Enhancement of *WRF Model* Using CUDA

- Ashish Maharjan, Mechanical Engineer, COAC, Germany & Asish Shakya, Kathmandu Model College, Bagbazar, Kathmandu, Nepal

Abstract:

The main aim of this paper is to implement and run WRF (Weather Research and Forecasting) model on Graphical Processing Unit (GPU) with the help of NVidia's CUDA (Compute Unified Device Architecture) in a normal machine. Without GPU, the model needs high-end systems to be executed smoothly. For this, CUDA code is executed for a particular microphysics module to create an object file which is then added to the WRF model. Later the object file is executed on the GPU with the help of CUDA.

Keywords: compute unified device architecture, graphical processing unit, microphysics, object, weather research and forecasting

Introduction:

Weather Research and Forecasting (WRF) is a next generation meso-scale Numerical Weather Prediction (NWP) system designed to serve both atmospheric research and operational forecasting needs. WRF is an application of meteorology that needs quantitative data of the current atmospheric state [1]. Numerical Weather Forecasting uses the equations of fluid dynamics and thermodynamics for the estimation of future state of the atmosphere. The whole area is converted into a 3D grid by the weather models. Laws of physics, fluid motion and chemistry are used with differential equations to model weather. These models are used to calculate humidity, solar radiation, winds, surface hydrology and heat transfer for each grid cell [2, 3]. WRF allows researchers to produce simulations reflecting either real data (observation, analysis) or idealized atmospheric conditions. It provides operational forecasting a flexible and robust platform. WRF is used for research purposes and real-time forecasting throughout the globe and was first released in the year 2000. WRF is being used by the National Oceanic and Atmospheric Administration (NOAA). WRF is a huge model and takes a lot of time to process using a normal computer. As efficiency of the WRF model is determined by the speed of its execution process, therefore a powerful supercomputer is the best to process and execute the data and forecast smoothly. This problem of WRF model can be solved by running the WRF model on a Graphical Processing Unit (GPU).

By implementing WRF model on GPU the processing speed of WRF on data will be accelerated. GPU is used for parallel computing used heavily on deep learning. The field of computer graphics traces its roots back to the earliest days of computer sciences. HP and IBM were some of the companies to introduce silicon hardware capable of generating imagery. Programmers created graphical software and that software was broken down into a series of instructions that the central processing unit of the computer was capable of carrying out. As the model got complex the

amount of transformation outgrew the capabilities of CPU alone so NVIDIA came up with a first GPU capable of hardware transform and lighting. CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller more efficient cores designed for handling multiple tasks simultaneously.

For the purpose of implementing WRF model on the GPU, NVidia's CUDA has been used. CUDA stands for Compute Unified Device Architecture and is a parallel computing platform and application programming interface model created and developed by NVidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit for general purpose processing – an approach termed GPGPU (General Purpose Computing on a Graphics Processing Unit). The main purpose in our project is to implement WRF model on Graphical Processing Unit (GPU) using NVIDIA's CUDA.

Graphical Processing Unit (GPU)

GPU is a processor which can be programmed which is specially used for image rendering present on the computer's screen. The rapid processing of graphics is provided by the GPU. It was designed especially for in-depth graphics rendering. GPU supports parallel computing, meaning, can do thousands of operations at the same time or huge programs are divided into smaller ones and are executed simultaneously [8]. This parallel computing makes the GPU best for graphics as lightning, textures and rendering are supposed to be done at once so that it can be visualized on the screen of the PC. GPUs are great for rendering and decoding 3D animations and 3D videos. GPUs are also used for 2D data for zooming, etc. [9]. GPUs nowadays are used for Artificial Intelligence applications and for scientific applications as in such applications repetitive computation is required which can be solved by parallel computing. In the old days, graphics rendering was done by the Central Processing Unit (CPU), but when the applications got upgraded, CPU could not handle the load as CPU executes serially therefore GPU was introduced. GPU renders images faster than CPU but CPU performs single executions/calculations much faster than GPU because of its higher clock speed [10]. CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller more efficient cores designed for handling multiple tasks simultaneously. CPU and GPU have different numbers of cores present, processor inside a processor is known as cores which can process its own tasks. CPU is composed of only a few cores with lots of cache memory that can handle a few software threads at a time whereas GPU is composed of hundreds of cores that can handle thousands of threads at once. GPU, in AI, is mostly used in a technology called "deep-learning", deep-learning feeds lots of data into the neural networks and trains the model.

NVidia's CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical

processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs. In GPU-accelerated applications, the sequential part of the workload runs on the CPU – which is optimized for single-threaded performance – while the compute intensive portion of the application runs on thousands of GPU cores in parallel. When using CUDA, developers program in popular languages such as C, C++, Fortran, Python and MATLAB and express parallelism through extensions in the form of a few basic keywords. The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming.

CUDA-powered GPUs also supports programming frameworks such as OpenACC and OpenCL, and HIP by compiling such code to CUDA. The CUDA Toolkit from NVIDIA provides everything needed to develop for GPU-accelerated applications. The CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime. Thousands of applications developed with CUDA have been deployed to GPUs in embedded systems, workstations, data centers and in the cloud. CUDA serves as a common platform across all NVIDIA GPU families. The first GPUs were designed as graphics accelerators, becoming more programmable over the 90s, culminating in NVIDIA's first GPU in 1999. Researchers and scientists rapidly began to apply the excellent floating point performance of this GPU for general purpose computing. In 2003, a team of researchers led by Ian Buck unveiled Brook, the first widely adopted programming model to extend C with data-parallel constructs. Ian Buck later joined NVIDIA and led the launch of CUDA in 2006, the world's first solution for general-computing on GPUs. Since its inception, the CUDA ecosystem has grown rapidly to include software development tools, services and partner-based solutions. CUDA accelerated applications across a wide range of domains from image processing, to deep learning, numerical analytics and computational science.

Since the WRF model can only be installed and executed on Linux operating systems and NVidia's CUDA has been proven to get better results among other GPU's, our experiment considered using Ubuntu as operating system and CUDA as GPU.

Literature Survey:

The Weather Research and Forecasting Model: Overview, System Efforts, and Future Directions.

This paper gives an introduction to WRF Modeling System and the working of the model is explained in detail. The paper also discussed the future of the WRF model and how it is going to advance. From the paper, working on the WRF model, the kind of data required for the model to process was known.

WRF Physics Options

The article gives a brief introduction to all of the Physics options available in the WRF model. The work that the different types of physics options do are also explained

briefly. Jimy Dudhia has explained which option is suited to be used in different types of situations.

As GPU (CUDA) is to be implemented on the physics option of the WRF model, the physics options must be known to us and this article gave a brief idea about all the physics options.

GPGPU PROCESSING IN CUDA ARCHITECTURE [2].

This paper written by Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni and Amit Bawaskar discuss CUDA and its architecture. It took them through a comparison of CUDA C/C++ with other parallel programming languages like OpenCL and DirectCompute. The paper also lists out the common myths about CUDA and how the future seems to be promising for CUDA. As here, CUDA has been used for implementation of WRF model in GPU, from paper [27], CUDA's architecture and why CUDA is preferred over other platforms of parallel programming was known.

Methodology for Implementing WRF Model on GPU using CUDA.

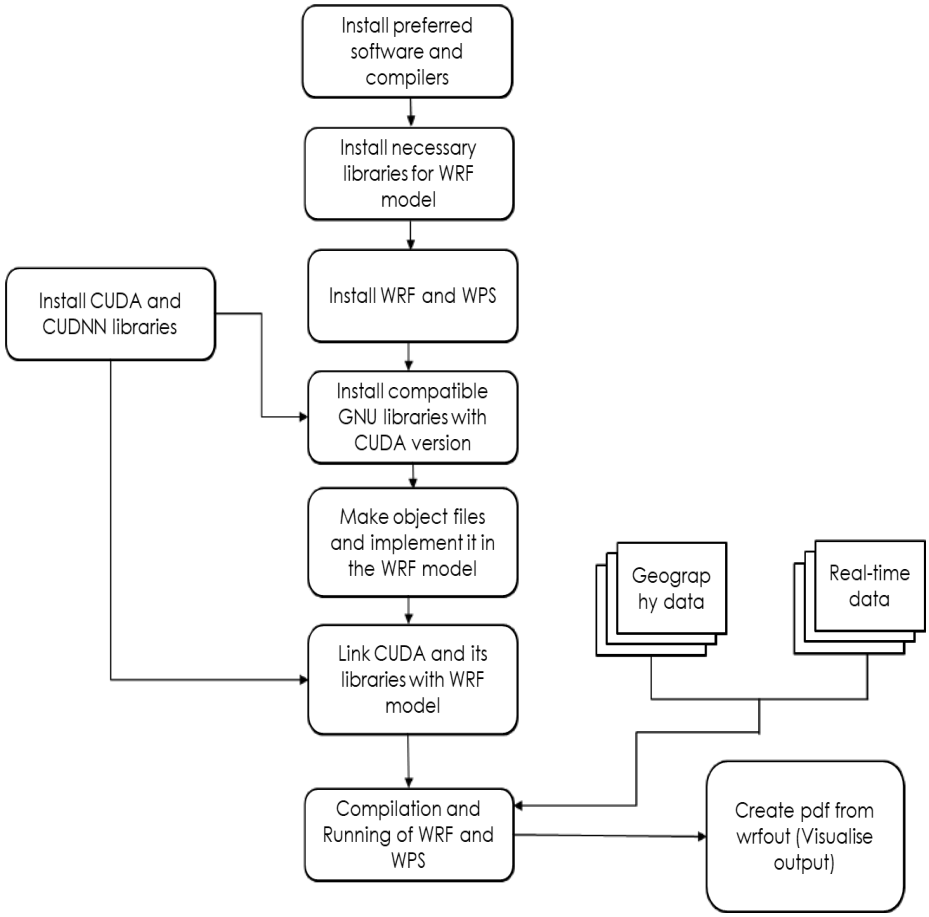

Fig. 1: Work-flow structure for implementing WRF model on Graphical Processing Unit (GPU) using NVIDIA's CUDA.

Firstly, to install WRF in a unix based system, various software and compilers are required to be installed.

The second step is to install required libraries for the WRF model to work. To install the WRF model, firstly the path variable is created by defining MALLOC_CHECK as 0, WRF_EM_CORE as 1 and WRFIO_NCD_LARGE_FILE_SUPPORT as 1. After the path variable is created, source code is downloaded and uncompressed inside the main directory. Similarly, WPS (WRF Preprocessing System) is also installed, that is, source code for WPS is downloaded and uncompressed inside the main directory. The next step here is to install Nvidia's cuda and cudnn libraries in the machine. As we require linux based system to install and run WRF model, the operating system we used in our machine is Ubuntu with a version 16.04, due to some complication (crashing problems) with the graphics driver in the system, version 18.04 was not used to run WRF model. The object files are generated by executing a make-file to run the codes. Make file is created to execute the above files. To execute the make file, firstly, the correct number of levels in the WRF configuration are set. Compilers are selected and values are set for the correct number of threads per block for the used GPU, i.e.16 for X and 16 for Y, this will control the numbers of threads per block. Correct path is set for CUDA LIBRARIES that has been installed in the system earlier. After this, the make file is executed which as a result will create an object file for the CUDA code. The resulting object file is copied to the physics directory of the WRF model. In the top-level WRF directory, configure script is executed to generate a configure.wrf file for the system and configuration. "Dmpar" is selected as we are using gcc for the code execution and nesting is set to basic for this experiment. Finally, em_real model is compiled which creates 2 files named real.exe and wrf.exe.

WPS was configured with Linux x86_64, gfortran (dmpar) option. Configure.wps file was edited by changing Change DM_FC to mpif90 and Append -lgomp in WRF_LIB. After configuration, it was compiled which created 3 files:

i. Geogrid.exe
ii. Metgrid.exe
iii. Ungrib.exe

The Geography data was downloaded from http://www2.mmm.ucar.edu/ and uncompressed in the Geography data directory.

Real-time data is information that is delivered immediately after collection. There is no delay in the timeliness of the information provided. Real-time data is often used for navigation or tracking. Here, Real-time data was collected from North-Eastern Space Application Centre. The data collected was from 2019-04-01 to 2019-04-03, i.e., regularly collected for the interval of 3 hours for the period of 3 days. The data is a Global longitude-latitude grid data. The real-time datasets are stored at the Real-time data directory. To run WPS, the first namelist.wps was edited. Section max_dom, start_date, end_date same Real-time Data and set geog_data_path to Geography Data was edited according to the data collected.

After that, Geography Data was created with a geogrid.exe file created while compiling WPS. After the Geography Data was created, Real-time data was linked with WPS. Variable table (Vtable) was also linked with WPS. Grib files were created with an ungrib.exe file created while compiling WPS. After the grib files are created, met files were created using metgrid.exe file created while compiling WPS to run the WRF model, appropriate microphysics is selected in the namelist.input file for which the object file was created, and then it was ran in the GPU enabled node. Since WRF was compiled with the dmpar option, the code will assume that there is one GPU device available per MPI task. The real case was created. After running WRF, wrfout files are created. In the post-processing step, NCL (NCAR Command Language) script is used to display the output in a PDF form. The PDF is created from wrfout created after running WRF. The downloaded ncl script is edited to add a location of the wrfout file. Finally, command is given to create a PDF from wrfout files.

Result and Discussion:

The Weather Research and Forecasting (WRF) model has been implemented and executed on the Graphical Processing Unit (GPU) in a normal system using Nvidia's CUDA.

Only one of the micro physics options (WSM5) of the WRF model was implemented on the GPU here. By implementing just one microphysics option on GPU, the WRF model could run normally and smoothly in a normal computer with small desktop GPU-clusters which would otherwise have required a high-end system to run the model smoothly. This experiment was done to prove that WRF model can run on normal computer using GPU and does not require high end systems to run the model smoothly and if all of the microphysics options can be implemented on GPU, it will enhance the performance of the WRF model and speed up the execution time of the model.

Table 1:

Successful implementation of WRF (Weather Research and Forecasting) on GPU (Graphical Processing Unit)

NVIDIA-SMI 384.130				
Name	Power: Usage/ Cap	Bus-ID	Memory-Usage	Volatile GPU-Unit
Quadro P2000	18W/75W	00000000:9E.00.00	2467MiB/ 5045MiB	1%

Conclusion:

WRF with one of the microphysics options, WSM5 CUDA variant was able to be executed on GPU. Here, it is concluded that WRF model with the help of CUDA can be executed on GPU and shows that WRF can be executed smoothly even in a normal system with GPU installed and does not require high-end systems to run WRF model smoothly. If most of the microphysics options are implemented using CUDA and then run on GPU then WRF model will not only run smoothly on normal systems but may even be enhanced and accelerate the performance and run-time.

References:

Boston Limited. "What Is GPU Computing?" *Boston Page,* 2020, https:/ www.boston.co.uk/info/nvidia-kepler/what-is-gpu-computing.aspx.

Boukabara, Sid A., et al. "S4: An O2R/R2O Infrastructure for Optimizing Satellite Data Utilization in NOAA Numerical Modeling Systems: A step toward bridging the gap between research and operations." *Bulletin of the American Meteorological Society* 97.12 (2016): 2359-2378.

Dudhia, J. "WRF modeling system overview." *WRF-ARW tutorials.* Mesoscale and Microscale Meteorology Division. NCAR. Recuperado de http://www2.mmm.ucar.edu/wrf/users/supports/tutorial. html (2014).

Ghorpade, J., et al. "GPGPU Processing in CUDA Architecture." *arXiv preprint arXiv:*1202.4347 (2012).

Heller, M. "What is CUDA? Parallel Programming for GPUs." *InfoWorld,* https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html.

NCAR, U. "Weather Research and Forecasting Model|" *MMM: Mesoscale & Microscale Meteorology Laboratory,* (2020).

OmniSci, Inc. "CPU Vs GPU." *OmniSci | Accelerated Analytics Platform,* 7 May 2020, https://www.omnisci.com/technical-glossary/cpu-vs-gpu.

Pattanayak, S., and Mohanty, UC. "Development of Extended WRF Variational Data Assimilation System (WRFDA) for WRF Non-hydrostatic Mesoscale Model." *Journal of Earth System Science* 127.4 (2018): 48.

Powers, Jordan G., et al. "The Weather Research and Forecasting Model: Overview, System Efforts, and Future Directions." *Bulletin of the American Meteorological Society* 98.8 (2017): 1717-1737.

Yu, S. et al. "Comparative Evaluation of the Impact of WRF–NMM and WRF–ARW Meteorology on CMAQ Simulations for O3 and Related Species during the 2006 TexAQS/GoMACCS Campaign." *Atmospheric Pollution Research* 3.2 (2012): 149-162.