

Associate Scientist mission report

Document NWPSAF-MO-VS-043

Version 1.0

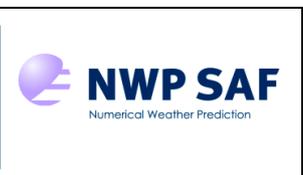
4 July 2011

Development of the GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models

Bormin Huang, Jarno Mielikainen and Hung-Lung Allen Huang
Cooperative Institute for Meteorological Satellite Studies
Space Science and Engineering Center, University of Wisconsin-Madison
1225 W. Dayton Street, Madison, WI 53706, USA

Roger Saunders

Met Office, Fitz Roy Road, Exeter, Devon, EX1 3PB, UK

		Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models	Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011
--	--	---	--

Development of the GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models

Bormin Huang, Jarno Mielikainen, and Hung-Lung Allen Huang
Cooperative Institute for Meteorological Satellite Studies
Space Science and Engineering Center, University of Wisconsin-Madison
1225 W. Dayton Street, Madison, WI 53706, USA

Roger Saunders
Met Office, Fitz Roy Road, Exeter, Devon, EX1 3PB, UK

This documentation was developed within the context of the EUMETSAT Satellite Application Facility on Numerical Weather Prediction (NWP SAF), under the Cooperation Agreement dated 1 December, 2006, between EUMETSAT and the Met Office, UK, by one or more partners within the NWP SAF. The partners in the NWP SAF are the Met Office, ECMWF, KNMI and Météo France.

Copyright 2011, EUMETSAT, All Rights Reserved.

Change record			
Version	Date	Author / changed by	Remarks
0.1	17.06.2011	J. Mielikainen; B. Huang	Initial draft
1.0	4.07.2011	J. Mielikainen; B. Huang	Version approved for release

Scope

This document describes the scientific approach to a study that was performed by Principal Investigators Drs. Bormin Huang and Allen H.-L. Huang at the Cooperative Institute for Meteorological Satellite Studies, the University of Wisconsin-Madison under contract to the EUMETSAT NWP-SAF between February 11 and December 31, 2010. The objective of this study was to develop the Graphics Processing Unit (GPU) accelerated code for the RTTOV-7 IASI and AMSU-A forward models.

1. Introduction

In recent years the graphics processing unit (GPU) has evolved into a highly parallel, multithreaded, manycore processor with tremendous computational horsepower and very high memory bandwidth, as illustrated by the following figure 1.1. Currently, a low-cost personal computer with the 4 NVIDIA Tesla GPU cards (total 960 GPU cores) delivers 4 TFlops of compute power. A GPU cluster with 10 such GPU computers is comparable to the Earth Simulator, the world fastest supercomputer in 2004, which was a stadium-sized cluster with 5120 CPU cores to deliver 40 TFlops.

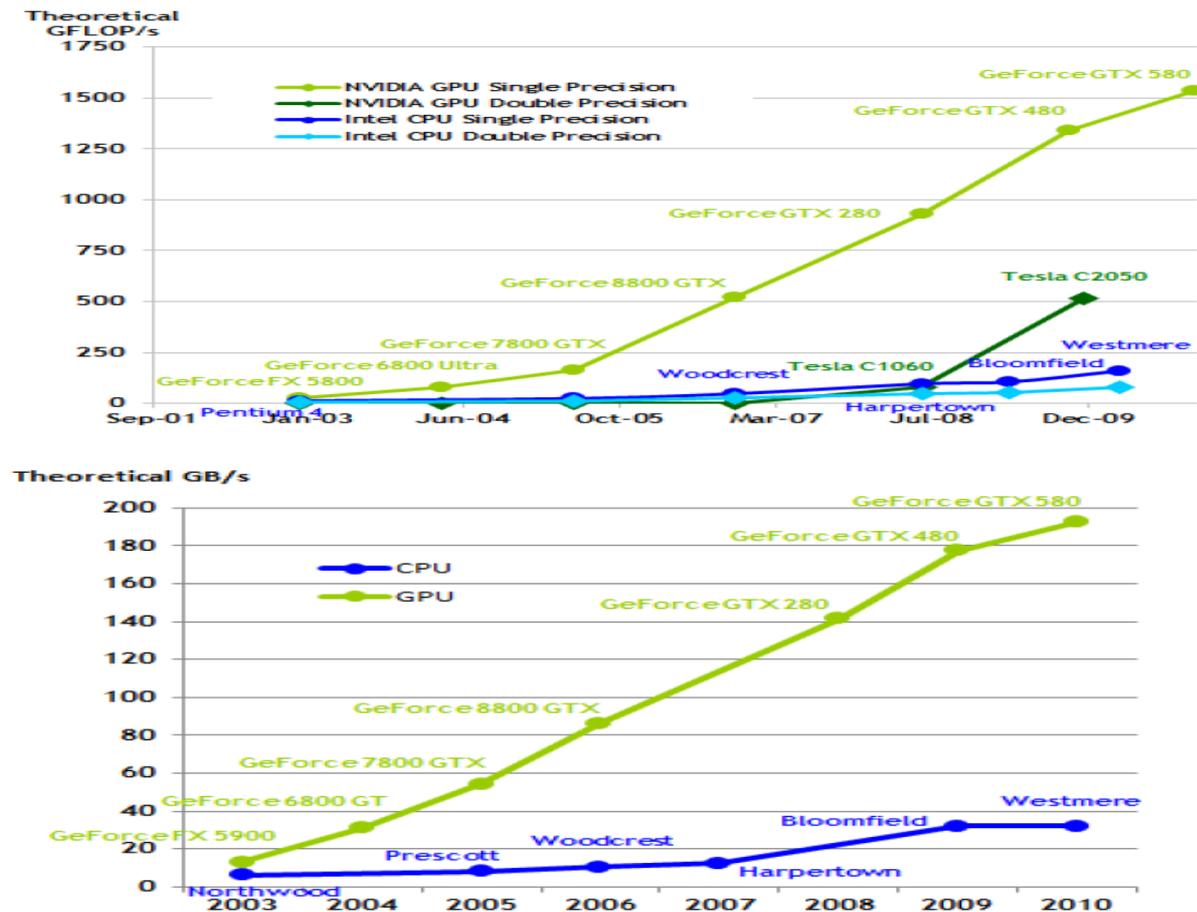


Figure 1-1. Floating-Point Operations per Second (upper) and Memory Bandwidth (lower) for the CPU and GPU. (source: NVIDIA CUDA C Programming Guide v.4.0)

<p>The EUMETSAT Network of Satellite Application Facilities</p>		<p>Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models</p>	<p>Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011</p>
---	---	---	---

The current version of RTTOV is written in FORTRAN-90 and runs on Linux machines from PC desktops to massively parallel supercomputers (e.g. IBM Power 6). RTTOV performance in operational NWP systems still limits the number of channels we can use in hyperspectral sounders to a few hundred, although new PC based simulations of the spectra are becoming possible. The computer architecture for the next generation of supercomputers is not clear. One possibility is to benefit from the power of GPUs and the USA WRF model is being recoded to run on GPU systems. The fast radiative transfer model (RTM) is very suitable for the GPU implementation as it can take advantage of the hardware's efficiency and parallelism, where radiances of many channels can be calculated in parallel in the GPU. Previously, GPUs have also been used very successfully to accelerate Cooperative Institute for Meteorological Satellite Studies (CIMSS) RTM (Huang et al., 2010) and (Mielikainen et al. 2010). This AS mission was a first step to recode a stripped down version of the RTTOV forward model to run on a GPU based platform and compare its performance for both AMSU-A and IASI.

2. RTTOV-7 forward model

Originally the RTTOV model was developed at European Centre for Medium-Range Weather Forecasts (ECMWF) (Eyre and Woolf, 1988), to retrieve temperature and humidity profiles from the Television InfraRed Observation Satellite (TIROS-N) Operational Vertical Sounder (TOVS) (Smith *et al.* 1979). The RTTOV forward model performs the fast computation of the radiances, brightness temperatures, overcast radiances, surface to space transmittances, surface emissivities and pressure level to space transmittances. The RTTOV fast transmittance computation uses regression coefficients derived from accurate line-by-line (LBL) computations. This allows expressing the optical depths as a linear combination of profile dependent predictors that are functions of temperature, absorber amount, pressure and viewing angle (Matricardi and Saunders, 1999). LBL models are too computationally expensive to be used in an Numerical Weather Prediction (NWP) operational environment. The main features of the LBL models that were used to derive the RTTOV coefficients for Infrared Atmospheric Sounding Interferometer (IASI) are discussed in (Matricardi, 2009). The scientific aspects of RTTOV-7, which are different from RTTOV-6 are described in (Saunders *et al.*, 2002).

Both clear sky radiances and cloudy radiances can be simulated by the model. An approximate form of the atmospheric radiative transfer equation is used. Neglecting scattering effects, the top of the atmosphere upwelling radiance, $L(\nu, \theta)$, at a frequency ν and viewing angle θ from zenith at the surface is computed as follows

$$L(\nu, \theta) = (1 - N)L^C(\nu, \theta) + NL^F(\nu, \theta)$$

where $L^C(\nu, \theta)$ and $L^F(\nu, \theta)$ are the clear sky and fully cloud cover top of the atmosphere upwelling radiances and N is the fractional cover. Clear sky top of the atmosphere upwelling radiance is computed as follows

<p>The EUMETSAT Network of Satellite Application Facilities</p>		<p>Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models</p>	<p>Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011</p>
---	---	---	---

$$L^C(\nu, \theta) = \tau_s(\nu, \theta)\epsilon_s(\nu, \theta)B(\nu, T_s) + \int_{\tau_s}^1 B(\nu, T)d\tau + (1 - \epsilon_s(\nu, \theta))\tau_s^2(\nu, \theta) \int_{\tau_s}^1 \frac{B(\nu, T)}{\tau^2} d\tau$$

where τ_s is the surface to space transmittance, ϵ_s is the surface emissivity and $B(\nu, T)$ is the Planck function for a frequency ν and temperature T . A microwave surface emissivity model FASTEM-1 (English and Hewison, 1998) is used to compute ocean surface emissivity given a sea surface temperature, surface wind speed and viewing angle for a microwave radiometer channel. The model computes a surface emissivity for the channel of interest at the given viewing angle.

The transmittances are computed by means of a linear regression in optical depth based on variables from the input profile vector. The simulation of transmittances in RTTOV-7 is based on a regression scheme with 10 predictors for the mixed gases, 15 for water vapor and 11 for ozone. The regression is performed to predict layer optical depth directly

$$d_{i,j} = d_{i,j-1} + \sum_{k=1}^K a_{i,j,k} X_{k,j}$$

where K is the number of predictors, $d_{i,j}$ is the level to space optical depth from level j and channel i , $a_{i,j,k}$ are the regression coefficients. The functions $X_{k,j}$ constitute the profile-dependent predictors of the fast transmittance model. Conversion of optical depths to transmittances is a computationally inexpensive procedure and it is described in detail in [13]. In RTTOV-7 fast forward model there are 43 pressure levels in total. Assuming emissivity of the cloud top to be unity and black, opaque clouds at a single level the simulation of cloud affected radiances is defined as follows

$$L^F(\nu, \theta) = \tau_c(\nu, \theta)B(\nu, T_c) + \int_{\tau_c}^1 B(\nu, T)d\tau$$

where τ_c is the top of the cloud top to space radiance and T_c is the cloud top temperature.

3. GPU-based computing

The GPU-based RTTOV-7 IASI and AMSU-A forward models were performed on a low-cost 960-core NVIDIA Tesla personal supercomputer with 1 AMD Phenom quad-core CPU and 4 NVIDIA Tesla C1060 GPUs. The Tesla C1060 GPU has massively parallel computing power and high memory bandwidth. Table 1 shows its specifications (Lindholm et al., 2008).

NVIDIA CUDA is a general purpose parallel computing architecture with a new parallel programming model and instruction set architecture to unlock the computing power of NVIDIA GPUs. CUDA is an extension to the C programming language offering programming GPU's directly. 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 to be executed simultaneously. The GPU-based RTTOV-7 IASI and AMSU-A forward models can also run on other NVidia CUDA-enabled GPUs listed in Appendix B.

The CUDA code consists of three computational phases: transmission of data into the global memory of the GPU, execution of the GPU kernel, and transmission of results from the GPU into the memory of CPU. The problem is divided in a grid of blocks. Each block consists of a number of threads, which are executed in a multiprocessor. A schematic visualization of multiprocessor architecture is presented in Fig 1.

Table 1. Specifications of the NVIDIA Tesla C1060 GPU card.

Number of Streaming Processor Cores	240
Frequency of Processor Cores	1.3 GHz
Single Precision floating point performance (peak)	933 GLOPS
Total Dedicated Memory	4 GB GDDR3
Memory Speed	800 MHz
Memory Interface	512-bit
Memory Bandwidth	102 GB/s
Max Power Consumption	187.8 W

NVIDIA Tesla C1060 consists of 30 multiprocessors. Figure 1 presents a schematic visualization of a GPU device. Each multiprocessor has eight cores and executes in parallel with the other multiprocessors. All eight cores in a multiprocessor execute in data parallel Single Instruction Multiple Data (SIMD) fashion; all cores in the same multiprocessor execute the same instruction at the same time. Each GPU has 4 GB of global memory, which have a higher bandwidth than the DRAM memory in the CPUs. However, access to 16 KB of software-managed data cache, called shared memory, inside a multiprocessor can be performed in one clock cycle compared to 400-600 cycles required by global memory access. Therefore, it is advisable to keep frequently used data in shared memory instead of global memory. Each core also has 16384 32-bit registers which can be accessed in one clock cycle. A description of the CUDA programming model can be found e.g. in (Nickolls and Dally, 2010) and (Sanders and Kandrot, 2010).

Multi processor 1

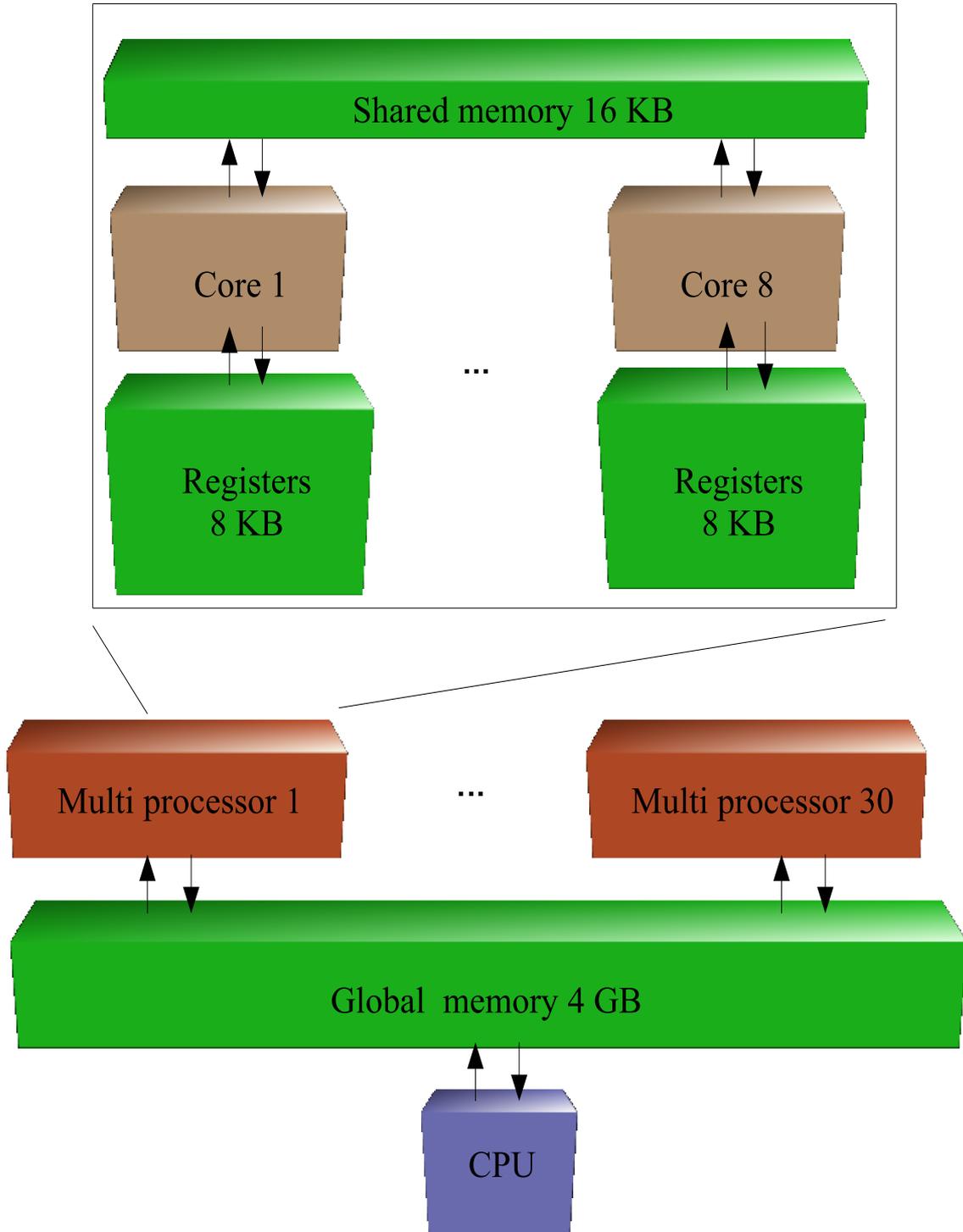


Figure 1. Schematic visualization of a GPU device.

<p>The EUMETSAT Network of Satellite Application Facilities</p>		<p>Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models</p>	<p>Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011</p>
---	---	---	---

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 and its maximum size is 65535 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. Each thread block can consist of up to 512 threads, which provide fine-grained data parallelism. The number of threads per thread block is also limited by the shared memory and register usage. In order to successfully launch a kernel there must be enough registers and shared memory available per multiprocessor to process at least one thread block.

The current generation of NVIDIA's GPUs group threads in each thread block into groups of 32 threads called warps. A multiprocessor 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. Since each processor can manage 24 warps, and there are up to 32 threads in each warp, each multiprocessor can support 768 active threads. At each clock cycle, the multiprocessor 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 multiprocessor with a lot of active threads allows the GPU to hide the latency of slow instructions.

An efficient use of global memory is an essential requirement for a high performance CUDA kernel. Global memory loads and stores by threads of a half-warp (16 threads) are coalesced by the device in as few as one memory transaction when the following access requirements are met. First, the data type must be 32-, 64- or 128-bit. Second, the starting address of the memory access must be aligned. Finally, the threads must access the data sequentially.

4. GPU implementation of RTTOV-7 forward model

The GPU-based RTTOV forward model experiments were performed on a low-cost 960-core NVIDIA Tesla personal supercomputer with 1 AMD Phenom quad-core CPU and 4 NVIDIA Tesla C1060 GPUs. The Tesla C1060 GPU has massively parallel computing power and high memory bandwidth. In all the kernels, the overall work is divided such that each thread is responsible for computing the results for a single channel. The difference between single-profile and multi-profile kernels is that in a multi-profile kernel each thread is responsible for computing the results for a single channel in *several profiles*. Additional difference between multi profile Advanced Microwave Sounding Unit (AMSU-A) and IASI kernels is that computational work for 15-band AMSU-A data is divided by half-warp. Thus, the first 16 threads in a thread block compute results for a complete profile and the next 16 threads compute results for second AMSU-A profile.

```

Calculate a channel number, ich, that is processed by a thread
For level = 1 to all levels
%Transfer predictors for mixed gasses (xxm), water vapour (xxw) and oxygen (xxo)
% for the current level from global memory to shared memory
For pred = 1 to 10 % mixed gas predictor
    C = cfm[level, pred, ich] % Mixed gas coefficient from global memory to a register
    For profile = 1 to nS profiles
        zopdp[level, profile] += C * xxm[pred, profile]
    For pred = 1 to 15 water vapour predictor
        C = cfw[level, pred, ich] % water vapour coefficient
        For profile = 1 to nS profiles
            zopdp[profile] += C * xxw[pred, profile]
    For pred = 1 to 11 ozone predictor
        C = cfo[level, pred, ich] % ozone coefficient
        For profile = 1 to nS profiles
            zopdp[profile] += C * xxo[pred, profile]
    For profile = 1 to nS profiles
        if zopdp[profile] < 0.0
            zopdp[profile] = 0.0 % ensure a non-negative layer transmittance value
            %Compute layer to space optical depths
            opdpa[profile] += zopdp[profile];
            opdp_cur = opdpa[profile];
            opdp_cur = -opdp_cur * gamma[ich]; % gamma factor transmittance corrections
            opdp_cur = max(opdp_cur, opdp_prev[profile]);
            opdp[profile, level, ich] = opdp_cur;
            opdp_prev[profile] = opdp_cur;
            tau[profile, level, ich] = expf(-opdp_cur); % optical depths to transmittances.

For profile = 1 to nS profiles
    isf = nlevsf[profile] % Index of nearest std press level at/below surface
    % Surface is above level nlevsf by fracps fraction of standard pressure level interval
    za = opdp[profile, isf+1, ich] + fracps[profile] * (opdp[profile, isf, ich] - opdp[profile, isf+1, ich])
    tausfc[profile, ich] = expf(-za) % Transmittances from surface to space

```

Figure 2. Pseudo code for CUDA kernel that calculates optical depths from every pressure level to space, transmittances from each level to space and transmittances from surface to space for *nS* profiles.

The most time consuming CUDA kernel for computing optical depths from every pressure level to space, transmittances from each level to space and transmittances from surface to space for *several* profiles is shown in Figure 2. The optimal number of profiles for simultaneous processing was 9 and 300 for AMSU-A and IASI, respectively. The kernel involves a dot product between regression coefficients for predicting the effective layer optical depths and predictors for the effective layer optical depths. The dot product is performed for three gases and 43 pressure levels. Each step of a dot product for all channels in spectra involves the same predictor for all spectra. This allows first storing a

regression coefficient in a register and then computing multiplications between a regression coefficients and a predictor for several profiles. Next, a new regression coefficient is loaded into register file and a next step of a dot product is performed. Therefore, multi-profile processing allows even greater processing speed than single-profile processing.

Table 1 shows processing times and speedups for AMSU-A. Processing times on a CPU are 0.287 ms for AMSU-A profile and 186.8 ms for IASI.profile. Column 1 gives a description of the computation. Second column depicts processing times for the original Fortran code using gfortran compiler with -O2 compiler switch for optimization. Third and fourth columns show computation times for GPU accelerated single-profile code and its speed up compared to the Fortran code. The last two columns show the results for multi-profiles processing. Processing multiple profiles at a time gives much better performance than a single profile processing. As explained earlier, the reason for this is that multiple profile processing allows reusing the data in the fast register and shared memory instead of loading the same data repeatedly from slow global memory into GPU cores.

Subroutine description	Fortran [ms]	Single profile CUDA C [ms]	Speedup	Multi profile CUDA C [ms]	Speedup
Calculates optical depths from every pressure level to space, transmittances from each level to space and transmittances from surface to space	0.141	0.412	0.34	0.00276	51.09
Integration of radiative transfer equation, convert atmospheric temperatures to Planck functions and radiances to brightness temperatures.	0.137	0.132	1.04	0.00063	209.52

Table 1. Processing times for Fortran subroutines, single-profile CUDA C kernels and multi-profile CUDA C kernels for AMSU-A.

In Table 2, similar processing time results are shown for IASI as were presented in Table 1 for AMSU-A. Although, both single profile and multi profile codes utilize all cores in IASI processing there is still a significant difference between the two versions. As explained earlier, this is due to the fact that the data that needs to be transferred from global memory to cores is reduced in multi profile processing.

Subroutine description	Fortran [ms]	Single profile CUDA C [ms]	Speedup	Multi profile CUDA C [ms]	Speedup
Calculates optical depths from every pressure level to space, transmittances from each level to space and transmittances from surface to space	110.32	0.75500	146	0.291	379
Integration of radiative transfer equation, convert atmospheric temperatures to Planck functions and radiances to brightness temperatures.	75.31	0.151	499	0.096	785

Table 2. Processing times for Fortran subroutines, single-profile CUDA C kernels and multi-profile CUDA C kernels for IASI.

	average time [ms]	speedup
AMSU-A	0.638	0.4
IASI	1.191	156.9

Table 3. Processing times for 1 profile on a GPU for both AMSU-A and IASI.

In Table 3, total computation time results for single profile processing are shown both for AMSU-A and IASI. The results are as the individual kernel results indicated before. A single-profile AMSU-A is slower on GPU and on CPU and IASI is significantly faster on GPU.

Concurrent GPU kernel execution and data transfer from CPU to GPU can be managed through streams. A stream is a sequence of commands that execute in order. By overlapping device computation with asynchronous data transfers and host computation it is possible to reduce the total execution time. Streams are logically independent queues of operations to be executed on GPUs. Hardware maps the commands on streams to an engine to execute kernels and an engine to perform memory transfers. A diagram depicting the execution timeline of the radiative transfer model is shown in Figure 2. The five different operations are colored in different colors. Vertical direction in Figure 2 represents time.

Predictors are first computed on a CPU and then they are transferred from host to device using a copy engine on a GPU. After the memory transfer CUDA kernels are launched asynchronously. Finally, results of CUDA kernels are transferred back to device. When the process is repeated several times only the initial predictors, host to device memory transfer and final device to host memory transfer are the latency that contributes to the total time.

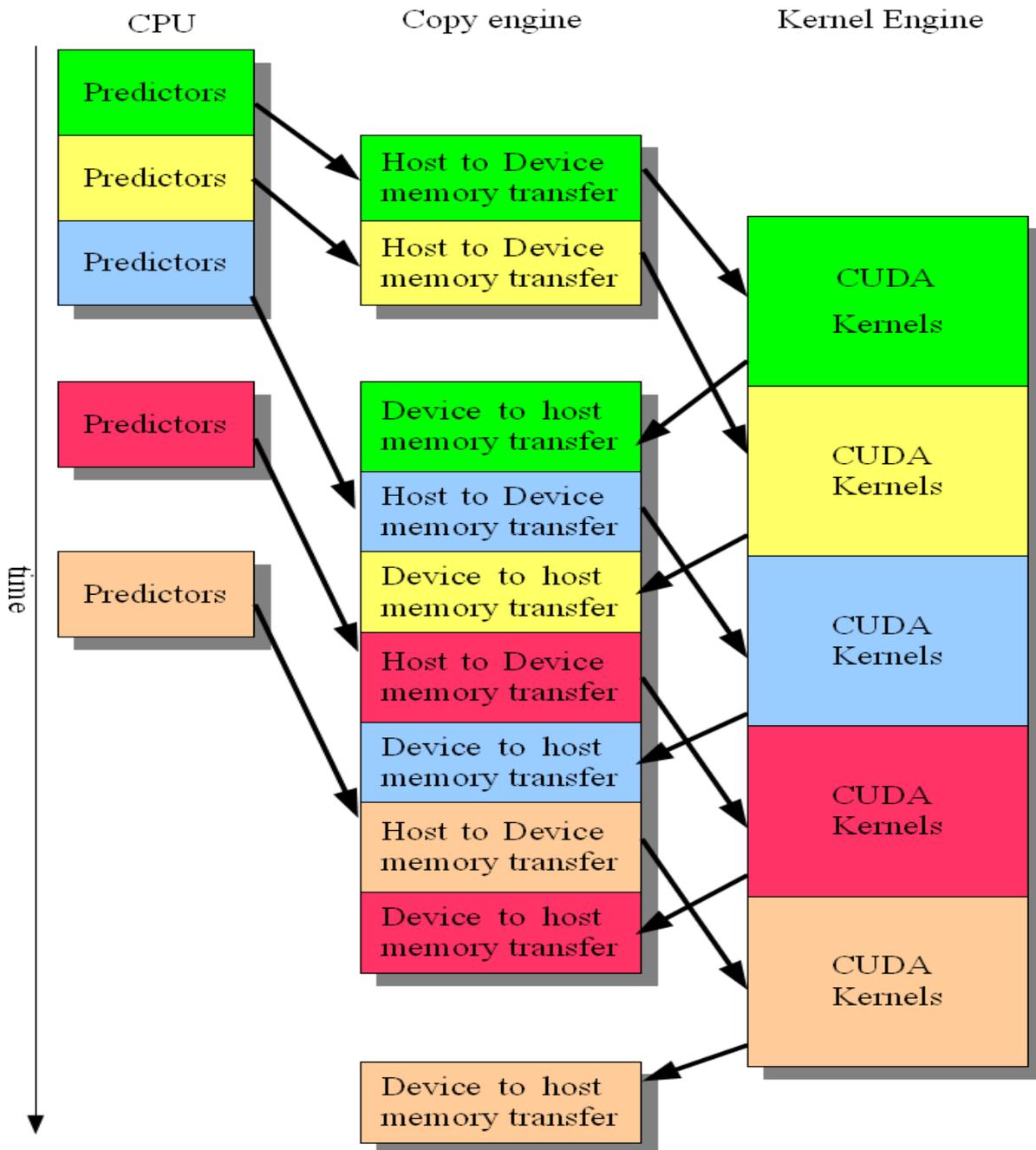


Figure 2. Execution timeline for five CUDA kernel executions. The five different operations are colored in different colors.

Figures 3 and 4 show average processing times for a single profile. As shown in Figure 2 the initial predictions and the final memory transfer from device to host are not overlapping the other computation. Thus, when the number of calls to CUDA kernels is high enough their contribution to the overall processing time becomes insignificant.

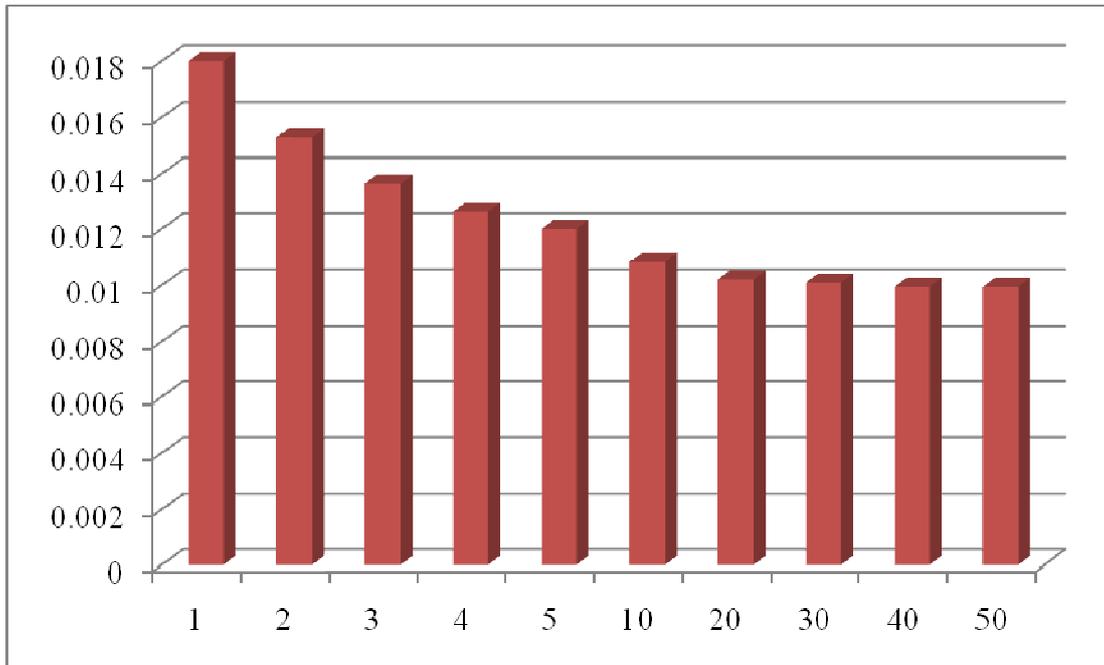


Figure 3. Average processing times [ms] for 1 profile of AMSU-A on a GPU as a function of the number of call to multi-profile CUDA kernels.

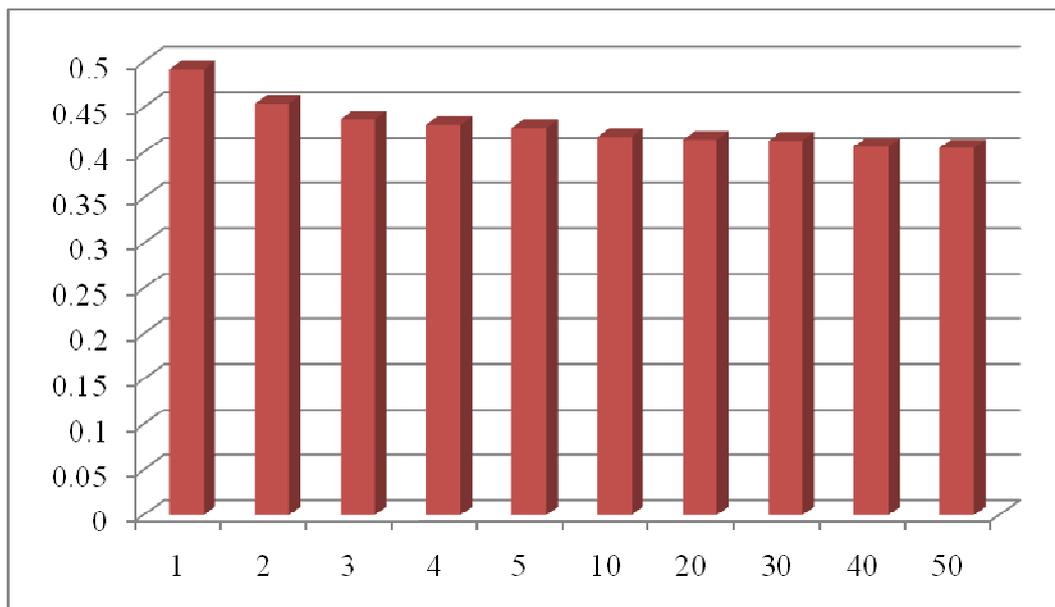


Figure 4. Processing times [ms] for 1 profile of IASI on a GPU as a function of the number of call to multi-profile CUDA kernels.

In Figure 5, schematic visualization of running RTTOV on 4 GPUs is illustrated. There are five threads controlling the four GPUs. The first thread is a master thread, which controls the slave threads. Each GPU must be controlled by a different CPU thread. Allocation of portable pinned memory is performed in the thread number 1. For pinned memory GPU can use direct memory access to copy data from or to host. Using portable pinned memory all thread see pinned memory. Between barriers 4 and 2 thread number 1 performs the pinned memory allocation for global memory. Barrier synchronizes participating threads. Computation begins after barrier 1. When all the threads have reached barrier 3 the computation is finished. Resources are released after barrier 5 is reached.

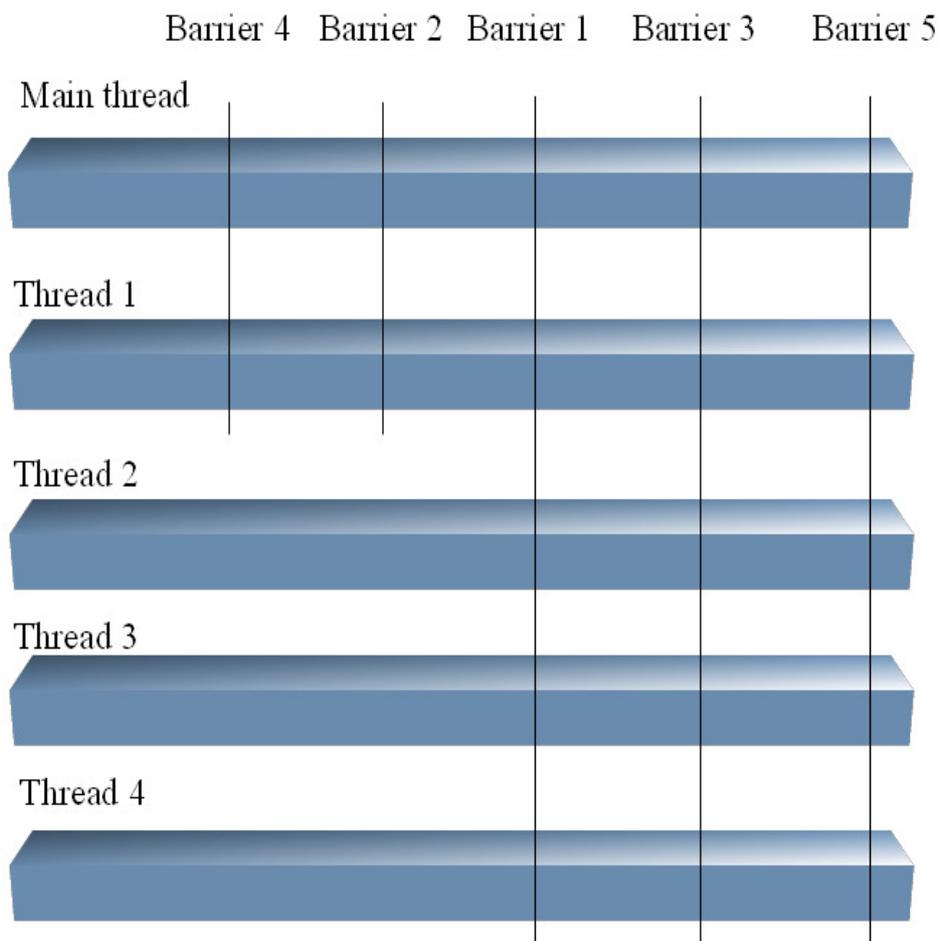


Figure 5. Schematic visualization of RTTOV running on multiple GPUs.

Figure 6 shows the scaling of AMSU-A for 1 to 4 GPUs in the form of the speedup compared to the original Fortran code. Speedup is the average GPU processing time to process 50 calls to CUDA kernels compared to the time to run the original Fortran code. Four GPUs take 230.26 ms to process the profiles, which is 53% longer than 149.62 ms to process the same amount of profiles per GPU on a single GPU.

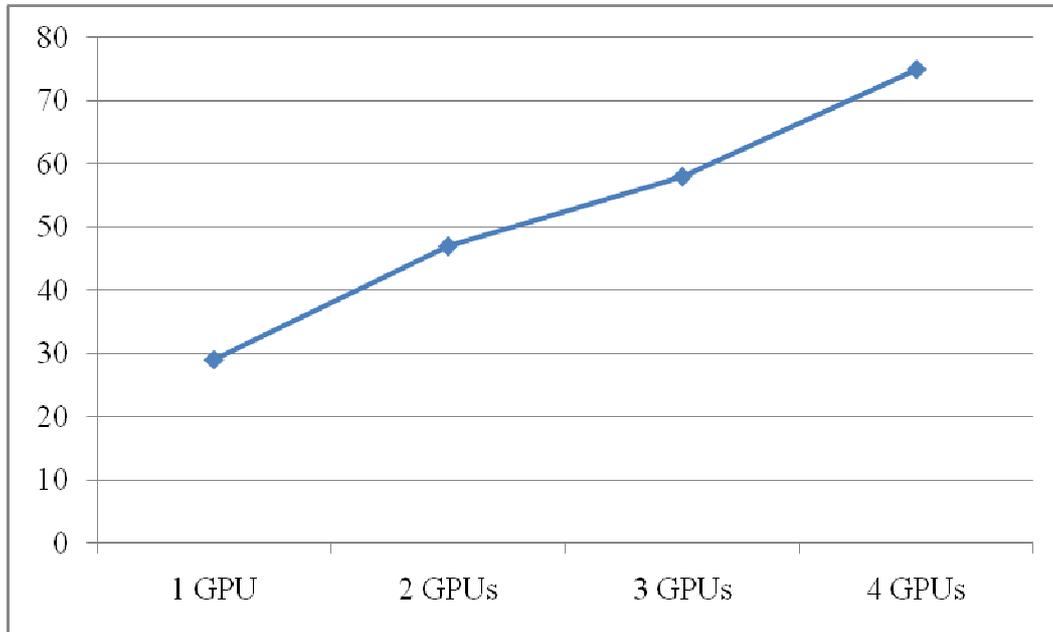


Figure 6. Speedups for AMSU-A for 1 to 4 GPUs.

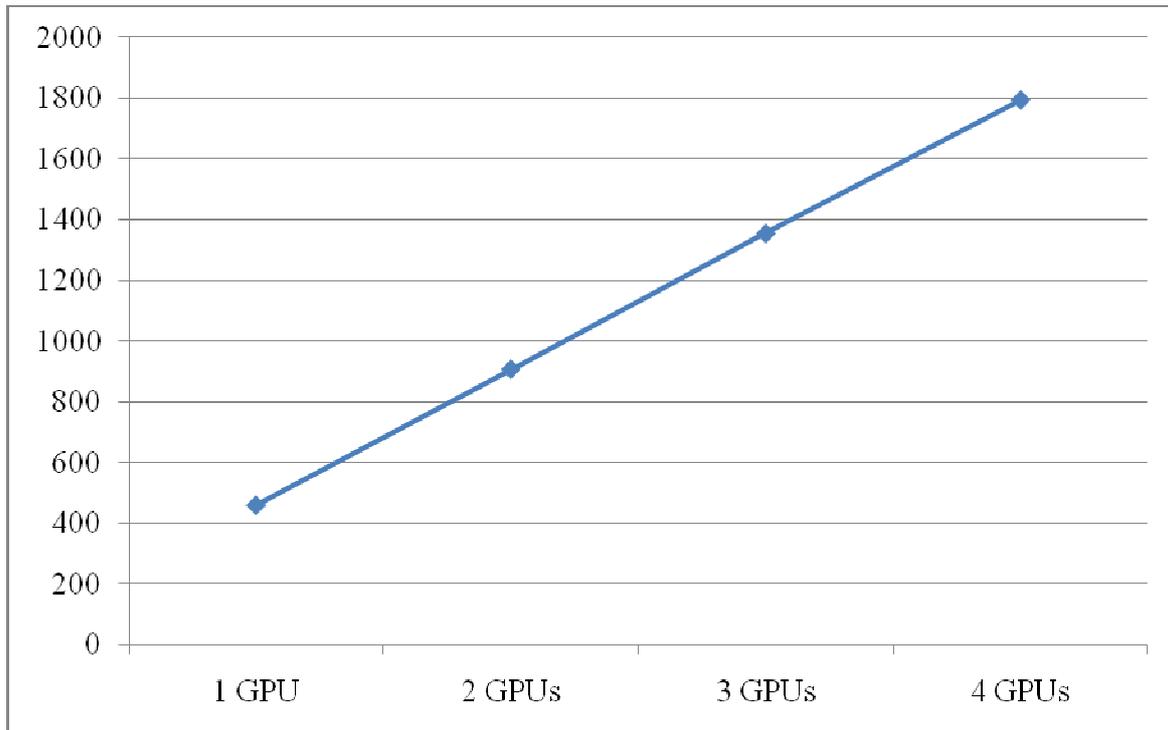


Figure 7. Speedups for IASI for 1 to 4 GPUs.

Figure 7 shows the scaling of IASI for 1 to 4 GPUs. Using four GPUs per profile processing time drops from 0.405 ms to 0.104 ms for a single GPU. Thus, four GPUs have 3.89x higher speedup than 1 GPU. Therefore, IASI is much closer to the ideal linear speedup than AMSU-A. The reasons for that will be analyzed next.

	AMSU-A	IASI
CPU to GPU	9112	76676
GPU to CPU	12032	136192

Table 4. The amount of data in bytes that is transferred between CPU and GPU per profile.

By dividing total amount of memory transfers in Table 4 by the per profile processing times we get the number of transferred bytes in a second. The resulting global memory throughput in MB/s for AMSU-A and IASI are shown in Figure 8. Theoretical PCI express 2.0 bandwidth is 8.0 GB/s (4.0 GB/s per direction). In practice, measuring throughput using NVIDIA's bandwidth test program memory throughputs of 3094.5 MB/s and 3162.5 MB/s for host to device and device to host transfers was measured. Based on the above, AMSU-A is limited by the PCI express bus throughput from GPU to CPU. On the other hand, the measured global memory overall throughput for IASI CUDA kernels is 80 GB/s. Thus, IASI is limited by global memory throughput.

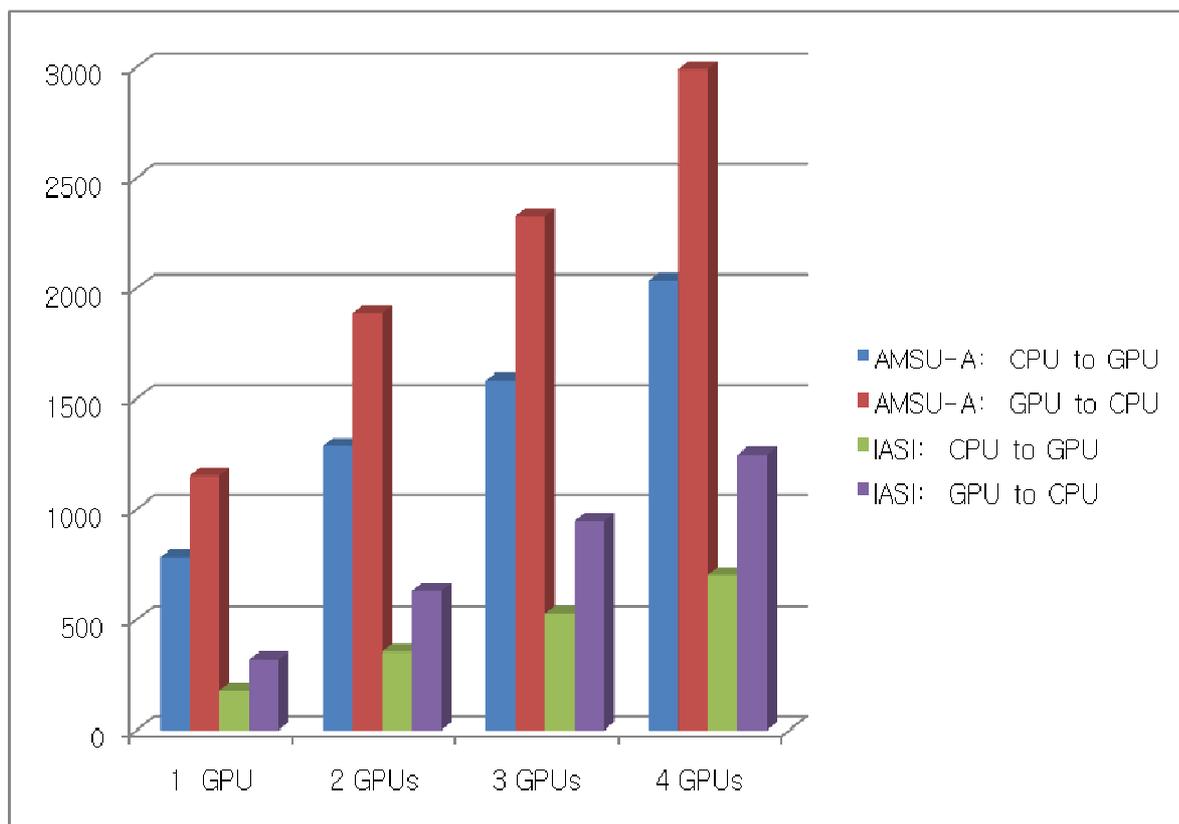


Figure 8. Global memory throughput in MB/s for AMSU-A and IASI for both transfer directions.

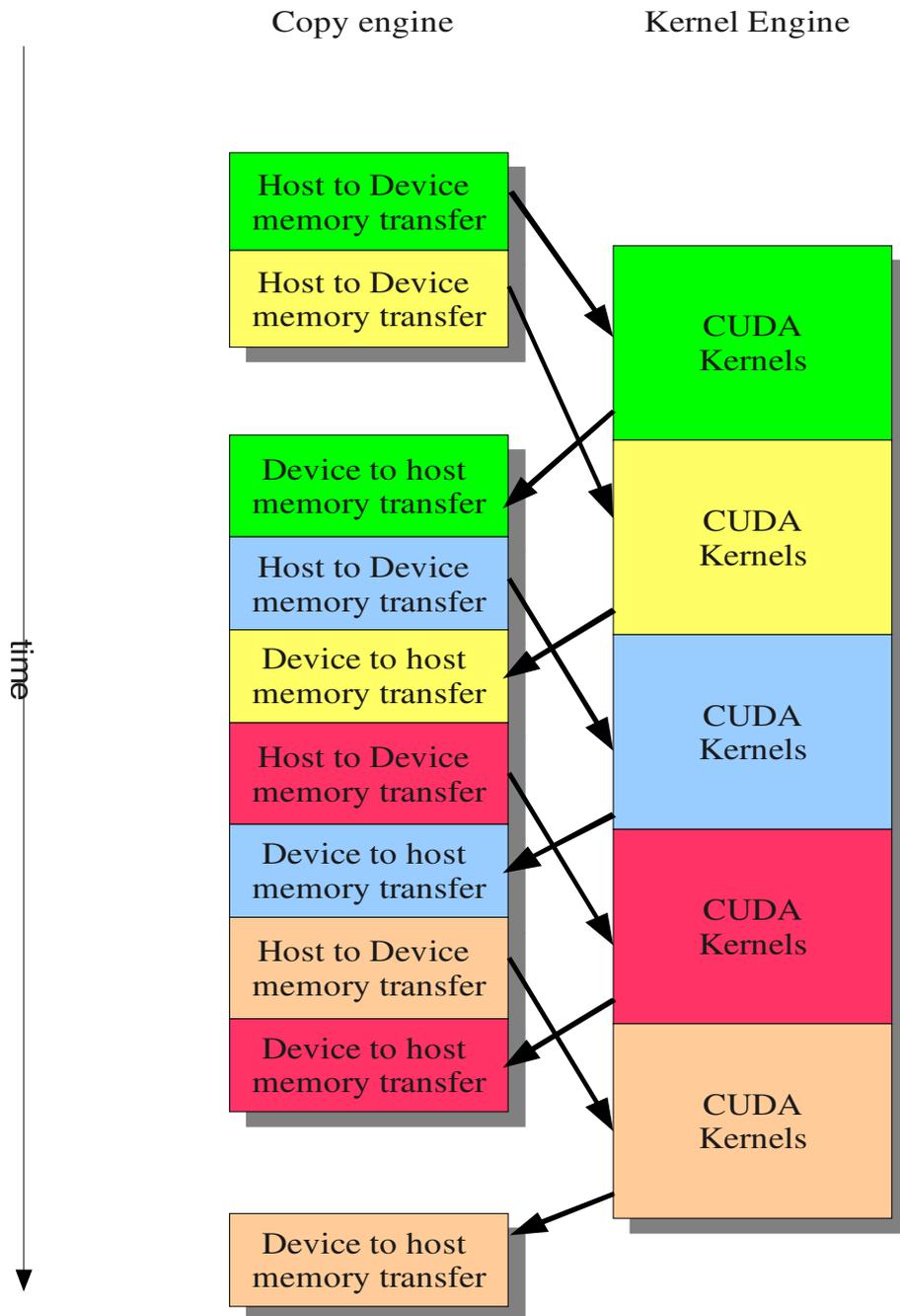


Figure 9. Execution timeline for five CUDA kernel executions. The five different operations are colored in different colors.

A diagram depicting the execution timeline of the Pure GPU AMSU-A is shown in Figure 9. Vertical direction in Figure 9 represents time. In this case CUDA kernels also include predictor kernel, which is computer before the other kernels. This arrangement reduced the amount of transferred data from CPU to GPU. Thus, reducing the execution time.

Number of GPUs	Execution time for 1 profile [ms]	Speedup
1	0.00513	56x
2	0.00276	104x
3	0.00236	122x
4	0.00229	125x

Table 5. Execution time for pure GPU AMSU-A.

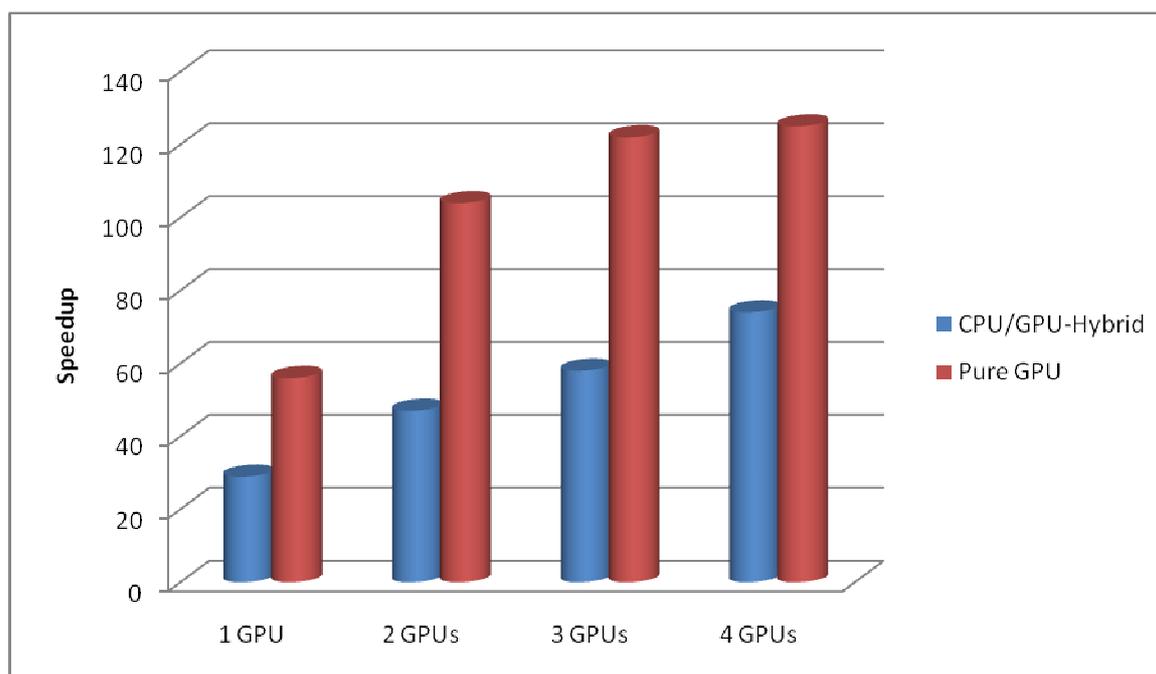


Figure 10. Speedups for CPU/GPU-hybrid and pure-GPU AMSU-A.

In Table 5, execution times and speedups for pure-GPU AMSU-A are depicted. Figure 10 compares speedups for CPU/GPU-hybrid AMSU-A and pure-GPU AMSU-A. It can be seen that version pure-GPU AMSU-A is two times faster than AMSU-A CPU/GPU hybrid for 1 GPU. With 4 GPUs the speedup is reduced to 1.64x as pure-GPU AMSU-A becomes limited by the PCI express bus throughput from GPU to CPU.

5. Conclusions

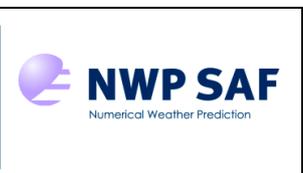
GPU computing is more effective for a hyperspectral IASI sensor than AMSU-A sensor as IASI yields more computations per data transfer between host and device. To compute one day's amount of 1,296,000 IASI spectra, the CPU code will take 2.8 days, whereas the multi-input 1-GPU and 4-GPU codes will take 8.75 and 2.25 minutes, respectively.

6. References

English S. J. and T. J. Hewison, 1998. A fast generic millimetre wave emissivity model. In *Proceedings of Microwave remote sensing of the atmosphere and environment, Beijing*,

<p>The EUMETSAT Network of Satellite Application Facilities</p>		<p>Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models</p>	<p>Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011</p>
---	---	---	---

- China 15-17 Sep 1998. Society of Photo-Optical Instrumentation Engineers: Bellingham, WA. DOI: 10.1117/12.319490.
- Eyre J. R., and H. M. Woolf, 1988. Transmittance of atmospheric gases in the microwave region: a fast model. *Appl. Optics*. 27: 3244 – 3249. DOI:10.1364/AO.27.003244
- Huang B., Mielikainen J., Oh H. and Huang H.-L, "Development of a GPU-based high-performance radiative transfer model for the Infrared Atmospheric Sounding Interferometer (IASI)," *Journal of computational Physics*, DOI:10.1016/j.jcp.2010.09.011
- Lindholm E., Nickolls J., Oberman S., and J. Montrym, 2008. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro*, 28: 39 – 55. DOI:10.1109/MM.2008.31.
- Matricardi M, and R. Saunders, 1999. Fast radiative transfer model for simulation of infrared atmospheric sounding interferometer radiances. *Appl. Optics*. 38: 5679 – 5691. DOI:10.1364/AO.38.005679.
- Matricardi M, 2009. An assessment of the accuracy of the RTTOV fast radiative transfer model using IASI data. *Atmos. Chem. Phys. Discuss.* 9: 9491 – 9535. DOI:10.5194/acp-9-6899-2009.
- Mielikainen J., Huang B. and Huang A., "GPU Accelerated Multi-Profile Radiative Transfer Model for the Infrared Atmospheric Sounding Interferometer," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, submitted for publication.
- Nickolls J, and W. J. Dally, 2010. The GPU Computing Era. *IEEE Micro*. 30: 56-69. DOI:10.1109/MM.2010.41.
- Saunders R. W., English S., Rayer P., Matricardi M., Chevallier F., Brunel P., and G. Deblonde, 2002. 'RTTOV-7: a satellite radiance simulator for the new millennium'. In *Proceedings of ITSC-XII Lorne*, Australia, Feb 26 – Mar 5, 2002.
- Sanders J and E. Kandrot, 2010. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional: Ann Arbor, Michigan.
- Smith W. L., Woolf H. M., Hayden C. M., Wark D. Q., and L. M. McMillin, 1979. The TIROS-N operational vertical sounder. *Bull. Am. Meteorol. Soc.* 60: 1177 – 1187.

		Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models	Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011
--	--	---	--

Annex: RTTOV-7 GPU Users Guide

This documentation was developed within the context of the EUMETSAT Satellite Application Facility on Numerical Weather Prediction (NWP SAF), under the Cooperation Agreement dated 10 February 2010, between the Space Science and Engineering Center (SSEC), University of Wisconsin-Madison, USA and the Met Office, UK, by one or more partners within the NWP SAF. The partner in the NWP SAF is the Met Office.

Before attempting to use the RTTOV-7 GPU model the reader is advised to also read the RTTOV-7 Users Guide for an overview of the RTTOV-7 fast radiative transfer model and information about the Fortran implementation running on a CPU. This document shows how to install the RTTOV-7 fast radiative transfer model GPU code on a Linux platform and run it.

1 Prerequisites

In order to install RTTOV-7 GPU, a CUDA enabled graphics processor is required along with some related software, which are described below.

1.1 CUDA enabled graphics processor

Every NVIDIA GPU since the 2006 release of the GeForce 8800 GTX has been CUDA-enabled. For a complete list a CUDA-enabled graphics processors consult NVIDIA website at <http://www.nvidia.com/cuda>.

1.2 NVIDIA device driver

Visit <http://www.nvidia.com/cuda> and click the "Download Drivers" link. Select the options that match the graphics card and operating system on which you plan to compile the code.

1.3 CUDA development toolkit

You can download the CUDA toolkit at <http://developer.nvidia.com/object/gpucomputing.html>.

1.4 Standard C compiler for the host code

Linux distributions ship with the GNU Compiler Collection (gcc) installed, which is used for compiling the host code. Also, Intel C++ Compiler (icc) can be used as a host compiler for CUDA 3.2 or higher.

2 Test programs

There are six Unix test scripts for running test programs. Each of the test scripts requires both associated input data files and radiative transfer coefficient files. The main directory

for those files is set in `dir.sh` file. The output of test programs run on user's machines are `print.dat` and `print.diff`, which are the output of the test program and the difference between the CPU and GPU versions outputs, respectively. The CUDA C code consists of CUDA C kernels and C functions and top level test programs (`tstrad.c` and `tstrad_multi.cu`) for complete testing of the RTTOV subroutines. The first step is to compile the code and to make the executables using the makefile supplied.

In order to compile the source code both NVIDIA device driver and CUDA development toolkit have to be installed (Sections 1.2 and 1.3). After installation, uncompress and expand the compressed tar file containing the source code:

```
tar xvfz rttov7_gpu.tgz
```

Next, change current directory to newly created `rttov7_gpu` directory:

```
cd rttov7_gpu
```

Type `make` and the code will compile and produce the following executables

<code>tstrad</code>	Processes a single IASI profile during one call to a CUDA kernel
<code>tstrad_multi</code>	Tests multi-profile GPU code that processes 9 IASI profiles during one call to a CUDA kernel. If the number of profiles is not a multiple of 9 then the residual profiles are computed using single-profile GPU code
<code>tstrad_multi_GPU</code>	Multi GPU version of the above
<code>tstrad_15</code>	Processes a single AMSU-A profile during one call to a CUDA kernel
<code>tstrad_multi_15</code>	Processes 300 IASI profiles during one call to a CUDA kernel. If the number of profiles is not a multiple of 300 then the residual profiles are computed using single-profile GPU code
<code>tstrad_multi_GPU_15</code>	Multi GPU version of the above

The corresponding test scripts for running test programs are `amsua.sh`, `amsua_multi.sh`, `amsua_multi_GPU.sh`, `iasi.sh`, `iasi_multi.sh` and `iasi_multi_GPU.sh`.

There also exists an experimental pure GPU version for AMSU-A, which is in compressed tar file named `rttov7_pure_gpu.tgz`. The above explanation for installation applies to it as well. However, there are only two test scripts for pure GPU AMSU-A: `amsua_multi.sh` and `amsua_multi_GPU.sh`.

3 Running RTTOV-7 GPU for your applications

3.1 High level interface to RTTOV-7 GPU

To run RTTOV-7 for a user's application files `tstrad.cu` and `tstrad_multi.cu` can be used as a rough guide or template. Using a C pre-processor directive **MULTI_GPU** files `tstrad_multi.cu` and `rttov_multi.cu` illustrate how to initialize multiple GPUs and to call

CUDA kernels that compute RTTOV-7 forward model for multiple profiles. The number of GPUs is defined in file *multiGPU.h* using variable **GPU_N**. The input to function *rttov* is transferred using a struct **TGPUplan**, which has the following fields:

data type	variable name * for arrays	field description
int	thr_id	Thread id
int	alloc_host_mem	1 for a thread that allocates host memory, 0 otherwise
int	alloc_nprof	Number of profiles the results are computed
int	device	GPU device id
float	pangl	Satellite local zenith angle (deg)
float	pangs	Solar zenith angle at surface (deg)
int	ksurf	Surface type index
int	knchpf	The number of channels
float	*pav	Atmospheric profile variables
float	*psav	Surface air variables
float	*pssv	Surface skin variables
float	*pcv	Cloud variables
float	*pemis	Surface emissivities
float	*prad	Radiances (mw/cm-1/ster/sq.m)
int	lcloud	Switch for cloud computation
float	*emc	Emissivity model data
float	*cfm	Mixed gas coeffs
float	*cfw	Water vapour coeffs
float	*cfo	Ozone coeffs
int	max_nprof	Number of profiles the memory is allocated
float	*freq	Channel frequencies in GHz
int	fmv_gas	Number of active gases
int	sensor	Sensor number
float	*tc1	Band correction coefficient: offset in K
float	*tc2	Band correction coefficient: slope in K
float	*bcon1	1st Planck function constant in mW/m**2/sr/cm**-4
float	*bcon2	2nd Planck function constant in K/cm**-1
float	*emcir	Satellite and ir channel for surf
float	*wvnum	Wavenumber in cm**-1
int	*mpol	Polarization of each channel
float	*gamma_factor	<i>Gamma factor</i> transmittance corrections
float	*xpres	Standard pressure levels for transmittance
float	*dpres	Intervals between standard pressure levels in mb

Two global integer arrays of size **GPU_N** are also used as inputs for *rttov_multi* function.

array name	description
global_nprof	The number of profiles that are computed using a thread
global_first_prof	The id number of the first profile

First, initialize the input variables to `rttov_multi` function in your code. Next, one `rttov_multi` is executed on a thread in order to initialize the pinned host memory for input data for CUDA kernels (see below for description). After memory initialization, all the other threads running `rttov_multi` function are started.

3.2 Low level interface to RTTOV-7 GPU

In `rttov_multi` function, a call to a function `prfin` is made to set up profile-dependent variables for subsequent radiative transfer calculations by CUDA kernels, which are called from `rttov`. Input and outputs to CPU function `prfin` and CUDA functions are listed in Appendix A. After that, the following data is transferred for CUDA kernels from CPU to GPU:

variable name	description
<code>pemis</code>	Surface emissivities
<code>plandfastem</code>	Microwave surface emissivity coefficients
<code>debyeprof</code>	Functions of debye terms of profile for cloud calc
<code>xxm</code>	Func. of profile for mixed gas
<code>xxw</code> for water vapour
<code>xxo</code> for ozone
<code>xxc</code> for cloud liquid water
<code>temp</code>	Temperature profile in K
<code>fracps</code>	Fraction of std press level interval by which surface is above a predefined level
<code>fracpc</code>	Fraction of std press level interval by which cloud is above a predefined level
<code>ta</code>	Surface air temperature in K
<code>ts</code>	Surface skin temperature in K
<code>cldf</code>	Fractional (ir) cloud cover
<code>nlevsf</code>	Index of nearest std press level at/below surface
<code>nlevcd</code>	Index of nearest std press level at/below cloud top

Next, the following five CUDA kernels are executed:

<code>opdep_multi</code>	Calculates optical depths from every pressure level to space, transmittances from each level to space and transmittances from surface to space
<code>plncx</code>	Converts atmospheric temperatures to Planck functions
<code>emiss</code>	Sets up surface emissivity for radiative transfer calculation
<code>rtint</code>	Integration of radiative transfer equation
<code>brigrv</code>	Converts radiances to brightness temperatures

Output from GPU to CPU:

variable name	description
<code>prad</code>	Radiances (mw/cm-1/ster/sq.m)
<code>ptb</code>	Brightness temperatures (K)
<code>rado</code>	Overcast radiance at given cloud top in mw/m2/sr/cm-1
<code>tausfc</code>	Transmittance from surface

tau Transmittance from each standard pressure level ¹

3.3 Multi-profile CUDA kernels on 1 GPU

Without **MULTI_GPU** pre-processor directive *tstrad_multi.cu* and *rttov_multi.cu* illustrate how to initialize a single GPU and to call CUDA kernels that compute RTTOV-7 forward model for multiple profiles using a single CUDA kernel call. In this case, the use of the routines is significantly simplified compared to multiple GPU case as there are no threads running on multiple GPU cores to worry about.

3.4 Single-profile CUDA kernels on 1 GPU

Files *tstrad.cu* and *rttov.cu* illustrate how to initialize a single GPU and to call CUDA kernels that compute RTTOV-7 forward model for a single profile. The input is similar to the multi-profile case above.

4 Source code

The source code of RTTOV-7 GPU consists of the following CUDA C and C files:

tstrad.cu	Main test program
rttovcf.c	Routine to read a RTTOV coefficient file
rttov.cu	Computes multi-channel level to space transmittances, top of atmosphere radiances and brightness temperatures for a profile
prfin.c	Sets up profile-dependent variables for subsequent radiative transfer calculations by other subroutines of rttov
prslev.c	Locate given pressures on array of fixed levels
debye.c	Sets up debye terms for radiative transfer calculations
prftau.c	Store profile variables for transmittance calc
opdep.cu	Calculates optical depths from every pressure level to space, transmittances from each level to space and transmittances from surface to space
plncx.cu	Converts atmospheric temperatures to Planck functions
emiss.cu	Sets up surface emissivity for radiative transfer calculation
rtint.cu	Integration of radiative transfer equation
briggv.cu	Converts radiances to brightness temperatures

¹ AMSU-A only

The EUMETSAT Network of Satellite Application Facilities		Development of GPU-based RTTOV-7 IASI and AMSU Radiative Transfer Models	Doc ID : NWPSAF-MO-VS-043 Version : 1.0 Date : 4 July 2011
---	---	--	--

The following three CUDA C files are kernels for both AMSU-A and IASI multi-profile versions. If a macro **M15** is defined then AMSU-A versions are compiled. Otherwise, IASI versions are compiled. For multi-profile files if macro **MULTI_GPU** is defined then multi GPU versions are compiled. Otherwise, single GPU versions are compiled. For multi GPU version the number of GPUs and their ids are defined in *tstrad_multi.cu* using **GPU_N** and **DEV** variables.

tstrad.cu rttov_multi.cu	Main test program for single-profile processing Computes multi-channel level to space transmittances, top of atmosphere radiances and brightness temperatures for multiple-profiles. If macro USE_TIMER is defined then processing times are printed to <i>stdout</i> .
tstrad_multi.cu	Main test program for multi-profile processing. If macro MEASURE_TIME is defined then processing times are printed to <i>stdout</i>

Source file *opdep_multi.cu* is CUDA C file for IASI multi-profile version. These are also four CUDA kernels for AMSU-A multi-profile version: *briggv_multi_15.cu*, *opdep_multi_15.cu*, *plncx_multi_15.cu* and *rtint_multi_15.cu*.

Appendix A: Parameters of RTTOV-7 GPU functions

Input to prfin is the following:

variable name	description
pav	Atmospheric profile variables
psav	Surface air variables
pssv	Surface skin variables
pcv	Cloud variables (0-1)
pemis	Surface emissivities (0-1)
knpf	Number of profiles
knchpf	Number of channels
pangl	Satellite local zenith angle (deg)
pangs	Solar zenith angle at surface (deg)
ksurf	Surface type index
ksat	Satellite index

Output for prfin is the following:

xxm	Functions of profile for mixed gas
xxw for water vapour
xxo for ozone
xxc for cloud liquid water
debyeprof	Functions of debye terms of profile for cloud calc
nstype	Surface type index; 1=sea, 0=land 2=ice
xpath	Secant of viewing path angle at surface
snad2	Sine nadir angle squared
cnad2	Cos nadir angle squared
czen	Cosine zenith angle
czen2	Cos zenith angle squared
szen2	Sine zenith angle squared
nlevsf	Index of nearest std press level at/below surface
fracps	Fraction of std press level interval by which surface is above level nlevsf
surfw	Surface wind-speed m/s
nlevcd	Index of nearest std press level at/below cloud top
fracpc	Fraction of std press level interval by which cloud is above level nlevcd
ts	Surface skin temperature in K
temp	Temperature profile in K
wmix	Specific humidity profile in ppmv
ozon	Ozone profile in ppmv
cldw	Cloud liquid water in kg/kg
emis	Surface emissivity (0-1)
ta	Surface air temperature in K
cldf	Fractional (ir) cloud cover
plandfastem	Fastem land microwave surface emissivity coefficients

Input to opdep_multi:

knchpf	Number of channels
xxm	Functions of profile for mixed gas
xxw for water vapour
xxo for ozone
xxc for cloud liquid water
debyeprof	Functions of debye terms of profile for cloud calc
pitch	Width of the allocated multi-dimensional arrays in bytes
cfm	Mixed gas coeffs
cfw	Water vapour coeffs
cfo	Ozone coeffs
freq	Channel frequencies in GHz
gamma	<i>Gamma factor</i> transmittance corrections
njplev	Number of pressure levels
fmv_gas	Number of active gases
sensor	Sensor number for each triplet
nmwclctop	Upper level for lwp calcs
nlevsf	Index of nearest std press level at/below surface
fracps	Fraction of std press level interval by which surface is above level nlevsf
htod_len	Length of input data for a profile in floats
dtoh_len	Length of output data for a profile in floats

Output from opdep_multi:

opdp	Optical depth from press level to space
tau	Transmittance from each standard pressure level (0-1)
taufc	Transmittance from surface (0-1)

Input to plncx:

ptemp	Brightness temperatures
knchpf	Width in bytes of the allocated arrays
tc1	Band correction coefficient: offset in K
tc2	Band correction coefficient: slope in K/K
bcon1	1st Planck function constant in $mW/m^{**2}/sr/cm^{**}-4$
bcon2	2nd Planck function constant in $K/cm^{**}-1$
pitch	Width of the allocated multi-dimensional arrays in bytes
ptemp2	Brightness temperatures
ptemp3	Brightness temperatures

Output from plncx:

b	Planck function for temperatures profiles
ba	Planck functions surface air temp
bs	Planck functions surface skin temp

Input to emiss:

knchpf	Number of processed radiances
tausfc	Transmittances from surface to space (0-1)
nstype	Surface type index; 1=sea, 0=land 2=ice
xpath	Secant of viewing path angle at surface
snad2	Sine nadir angle squared
cnad2	Cos nadir angle squared
czen	Cosine zenith angle
czen2	Cos zenith angle squared
szen2	Sine zenith angle squared
surfw	Surface wind-speed m/s
ts	Surface skin temperature in K
emis	Surface emissivity (0-1)
sensor	Sensor number for each triplet
freq	Channel frequencies in GHz
emcir	Satellite and ir channel for surf emissivity model ssirem
wvnum	Wavenumber in cm^{-1}
mpol	Polarization of each channel
emc	Emissivity model data
plandfastem	Microwave surface emissivity coefficients
pitch	Width of the allocated multi-dimensional arrays in bytes

Output from emiss:

pems	Surface emissivities (0-1)
pref	Surface reflectivities (0-1)

Input to rtint:

ems	Surface emissivities (0-1)
refl	Surface reflectivities (0-1)
emis	Surface emissivity (0-1)
fast_emis	1 if emissivity was computed using fast in-place method, 0 otherwise
tau	Transmittances from each level to space (0-1)
tausfc	Transmittances from surface to space (0-1)
b	Planck functions for temperatures profiles
ba	Planck functions for surface air temp
bs	Planck functions for surface skin temp
nlevsf	Index of nearest std press level at/below surface
fracps	Fraction of std press level interval by which surface is above a predefined level
nlevcd	Index of nearest std press level at/below cloud top
fracpc	Fraction of std press level interval by which cloud is above a predefined level
pitch	Width of the allocated multi-dimensional arrays in bytes
cldf	Fractional (ir) cloud cover
tc1	Band correction coefficient: offset in K
tc2	Band correction coefficient: slope in K
bcon1	1st Planck function constant in $mW/m^{**2}/sr/cm^{**4}$
bcon2	2nd Planck function constant in K/cm^{**1}
knchpf	Number of channels
lcloud	Switch for cloud computation
sensor	Sensor number for each triplet
njplev	Number of pressure levels

Output from rtint:

prad	Output array of radiances
rado	Overcast radiances for given cloud-top pressures
bdt	Stores upwelling radiation from atmosphere above each level
bdtr	Stores down-welling radiation at each level from atmosphere above
ztcold	Intermediate temperature array
zbcold	Intermediate temperature array

Input to brigv:

knchpf	Number of channels
prad	Radiances
tc1	Band correction coefficient: offset in K
tc2	Band correction coefficient: slope in K
bcon1	1st Planck function constant in $mW/m^{**2}/sr/cm^{**4}$
bcon2	2nd Planck function constant in K/cm^{**1}

Output from brigv:

ptb	Brightness temperatures
-----	-------------------------

Appendix B: CUDA enabled GPUs

GPUs	Cards	Compute capability (version)
G80	GeForce 8800GTX/Ultra/GTS, Tesla C/D/S870, FX4/5600, 360M	1.0
G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce 8400GS/GT, 8600GT/GTS, 8800GT, 9600GT/GSO, 9800GT/GTX/GX2, GTS 250, GT 120/30, FX 4/570, 3/580, 17/18/3700, 4700x2, 1xxM, 32/370M, 3/5/770M, 16/17/27/28/36/37/3800M, NVS420/50	1.1
GT218, GT216, GT215	GeForce 210, GT 220/40, FX380 LP, 1800M, 370/380M, NVS 2/3100M	1.2
GT200, GT200b	GTX 260/75/80/85, 295, Tesla C/M1060, S1070, CX, FX 3/4/5800	1.3
GF100, GF110	GTX 465, 470/80, Tesla C2050/70, S/M2050/70, Quadro 600,4/5/6000, Plex7000, 500M, GTX570, GTX580	2.0
GF108, GF106, GF104	GT 420/30/40, GTS 450, GTX 460	2.1