Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 1137–1141
DOI: 10.15439/2023F7199
ISSN 2300-5963 ACSIS, Vol. 35

Runge-Kutta Method and WSM6 Microphysics for Weather Prediction on Hybrid Parallel Platform

Hércules Cardoso da Silva
Faculty of Computer Science
Federal University of MS
Campo Grande, Brazil
hercules.cardoso.silva1@gmail.com

Marco A. Stefanes
Faculty of Computer Science
Federal University of MS
Campo Grande, Brazil
marco@facom.ufms.br

Vinícius Capistrano
Physics Institute
Federal University of MS
Campo Grande, Brazil
vinicius.capistrano@ufms.br

***Abstract*—In Numerical Weather Prediction (NWP) models we need to model the dynamics of the atmosphere and the physical variables that take place in a moment. In grid point models the temporal evolution of the model variables are calculated in a 3D grid which covers the atmosphere from the surface up to the model top. NWP models include two import routine namely Runge-Kutta method and microphysics scheme WSM6. This paper describes advances in the performance of the Runge-Kutta method and WSM6 microphysics by exploiting multi-level parallelism using CUDA-based GPU on hybrid parallel platform. We applied pipeline parallelism technique, workload balancing and asynchronous data exchange strategy each demonstrating be useful to improve performance. Our experiments show that the solution is scalable. We also realized an analysis of the accuracy of our implementation with good results.**

***Index Terms*—Runge-Kutta Method, WSM6 microphysics, Hybrid Parallel Algorithms, Multi-GPU Algorithms, High performance Computing**

I. Introduction

WEATHER prediction systems assess atmospheric changes. They have diverse uses, from aiding agriculture, aviation, navigation, to assisting in natural disaster prevention. Weather prediction, a complex system, requires modeling atmospheric dynamics and physical variables such as pressure, temperature, wind, water vapor, clouds, and precipitation. Outputs typically include future temperature, humidity, and rainfall based on initial conditions. Some small-scale processes, like cloud formation, which can't be resolved numerically, are incorporated into models through parameterization schemes.

There are several numerical prediction systems based on sets of physics-based equations which weather can be predicted from atmospheric data as temperature, radiation, air pressure, wind speed, wind direction, humidity, and rainfall, and how they behave in the atmosphere [1], [2].

Among them, Model for Prediction Across Scales (MPAS) [3], Global/Regional Assimilation and PrEdiction System (GRAPES) [4] and Weather Research Forecasting (WRF) model [5] are popular tools used for both research and operational purposes. The Runge-Kutta third order method (RK3), used to time integration, and the Single-Moment 6-Class Micro-physics (WSM6), which calculates several hydrometeors variables, are tasks present in important models of weather prediction. Moreover, they are the most time-consuming tasks in a weather forecasting system.

The main goal of this paper is to present some enhancement for the RK3 and for the WSM6 using multi-level parallelism and pipelining on hybrid parallel platform. Our tests show that we achieved speedup ranging from 10 to 39 for the RK3 and from 5 to 26 for the WSM6 when compared to a 12-thread CPU. The pipelining and asynchronous data exchange strategies improved the runtime by up to 37%.

II. Background and related Works

There are several works that implement RK3 method and the WSM6 using parallel platforms. Korch and Rauber [6] investigated the RK3 method They compare the RK3 implementation using MPI, Pthreads and Java on Sun SMP and on Cray T3E. The speedup ranged from 3 to 10 with 8 processors. The authors of [7] show an enhancing of twelve-fold of a CUDA RK4 implementation on a Tesla compared to an OpenMP implementation. Murray [8] describes an OpenMP RK4 implementation for physics simulation that achieved a speedup of 2 on a 4-core CPU. Wo *et all* [9] achieve a speedup of 2 on a Geforce GT450M. The authors of [10] achieved a gain of 45 times on 2x Xeon Phi versus the same implementation using OpenMP on 2x Intel Xeon.

An implementation of WSM6 on CUDA [11] obtained a speedup of up to 246 using a K40 GPU and up to 295 using two GPUs when compared to a single-threaded CPU. Kim *et al* [12] implemented the WSM6 using OpenACC for Model for Prediction Across Scales (MPAS) [1]. They achieved a speedup of 5.7 running on a V100 GPU compared to a multi-thread version executed on 48 CPUs. More recently, Silva *et al* [13] improved those results and reached speedup of 371 using four V100 GPUs compared to single-thread CPU e 108 fold compared to 24-thread CPU.

III. Weather Prediction: RK3 and WSM6

The dynamic core of an NWP is responsible for discretization in space. Variables such as temperature, pressure, wind, humidity are time integrated by this dynamic core model. RK3 method is responsible for this time integration. However, many processes cannot be spatially discretized. Therefore, these processes are parameterized in terms of other variables available

in the a model time-step state. Among these processes is the microphysics scheme.

The equations of the Runge-Kutta are formulated using the pressure vertical coordinate $\eta = (P_p - P_{ht})/\mu$, where $\mu = (P_{hs} - P_{ht})$, P_p is the hydrostatic component of the pressure, P_{hs} and P_{ht} are representing the pressure at surface and top boundaries. The transport equations in the flux form can be written as:

$$\frac{\partial \mu_d \phi}{\partial t} = -\nabla_\eta \cdot \mu_d \mathbf{v_h} - \frac{\partial \mu_d \omega \phi}{\partial \eta} \tag{1}$$

The quantity μ_d is the mass of the dry-air column (between the bottom and the top of the atmosphere for a vertical coordinate of mass type $s = [\pi_d - (\pi)_t]/\mu_d$, with $\mu_d = \partial \pi_d / \partial s$. Moreover, $\mathbf{v_h} = (u, v, \omega)$, which is the zonal, meridional and covariant vertical velocities, respectively. $\phi = (u, v, w, \theta, q_m)$, where w is the vertical movement, θ is the potential temperature, and q_m represents the scalar quantities, such as water vapor, hydrometeors and aerosol mixing ratio.

The WRF use third order Runge-Kutta for temporally discretized [14], which can be described by the equations:

$$\Phi^* = \Phi^n - \frac{\Delta t}{3} \nabla_\eta \cdot (\mu_d \mathbf{v_h} \phi)^n - \frac{\Delta t}{3} \frac{\partial (\mu_d \omega \phi)^n}{\partial \eta} \tag{2}$$

$$\Phi^{**} = \Phi^n - \frac{\Delta t}{2} \nabla_\eta \cdot (\mu_d \mathbf{v_h} \phi)^* - \frac{\Delta t}{2} \frac{\partial (\mu_d \omega \phi)^*}{\partial \eta} \tag{3}$$

$$\Phi^{n+1} = \Phi^n - \Delta t \nabla_\eta \cdot (\mu_d \mathbf{v_h} \phi)^{**} - \Delta t \frac{\partial (\mu_d \omega \phi)^{**}}{\partial \eta} \tag{4}$$

where $\Phi = \mu_d \phi$.

Now, the WSM6 scheme is based on six classes of cloud particles: water droplets, ice crystals, snow, hail, aggregates of ice crystals, and raindrops. In an NWP system, the WSM6 is a module that is coupled with other parameterization schemes which are described by differential equations that can be seen in more detail by Hong *et al* [15]. The WSM6 also considers the interactions between the different classes of hydrometeors.

IV. Hybrid Parallel platform

A Hybrid Parallel platform consists of a set of computer nodes which each node has a multi-core CPU and a many-core GPU. Each CPU as well GPU in the set can be heterogeneous. Under this model, we use MPI [16] to dispatch two threads by nodes using massage passing. One of them containing the code that will be executed on the multi-core CPU using OpenMP [17] and another thread containing CUDA [18] code that will be executed on GPU.

V. Multi-level Parallelism

Our weather model enables to use various levels of parallelism. The first level employs the MPI for coordinating various computer nodes. This level helps the task distribution across several nodes and their intercommunication, allowing concurrent execution of tasks. The second level is the division of tasks between different CPUs using shared memory. The last level is the use of CUDA-based GPUs to accelerate the processing. Each GPU executes the pipeline, which allows the model to run faster and more efficiently.

In our implementation of the method RK3 on CUDA, we performed a reorganization of the original code due to the different features between the Fortran compiler and CUDA. The first step in the implementation process involved consolidating all the subroutines into a single subroutine. The main goal of this initial stage is to identify sections that can be optimized and subsequently translated into CUDA. This strategy also helps avoid unnecessary copying of data from the CPU to the GPU. Besides, the consolidation of subroutines contributes to reducing the number of kernel calls.

Our CUDA WSM6 employs parallel optimization techniques similar to those used in the RK3. However, there is a distinction between the two implementations: WSM6 exhibits lower dependency on its operations. This feature enables the WSM6 to be executed in parallel over different portions of the data. We leverage this inherent characteristic of microphysics scheme to execute multiple WSM6 kernels simultaneously. This facilitates a better workload balance, improving distribution of tasks. Moreover, we distribute a workload to each node proportionally to its computational power.

A. Parallel Runge-Kutta

Take as input a grid of $nx \times ny \times nz$ of points which are the starting points of RK3, a function $f'(x, y)$ and a timestamp h. Moreover, it is given the computational power of each node and an integer N representing the total number of steps.

1. - Let T_f be the total of Gflops of the platform and let T_i be the computational power of the compute node i. Send to node i $(nz * zy * nz) * (\frac{Ti}{Tf})$ points.
2. - If node i is a CPU then perform the following steps:

2.1. Suppose we have t threads so that each thread computes $((nz * zy * nz) * (Ti/T_f))/t$ elements:
Compute the new x and y based on Eq. 2, 3, and 4.

3. - If node i is a GPU then perform the following steps:

3.1. Let v be an array of points attributed to i. Transfer array v to global memory of the device.
3.2. Invoke the kernel RK3 with $((k_1 + k_2 + 4k_3)/6)/32$ blocks and 32 threads per block.
3.3. Set $e = blockIdx.x * blockDim.x + threadIdx.x$;
3.4. For n_i from 0 to $N - 1$ do:
Compute the new x and y based on Eq. 2, 3, and 4.

B. Parallel WSM6

We developed the CUDA WSM6 to support multi-level parallelism and pipelining. Each equation responsible for calculating the transition from one hydrometeor to another (as described in details by [15]) can to be computed in parallel.

Let p be a partition of the domain points with $x \times y \times z$ points according to Fig. 1.

1. For $i = 0$ to x, do:

1.1. Let the set of blocks $B = (b_0, b_1 ... b_y)$ be such that each $b_i \in B$ has z threads.
Each thread $t \in b_i$ computes the hydrometeors droplets

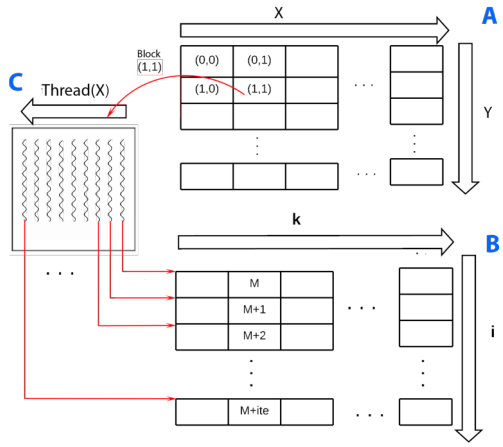

Fig. 1. GPU memory access of the M(k, i): (A) GPU blocks organized in two dimensions; (B) Matrix M(k, i), (C) A threads block of the (A) where each block thread accesses the matrix den with coalesced memory access.

(qc), ice particles (qi), rain droplet (qr), snow crystal (qs), graupel (qg) and water vapor (qv) as folows: Let k be the thread index and j the block index. For each (qc, qi, qr, qs, qg, qv) in $p(k, i, j)$ compute the transition considering the variables of that point.

VI. Pipeline Parallel

Pipelining is extensively utilized by hardware designers. Enhancement is accomplished by dividing each processed instruction into multiple stages. Below we describe how we apply the pipeline technique at the software layer, where the domain to be computed on the GPU is divided into partitions. Each partition represents a stage, allowing parallel processing.

We note that there are two main tasks of system working alternately, namely the dynamic RK3 and the microphysics WSM6. Each task is implemented in a different kernel. In this approach, as soon as a kernel finishes its job another kernel is launched. Hence, during some time the GPU is idle or wasting time with data exchange and, as result, reducing efficiency.

Firstly we identify data used by both RK3 and WSM6 and that may remain within the GPU memory during two consecutive call of different kernels avoiding data exchange.

Additionally, we noted that the idle GPU is the most important bottleneck to achieve better performance. To solve this problem the pipeline parallelism could be a interesting strategy. So, each part of the domain attributed to a GPU is divided into small partitions such that can be processed in parallel by the pipeline stages. Once a stage completes the WSM6 processing for its partition, that partition can start the task RK3 on it corresponding to the next stage in the pipeline. Similarly, as the next stage completes its RK3 processing for its partition it starts the WSM6 processing.

Each GPU executes its own pipeline independently. For clarity of presentation let us assume we have a two-stage pipeline. In this version we split the domain points given as input in two partitions p_1 and p_2: The RK3's and WSM6's kernels were implemented in such a way as to allow splitting the entry points into two partitions. Thus, we are able to execute RK3 with input p_2, while simultaneously, on the same GPU, we execute WSM6 with input p_2.

VII. Multi-stages Pipeline RK3 WSM6

Using the same ideas of the previous section we may divide the domain in n blocks. Hence, we can generalize the pipelining for $2n$ stages considering n partitions of points.

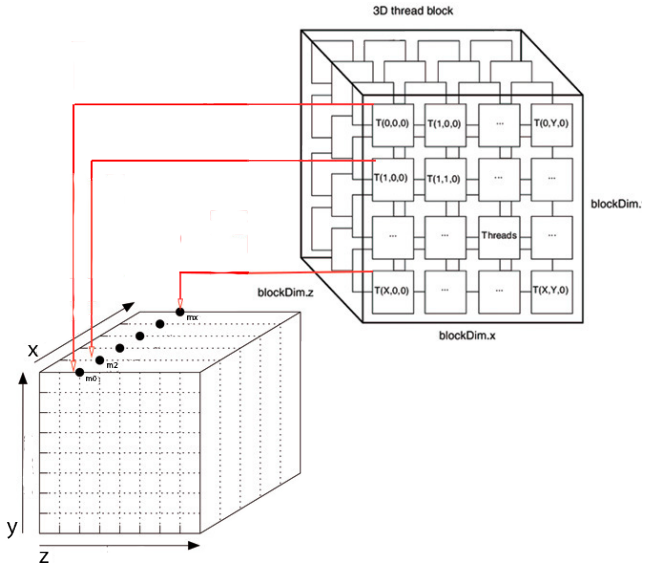

Fig. 2. GPU Memory access of the M(x, y, z): The thread (x, y, z) accesses the rank (x, y, z) of the M such that if the rank in the global memory of the element (x, y, z) is m, the rank of the element $(x + 1, y, z)$ will be m+1.

The second step in implementing involves properly modeling the data structures to ensure coalesced access to the GPU's memory. This step is performed both in loops operating on three-dimensional (3D) portions of the domain and in two-dimensional (2D) portions. The way to map threads and blocks depends on the type of loop and its interaction with the data structure. In general, the structures are mapped as follows.

For 3D loops, threads in a block are mapped to a contiguous subset of elements in a specific dimension as can be seen in Figure 2, blocks are mapped to different portions of the 3D domain and for 2D loops, threads in a block are mapped to a contiguous rectangular region of the data in two dimensions, blocks are mapped to different sections of the 2D domain. The goal is to maximize coalesced access to the GPU's memory, allowing threads to simultaneously access data, taking advantage of the CUDA architecture.

We employed an optimization involving the use of constant memory and texture memory to variables and small arrays that remain constant during the execution. By storing these constant values in dedicated memory, we can benefit from their optimized access and reduce the memory traffic.

Shared memory is a valuable resource on GPUs. It is notably more limited than GPU global memory and is visible to threads

within the same block. Due to this feature, we chose to rub this implementation with a configuration that prioritizes a wider cache over shared memory. Shared memory is firstly used for some loops that allow for blocking optimizations. Moreover, it is used to store some intermediate results and, when possible, loop invariants. This strategy maximizes the efficiency of shared memory utilization.

Our second implemented optimization is loop fusion. Since loops often iterate over the same data, they can be combined into a single loop without compromising the accuracy, depending on the operation. Along with loop fusion, we also strive to organize memory accesses to the same structure as closely as possible in the code whenever feasible.

Another optimization we used involved rewriting code snippets related to floating-point operations. It's important to exercise caution when applying this type of optimization to avoid introducing discrepancies between the CPU and GPU code. Taking care, we ensure that our CUDA implementation is both accurate and efficient.

Our last optimization creates a pipeline that splits RK3 processing into up to four stages. It works as follows: initially, a CPU thread is assigned for each stage and a master thread oversees operations. At the start, all threads are blocked. The master selects the thread responsible for processing the initial data and enters a busy-wait state. Once the kernel signals that more data can be processed, the master unblocks a thread. This process continues until no CPU threads are blocked. When a thread finish, it chooses a new data block and blocks itself, waiting for the master to unblock it.

A. WSM6-RK3-Pipeline

The joint execution of the RK3 pipeline and the multi-kernel implementation of WSM6 enables the creation of an integrated pipeline that performs the combined computation of both tasks. This implementation resembles the thread management described in the RK3 pipeline. Each of these stages is assigned to a thread on CPU. This mechanism has a similar principle to the RK3 pipeline. This case requires monitoring two distinct CUDA streams, one for RK3 and another for WSM6. Moreover, it is crucial to check for dependencies that arise between the stages of WSM6, thus ensuring proper synchronization between stages.

VIII. Experimental results

We performed a two-domains WRF simulation for March 4th of 2021, starting at 00UTC with duration of 12h. See Fig. 3 for details. The set of physical parameterization used in the simulation was: (a) WSM6 microphysics [15]; (b) longwave and (c) shortwave radiation schemes of Community Atmospheric Model 3 (CAM) [19]; (d) planetary boundary layer scheme of Yonsei University [20]; and, (e) Kain-Fritsch for cumulus parametrization [21].

A. Experiments

In our tests we first measured the time of the RK3 and WSM6 separately. After that, we measured the overall system performance considering both the stages.

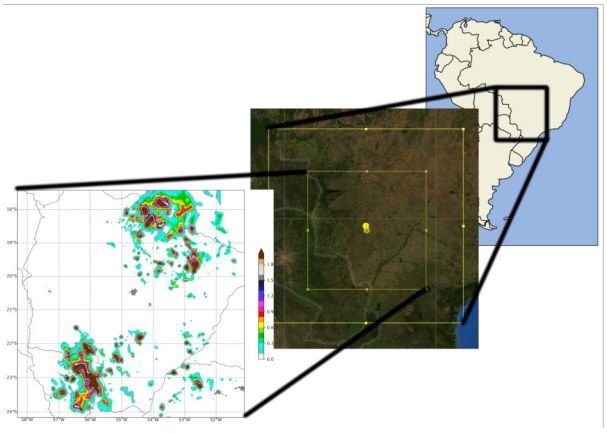

Fig. 3. Domains of the WRF simulation with 12 km of horizontal resolution: X-axis and Y-axis represent latitude and longitude, respectively. The 4 km horizontal resolution domain is represented by the highlighted square in the center of the 12-km resolution domain. The model is set to 34 vertical levels. Colors represent the vertical levels sum for q_c, q_i, q_r, q_s and q_g.

TABLE I
Time and acceleration of the RK3 and WSM6

	RK3		WSM6		WRF
Hardware	time(s)	speedup	time(s)	speedup	Total
12-thread-CPU	1335	-	2956	-	6156
1xP100	125	10.2	520	5.6	2857
1xP100+1xk80	96	13.9	436	6.7	2532
1xP100+2xk80	77	17.3	280	10.5	2410
2xP100	71	18.8	225	13.1	2337
1xP100+4xk80	53	25.2	169	17.4	2009
2xP100+4xk80	34	39.3	112	26.3	1776

With the CUDA-RK3, we achieved a speedup of 10.6 using one GPU P100 and 39.2 using 2x P100 + 4x k80 when compared to 12 thread version (see Table I). we also see the run-time and speedup of the CUDA-WSM6. We achieved speedups ranging from 5.6 on one GPU P100 to 26.3 on the 2x P100 + 4xk80 set. We note we obtained increasing speedup consistently with the growth of computational power.

TABLE II
Time (in seconds) of the RK3+WSM6 with pipeline stages

stages RK3+WSM6	1	2	4	8	16
12-Thread-CPU	6156	6156	6156	6156	6156
1x P100	3023	2802	2525	2420	2318
1x P100 + 1xk80	2736	2445	2212	2116	2048
1x P100 + 2xk80	2614	2343	2114	2022	1979
2x P100	2535	2214	1998	1911	1828
1x P100 + 4xk80	2213	1892	1707	1633	1502
2x P100 + 4xk80	1980	1698	1523	1463	1448

Table II show the overall performance when we increase the computational power(in each column). Besides, we notice the time gain when varying the number of pipeline stages.

Besides we apply the technique of overlapping transfer and processing through the use of asynchronous transfers. This enabled performance gains as can be seen in the Table III.

TABLE III
TIME (IN SECONDS) OF THE WSM6+RK3 WITH OVERLAPPING

stages RK3+WSM6	1	2	4	8	16
Ryzen 1600 12 Ths	6156	6156	6156	6156	6156
1x P100	2857	2613	2315	2216	2079
1x P100 + 1xk80	2532	2242	2009	1913	1844
1x P100 + 2xk80	2410	2140	1909	1818	1774
2x P100	2337	2014	1869	1728	1612
1x P100 + 4xk80	2009	1690	1502	1430	1300
2x P100 + 4xk80	1776	1493	1319	1260	1246

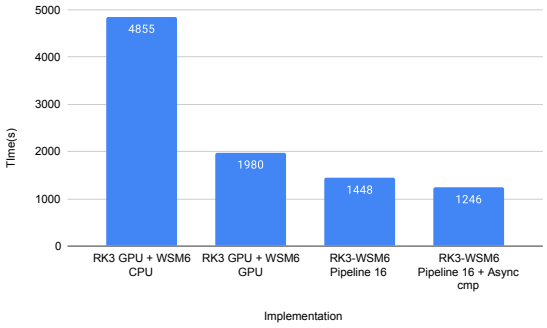

Fig. 4. Runtime of the implementation on platform 2xP100 + 4xK80.

Fig. 4 show the result using our load balance heuristic. We see that with each new device added, we have a performance gain nearly proportional to the increase in computational power, so that our implementation proves to be scalable.

B. Accuracy Analysis

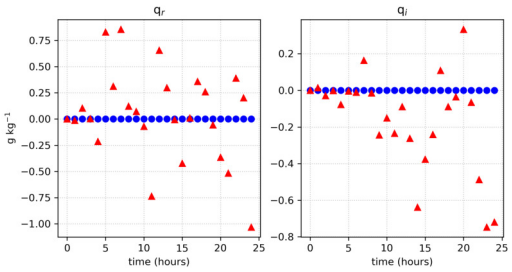

Fig. 5. The bias of the vertical profile in relation to the CPU simulation for mixing ratio of rain droplets (q_r) and ice particles (q_i) from 00UTC on March 4th to 00UTC on March 6th, 2021. The x-axis is in g kg^{-1} and the y-axis is in σ vertical coordinate. The blue and red markers are related GPU arithmetic optimizations off and GPU arithmetic optimization on, respectively.

To measure the accuracy of our GPU implementation, we used the results of the CPU implementation as a reference and analyzed the deviations of the GPU solution. We examined the results of the solution with the configuration 2xP100+4xK80.

The Fig. 5 show the time biases. We illustrate the output hydrometeors ice particles and rain droplets to compare the optimization levels. We can note a deviation between the GPU and the CPU implementation. We analyzed a version of the GPU implementation with enabled optimizations and another without optimizations. Note that when these optimizations are disabled, the deviation from the CPU implementation is negligible. When they are enabled, some points show a reasonable deviation compared to the CPU.

REFERENCES

[1] M. R. Petersen, X. S. Asay-Davis, D. W. Jacobsen, M. E. Maltrud, T. D. Ringler, L. Van Roekel, C. Veneziani, and P. J. Wolfram, Jr., "MPAS-Ocean model user's guide version 6.0," OSTI, Tech. Rep., 4 2018.

[2] S. R. Freitas, J. Panetta, K. M. Longo, L. F. Rodrigues, D. S. Moreira *et al.*, "The brazilian developments on the regional atmospheric modeling system (BRAMS 5.2): an integrated environmental model tuned for tropical areas," *Geosci. Model Dev.*, vol. 10, pp. 189–222, 2017.

[3] W. Skamarock, J. Klemp, M. Duda, L. Fowler, S. Park, and T. Ringler, "A multiscale nonhydrostatic atmospheric model using centroidal voronoi tesselations and c-grid staggering," *Monthly Weather Review*, vol. 140, no. 9, pp. 3090–3105, Sep. 2012.

[4] Z. Ma, Q. Liu, C. Zhao, X. Shen, Y. Wang, J. H. Jiang, Z. Li, and Y. Yung, "Application and evaluation of an explicit prognostic cloud-cover scheme in grapes global forecast system," *Journal of Advances in Modeling Earth Systems*, vol. 10, no. 3, pp. 652–667, 2018.

[5] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, M. Barker, K. G. Duda, X. Y. Huang, W. Wang, and J. G. Powers, "A description of the Advanced Research WRF Version 3," National Center for Atmospheric Research, Tech. Rep., 2008.

[6] M. Korch and T. Rauber, "Comparison of parallel implementations of runge-kutta solvers: Message passing vs. threads," in *Parallel Computing - Advances in Parallel Computing*, G. Joubert, W. Nagel, F. Peters, and W. Walter, Eds. North-Holland, 2004, vol. 13, pp. 209–216.

[7] L. Murray, "GPU acceleration of runge-kutta integrators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, pp. 94–101, 2012.

[8] C. Chantrapornchai, S. Dainpraprai, and O. Wongtaweesap, "On the Computer Simulation of Microparticles Capture in Magnetic Filters using OpenMP," *International Journal of Computer Applications*, vol. 51, no. 14, pp. 23–30, Aug. 2012.

[9] M. S. Wo, R. Gobithaasan, and K. Miura, "GPU acceleration of runge kutta-fehlberg and its comparison with dormand-prince method," *AIP Conference Proceedings*, vol. 1605, pp. 16–21, 2014.

[10] B. Bylina and J. Potiopa, "Explicit fourth-order runge—-kutta method on intel xeon phi coprocessor," *International Journal of Parallel Programming*, vol. 45, no. 5, p. 1073–1090.

[11] M. Huang, B. Huang, L. Gu, H. Huang, and M. Goldberg, "Parallel GPU architecture framework for the WRF single moment 6-class microphysics scheme," *Computers & Geosciences*, vol. 83, 06 2015.

[12] J. Y. Kim, J.-S. Kang, and M. Joh, "GPU acceleration of MPAS microphysics WSM6 using OpenACC directives: Performance and verification," *Computers and Geosciences*, vol. 146, p. 104627, Jan. 2021.

[13] H. C. da Silva, M. A. Stefanes, and V. Capistrano, "OpenACC Multi-GPU approach for WSM6 microphysics," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2021, pp. 382–387.

[14] J. B. Klemp, W. C. Skamarock, and J. Dudhia, "Conservative Split-Explicit Time Integration Methods for the Compressible Nonhydrostatic Equations," *Monthly Weather Review*, vol. 135, pp. 2897–2913, 2007.

[15] S. Hong and J. Lim, "The WRF Single-Moment 6-Class Microphysics Scheme (WSM6)," *J. Korean Meteor. Soc.*, vol. 42, pp. 129–151, 2006.

[16] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.0*, Jun. 2021.

[17] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[18] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Elsevier Science, 2012.

[19] W. Collins, P. Rasch, B. Boville, J. McCaa, D. Williamson, J. Kiehl, B. Briegleb, C. Bitz, S. Lin, M. Zhang, and Y. Dai, "Description of the NCAR Community Atmosphere Model (CAM 3.0)," Tech. Rep., 2004.

[20] S.-Y. Hong, Y. Noh, and J. Dudhia, "A New Vertical Diffusion Package with an Explicit Treatment of Entrainment Processes," *Monthly Weather Review*, vol. 134, no. 9, pp. 2318–2341, Sep. 2006.

[21] J. S. Kain, "The Kain–Fritsch Convective Parameterization: An Update," *Journal of Applied Meteorology*, vol. 43, no. 1, pp. 170 – 181, 2004.