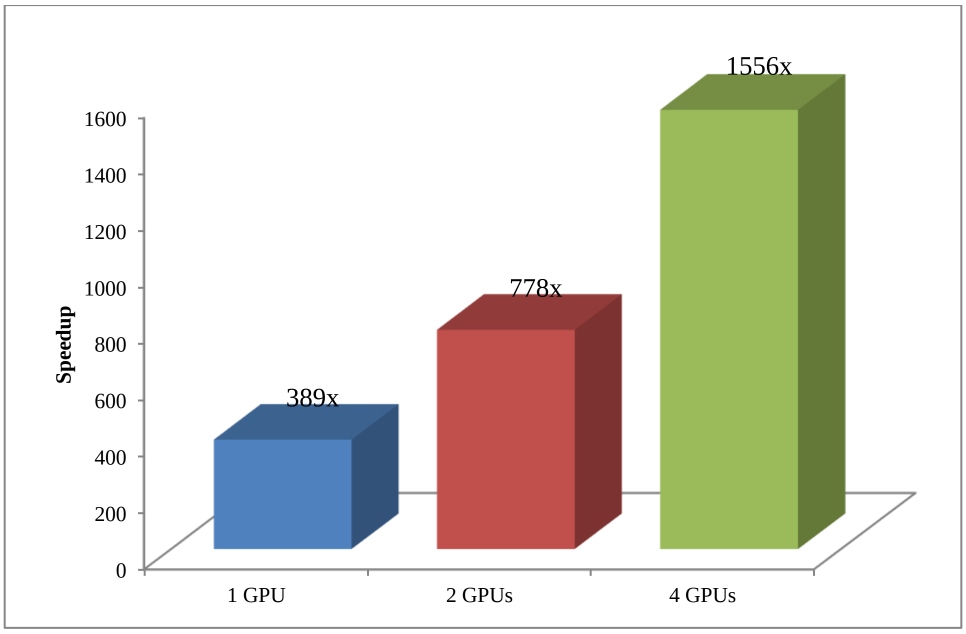


Fig. 16. WSM5 microphysics GPU speedups without I/O over serial CPU implementation.

5. CONCLUSIONS

In this paper, we have presented results for GPU accelerated WSM5 microphysics scheme, which is extremely well suited for implementation on a graphics processor. The implementation achieves *206x* and *389x* speedups with and without I/O using a single GPU as compared to the Fortran implementation running on a CPU. These results represent a *60%* improvement over the earlier GPU accelerated WSM5 module. The improvements over the previous GPU accelerated WSM5 module were numerous. Some minor improvements were that we scalarized more temporary arrays and compiler tricks specific to Fermi class GPU were used. The most important improvements were the use of coalesced memory access pattern for global memory and asynchronous memory transfers. Using 4 GPUs, the speedup was increased to *357x* with I/O. Without I/O the speedup increases linearly with the number of GPUs. Thus, 4 GPUs have a speedup of *1556x*. Overall, the GPU version of WSM5 microphysics scheme was successful and achieved a significant performance increase as compared to the original CPU implementation. Thus, we believe that GPU computing is a viable way to accelerate WRF computation.

ACKNOWLEDGEMENTS

This work is supported by the National Oceanic and Atmospheric Administration (NOAA) under Grant No. NA10NES4400013.

REFERENCES

[1] C. Ahrens, *Meteorology Today*, 9th edition.

[2] R. Bradley, P. Jones, “Climate Since A.D. 1500,” Routledge, 1992.

[3] J. Nickolls, W. J. Dally, "The GPU Computing Era," *IEEE Micro*, 30(2), 56-69 (2010), doi:10.1109/MM.2010.41.

[4] W. Hwu (ed.), GPU Computing Gems Emerald Edition.

[5] W. Hwu (ed.), GPU Computing Gems Jade Edition.

[6] T. Balz, U. Stilla, "Hybrid GPU-Based Single- and Double-Bounce SAR Simulation," IEEE Transactions on Geoscience and Remote Sensing, vol. 47, no. 10, pp. 3519-3529, DOI: 10.1109/TGRS.2009.2022326.

[7] T. Hobiger, R. Ichikawa, Y. Koyama, T. Kondo, "Computation of Troposphere Slant Delays on a GPU," IEEE Transactions on Geoscience and Remote Sensing, vol. 47, no. 10, pp. 3313-3318. DOI:10.1109/TGRS.2009.2022168.

[8] A. Plaza, Q. Du, Y.-L. Chang, R. L. King, “Foreword to the Special Issue on High Performance Computing in Earth Observation and Remote Sensing,” *IEEE J. Sel, Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 503-507, Sept. 2011, DOI: 10.1109/JSTARS.2011.2163551.

[9] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, B. Huang, "Recent Developments in High Performance Computing for Remote Sensing: A Review," *IEEE J. Sel, Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508-527, 2011, DOI: 10.1109/JSTARS.2011.2162643.

[10] J. Setoain, M. Prieto, C. Tenllado, A. Plaza, and F. Tirado, “Parallel morphological endmember extraction using commodity graphics hardware,” *IEEE Geosci. Remote Sens. Lett.*, vol. 4, no. 3, pp. 441–445, Sept. 2007, DOI: 10.1109/LGRS.2007.897398.

[11] A. Plaza, J. Plaza, and H. Vegas, “Improving the performance of hyperspectral image and signal processing algorithms using parallel, distributed and specialized hardware-based systems,” *J. Signal Process. Syst.*, vol. 61, pp. 293–315, 2010, DOI: 10.1007/s11265-010-0453-1.

[12] A. Paz and A. Plaza, “Clusters versus GPUs for parallel automatic target detection in remotely sensed hyperspectral images,” *EURASIP J. Advances in Signal Processing*, 2010, DOI:10.1155/2010/915639.

[13] B. Huang, J. Mielikainen, H. Oh, and H.-L. Huang, “Development of a GPU-based high-performance radiative transfer model for the infrared atmospheric sounding interferometer (IASI),” *J. Comput. Phys.*, vol. 230, no. 6, pp. 2207–2221, DOI: 10.1016/j.jcp.2010.09.011.

[14] J. Mielikainen, B. Huang, and A. H.-L. Huang, “GPU-accelerated multi-profile radiative transfer model for the infrared atmospheric sounding interferometer,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens. (JSTARS)*, vol. 4, no. 3, pp. 691–700, Sept. 2011, DOI: 10.1109/JSTARS.2011.2159195.

[15] J. Mielikainen, R. Honkanen, B. Huang, P. Toivanen, and C. Lee, “Constant coefficients linear prediction for lossless compression of ultraspectral sounder data using a graphics processing unit,” *J. Applied Remote Sens.*, vol. 4, no. 1, pp. 751–774, 2010, DOI:10.1117/1.3496907.

[16] C. Song, Y. Li, and B. Huang, “A GPU-accelerated wavelet decompression system with SPIHT and Reed-Solomon decoding for satellite images,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens. (JSTARS)*, vol. 4, no. 3, pp. 683–690, Sept. 2011, DOI: 10.1109/JSTARS.2011.2159962.

[17] S.-C. Wei and B. Huang, “GPU acceleration of predictive partitioned vector quantization for ultraspectral sounder data compression,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens. (JSTARS)*, vol. 4, no. 3, pp. 677–682, Sept. 2011, DOI: 10.1109/JSTARS.2011.2132117.

[18] S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. ichi Habata, M. Yokokawa, H. Itoh, and K. Otsuka, “A 26.58 tflops global atmospheric simulation with the spectral transform method on the earth simulator,” *Proceedings of the ACM/IEEE conference on Supercomputing*, pp. 1–19, Sept. 2002, DOI: 10.1109/SC.2002.10053.

[19] J. Dukowicz, R. D. Smith, and R. Malone, “A reformulation and implementation of the bryancox- semtner ocean model on the connection machine.,” *Atmos. Ocean. Tech*., vol. 10, pp. 195–208, 1993.

[20] S. W. Hammond, R. D. Loft, J. M. Dennis, and R. K. Sato. “Implementation and performance issues of a massively parallel atmospheric model,” *Parallel Computing*, vol. 21, 1593–1619, 1995.

[21] Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Barker, D. M., Duda, M. G., Huang, X.-Y., Wang, W. and Powers J. G., “A Description of the Advanced Research WRF Version 3,” *NCAR Technical Note*, NCAR/TN-475+STR.

[22] J. Michalakes, M. Vachharajani, "GPU acceleration of Numerical Weather Prediction," Parallel Processing Letters, vol. 18, no. 4, pp. 531-548, 2008, doi: 10.1142/S0129626408003557.

[23] G. Ruetsch, E. Phillips and M. Fatica, "GPU acceleration of the Long-Wave Rapid Radiative Transfer Model in WRF using CUDA Fortran," *Many-Core and Reconfigurable Supercomputing Conference*, (2010), url: http://www.pgroup.com/lit/articles/nvidia_paper_rrtm.pdf.

[24] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, S., "An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code," *International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-11, 2010, DOI: 10.1109/SC.2010.9.

[25] V.T. Vu, G. Cats, L. Wolters, "GPU acceleration of the dynamics routine in the HIRLAM weather forecast model," *International Conference on High Performance Computing and Simulation (HPCS)*, pp. 31-38, 2010, DOI: 10.1109/HPCS.2010.5547152.

[26] M. W. Govett, J. Middlecoff and T. Henderson, "Running the NIM Next-Generation Weather Model on GPUs," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 792-796 2010, DOI: 10.1109/CCGRID.2010.106.

[27] B. Huang, J. Mielikainen, H. Oh and H.-L. Huang, "Development of a GPU-based High-Performance Radiative Transfer Model for the Infrared Atmospheric Sounding Interferometer (IASI)," *Journal of Computational Physics*, vol. 230, no. 6, pp. 2207-2221, 2011, DOI:10.1016/j.jcp.2010.09.011.

[28] J. Mielikainen, B. Huang, and H.-L. Huang, "GPU-Accelerated Multi-Profile Radiative Transfer Model for the Infrared Atmospheric Sounding Interferometer", *IEEE J. Sel, Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 691-700, Sept. 2011, DOI:10.1109/JSTARS.2011.2159195.

[29] Hong, S., Dudhia, J., Chen, S.-H., "A Revised Approach to Ice Microphysical Processes for the Bulk Parametrization of Clouds and Precipitation," *Monthly Weather Review*, vol. 132, no. 1, pp. 103-120, 2004.

[30] Hong, S.-Y. and Lim., J.-O. J., "The WRF Single-Moment 6-Class Microphysics Scheme (WSM6)," *Journal of the Korean Meteorological Society*, vol. 42, no. 2, pp. 129-151, 2006.

[31] Sanders, J., Kandrot, E., *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Pearson Education, Inc. Boston, MA (2010).

[32] http://www.mmm.ucar.edu/wrf/WG2/bench/Bench_V3_20081028.htm

[33] http://www.mmm.ucar.edu/wrf/WG2/GPU/WSM5.htm