# GPU Optimizations for Atmospheric Chemical Kinetics

Theodoros Christoudias
christoudias@cyi.ac.cy
The Cyprus Institute
Nicosia, Cyprus

Timo Kirfel
Astrid Kerkweg
Domenico Taraborrelli
Forschungszentrum Jülich GmbH, IEK-8
Jülich, Germany

Georges-Emmanuel Moulard
Erwan Raffin
Center for Excellence in Performance Programming, Atos
Rennes, France

Victor Azizi
Gijs van den Oord
Ben van Werkhoven
Netherlands eScience Center
Amsterdam, Netherlands

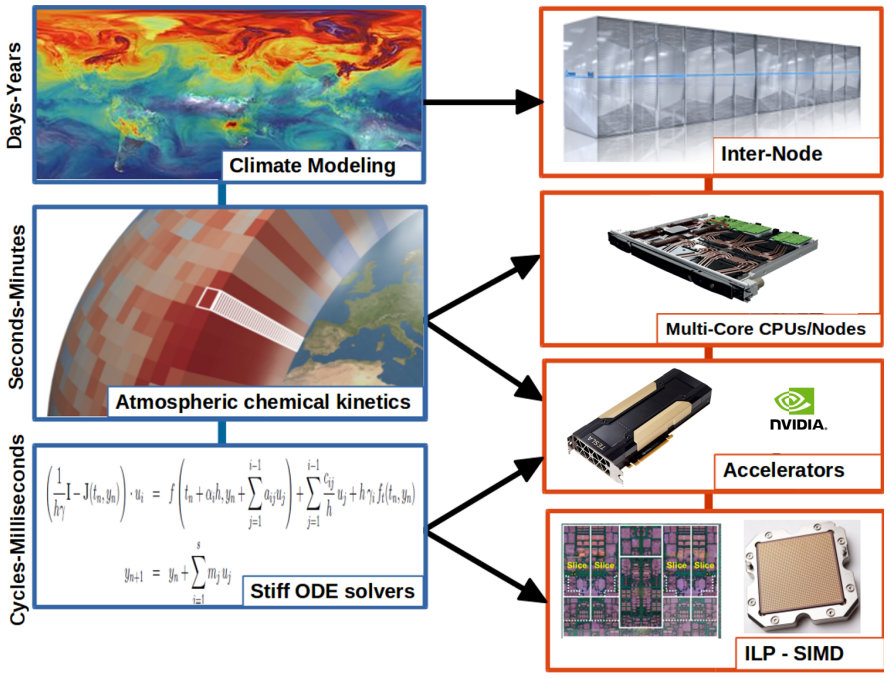


**Figure 1: Mutli-scale approach of accelerator optimizations for climate modeling and chemical kinetics.**

## ABSTRACT

We present a series of optimizations to alleviate stack memory overflow issues and improve overall performance of GPU computational kernels in atmospheric chemical kinetics model simulations. We use heap memory in numerical solvers for stiff ODEs, move chemical reaction constants and tracer concentration arrays from stack to global memory, use direct pointer indexing for array memory access, and use CUDA streams to overlap computation with memory transfer to the device. Overall, an order of magnitude reduction in GPU memory requirements is achieved, allowing for simultaneous offloading from multiple MPI processes per node and/or increasing the chemical mechanism complexity.


*HPCAsia 2021, January 20–22, 2021, Virtual Event, Republic of Korea*

ACM ISBN 978-1-4503-8842-9/21/01.
https://doi.org/10.1145/3432261.3439863

## CCS CONCEPTS

• **Hardware** → **Memory test and repair**; **Testing with distributed and parallel systems**; • **Computing methodologies** → *Massively parallel algorithms*; Model verification and validation.

## KEYWORDS

GPU, Memory test, Parallel systems, Distributed computing, CUDA

**ACM Reference Format:**
Theodoros Christoudias, Timo Kirfel, Astrid Kerkweg, Domenico Taraborrelli, Georges-Emmanuel Moulard, Erwan Raffin, Victor Azizi, Gijs van den Oord, and Ben van Werkhoven. 2021. GPU Optimizations for Atmospheric Chemical Kinetics . In *The International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia 2021), January 20–22, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3432261.3439863

## 1 INTRODUCTION

The ECHAM/MESSy Atmospheric Chemistry (EMAC) model is a numerical chemistry and climate simulation system that includes sub-models describing tropospheric and middle atmosphere processes and their interaction with oceans, land and human influences [4]. It uses the second version of the Modular Earth Submodel System (MESSy2) to link multi-institutional computer codes. The core atmospheric model is the 5th generation European Centre Hamburg general circulation model (ECHAM5) [6].

The acceleration of atmospheric chemical kinetics [3] and refactoring for new technologies (such as GPU accelerators) can reduce the required CPU-nodes and time-to-solution by a factor of 5–10 [1], with an order of magnitude more complex atmospheric chemical mechanism (in terms of number of species and reactions) compared to the current state-of-the-art, towards enabling Exascale performance [2].

The objectives of this work are to:

(1) Optimize performance for current and future GPU technologies, including NVIDIA Volta GPU and later, and
(2) Extend computational capability to be able to handle an order of magnitude more complex chemistry, such as the Mainz Organic Mechanism (MOM) [7].

## 2 BACKGROUND

EMAC is a distributed memory application comprising:

- A non-local dynamical (spectral) part with low scalability, exclusively relying on the Message Passing Interface (MPI) for parallelization.
- Local physical/chemical processes that can be massively parallelized with high scalability, ported on GPU architectures using the MEDINA (MECCA Development in Accelerators) code-to-code parser.

The EMAC MECCA sub-model [7] numerically solves the ordinary differential equations (ODEs) describing atmospheric chemical kinetics. In typical climate simulations, chemical kinetics can take up to 90% of the execution time. Each CPU process that offloads to the GPU requires a chunk of the GPU VRAM memory, whose size depends on the number of species and reaction constants in the MECCA mechanism. The number of GPUs per node and VRAM memory available in each GPU dictates the total number of CPU processes that can run concurrently.

## 3 GPU MEMORY ALLOCATION

Stack frame memory or local memory is allocated for all available threads of the GPU. For instance, on a V100 accelerator that has 2048 active threads per streaming multiprocessor (SM) and 80 SMs, at runtime the kernel will allocate a total of 2048 $\times$ 80 $\times$ (size of local memory) Bytes in the VRAM of the GPU. Thus, for the same code and problem size the total stack memory requirement is higher for newer generation GPUs, as they have more SM units and more active threads per SM. For comparison, an older generation K40 accelerator employs a maximum of 2048 active threads in 15 SMs (compared to 80 SMs in V100). However, as stack memory is allocated for the maximum total number of threads running concurrently when multiple CPU processes are offloading work simultaneously, the GPU quickly runs out of memory.

As an example, when declaring a local array with 5000 double-precision elements on the GPU, approximately 40 kBytes of stack memory are required per thread. If the kernel only uses 4096 threads, the theoretical amount of memory needed to be allocated in the VRAM at runtime is 164 MBytes. But as the stack memory will be allocated for the maximum number of threads a GPU can run concurrently, the kernel will be be using more than 6 GBytes on a V100 (whereas the same code would be using ~1 GByte on a K40 generation accelerator). So, on each V100 with 32 GBytes of memory, no more than 5 processes can run concurrently, even though theoretically more than 200 concurrent processes can be supported.

The NVIDIA Multi-Process Service (MPS) can be used to force each process launching a kernel on a GPU to occupy only a part of the available threads, which limits the total memory requirement of the MPI processes offloading to the GPU. In our tests, setting the CUDA MPS active thread percentage to 10% in our runtime environment allowed us to run with ten times more processes per GPU, thus keeping the CPU thread count high on each individual compute node for better balancing the CPU/GPU application heterogeneity. However, use of this setting can be regarded as a limiting workaround, rather than a solution.

To improve the overall performance of the computational kernel for atmospheric chemical kinetics, it is thus critical to overcome the GPU stack memory overflow.

## 4 OPTIMIZATIONS OVERVIEW

To alleviate the GPU memory overflow issues and improve overall performance of the computational kernel we:

- Use heap memory in numerical solvers for stiff ODEs
- Move chemical reaction constants from stack to global memory
- Transfer tracer concentration arrays to global memory
- Use direct pointer indexing for array memory access
- Use streams to overlap computation with device to-and-from memory transfers [8]

Moreover, the MPS can be used without the "active thread percentage" variable set, which still allows kernel and `memcopy` operations from different processes to overlap on the GPU, achieving higher utilization and shorter running times according to NVIDIA.

## 5 BENCHMARK METHODOLOGY

The results presented in this work were obtained on a benchmark representative of a real-world application, simulating one model month at 15 min timesteps with 142 chemical species, 310 chemical reactions, $128 \times 64 \times 31 = 253952$ gridpoints (longitude, latitude,

**Table 1: Time to solution for the benchmark one-month simulation of the original reference implementation that could only use 16 MPI processes per compute node, and the optimized version including all memory optimizations that can now take advantage all 40 available MPI processes (one per physical processor core). Note that using Nvidia Multi-Process-Service (MPS) but without setting the active thread percentage (ATP) achieves the best time to solution.**

| Code Version | Time to solution (s) | Speedup |
|---|---|---|
| Original reference<br>16 MPI processes/node (max. possible) | 13504 | 1× |
| Optimized w/o MPS<br>40 MPI processes/node | 7674 | 1.76× |
| Optimized w/ MPS+ATP=10%<br>40 MPI processes/node | 7633 | 1.77× |
| Optimized w/ MPS<br>40 MPI processes/node | 7402 | 1.82× |

**Table 2: GPU Memory use for the benchmark one-month simulation with different optimizations. The achieved memory reduction is at least of order five between reference and with all optimizations. Note: the active thread percentage ATP at 10% does not result in memory reduction by a proportional factor of 10.**

| Optimisation | GPU Memory per MPI Process (MiB) | | |
|---|---|---|---|
| | Nominal | MPS | MPS+ATP=10% |
| Reference (No optimisation) | 7655 | 7653 | 801 |
| + Global reaction constants and kernel arrays | 2313 | 2311 | 415 |
| + Pointer direct indexing | 1549 | 1549 | 335 |

vertical levels respectively) – at a spherical truncation of T42 (corresponding to a quadratic Gaussian grid of $\sim 2.8 \times 2.8^{\circ}$ in latitude and longitude) and using the Rosenbrock 3-step implicit Runge–Kutta numerical integrator.

When compiling CUDA we have used the `nvcc` compiler switch `--ptxas-options=-v` to obtain the memory that the kernel will use at runtime per thread. This amount includes the stack frame memory declared by the user for each thread.

The results have been obtained on an Atos BullSequana XH2000 supercomputer equipped with compute nodes having the following specifications: 2× Intel Skylake Xeon 6148 (20 cores @ 2.4 GHz), 4× NVIDIA Tesla V100 32GB SXM2, 394 GB RAM DDR4 2667 MT/s.

## 6 RESULTS

The achieved performance after these optimizations in terms of *i)* runtime and *ii)* required memory is presented in Table 1 and Table 2 respectively.

As a result of our optimizations, it is now possible to use at least 40 MPI processes per node concurrently, compared to a maximum possible of 16 MPI processes originally, improving the overall time to solution by a factor of 1.82× at node level. This performance is achieved without the use of the MPS active thread percentage feature, resulting in a more efficient and future-proof implementation.

The overall speedup offered by the use of CUDA streams is not significant on V100 as it is constrained here by the size of the chemical ODE system, and it is still ongoing work.

## 7 OUTLOOK

The improved source-to-source parser and Rosenbrock-family implicit ODE solver CUDA code can be used by other researchers to transform chemistry kinetics packages to CUDA-accelerated code. The methodology can be used as a template for enabling high performance computing capability on GPU architectures for atmospheric modelling applications. The code is available on Github[1] under the MIT license.

## ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988 (ESiWACE2) and was in part supported by the NVIDIA Application Lab at JSC [5].

## REFERENCES

[1] Michail Alvanos and Theodoros Christoudias. 2017. GPU-accelerated atmospheric chemical kinetics in the ECHAM/MESSy (EMAC) Earth system model (version 2.52). *Geoscientific Model Development* 10, 10 (2017), 3679.

[2] Michail Alvanos and Theodoros Christoudias. 2019. Accelerating Atmospheric Chemical Kinetics for Climate Simulations. *IEEE Transactions on Parallel and Distributed Systems* 30, 11 (2019), 2396–2407.

[3] Theodoros Christoudias and Michail Alvanos. 2016. Accelerated chemical kinetics in the EMAC chemistry-climate model. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 886–889.

[4] Patrick Jöckel, Astrid Kerkweg, Andrea Pozzer, Rolf Sander, Holger Tost, Hella Riede, Andreas Baumgaertner, Sergey Gromov, and Bastian Kern. 2010. Development cycle 2 of the modular earth submodel system (MESSy2). *Geoscientific Model Development* 3 (2010), 717–752.

[5] Jülich Supercomputing Centre. 2018. JURECA: Modular supercomputer at Jülich Supercomputing Centre. *Journal of large-scale research facilities* 4, A132 (2018). https://doi.org/10.17815/jlsrf-4-121-1

[6] E Roeckner, R Brokopf, M Esch, MA Giorgetta, S Hagemann, L Kornblueh, E Manzini, U Schlese, and U Schulzweida. 2006. Sensitivity of simulated climate to horizontal and vertical resolution in the ECHAM5 atmosphere model. *Journal of Climate* 19, 16 (2006), 3771–3791.

[7] R. Sander, A. Baumgaertner, D. Cabrera-Perez, F. Frank, S. Gromov, J.-U. Grooß, H. Harder, V. Huijnen, P. Jöckel, V. A. Karydis, K. E. Niemeyer, A. Pozzer, H. Riede, M. G. Schultz, D. Taraborrelli, and S. Tauer. 2019. The community atmospheric chemistry box model CAABA/MECCA-4.0. *Geoscientific Model Development* 12, 4 (2019), 1365–1385. https://doi.org/10.5194/gmd-12-1365-2019

[8] Ben van Werkhoven, Jason Maassen, Frank J Seinstra, and Henri E Bal. 2014. Performance models for CPU-GPU data transfers. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 11–20.

[1] https://github.com/CyprusInst/medina