

GPU Acceleration at Scale with OpenPower platforms in *Code_Saturne*

Samuel Antao

IBM Research

Daresbury, UK

samuel.antao@ibm.com

Charles Moulinec

STFC

Daresbury, UK

charles.moulinec@stfc.ac.uk

Yvan Fournier

EDF R&D

Paris, France

yvan.fournier@edf.fr

Robert Sawko

IBM Research

Daresbury, UK

rsawko@uk.ibm.com

Malgorzata Zimon

IBM Research

Daresbury, UK

malgorzata.zimon@uk.ibm.com

Christopher Thompson

IBM Research

Daresbury, UK

christthompson@uk.ibm.com

Alex Skillen

STFC

Daresbury, UK

alex.skillen@stfc.ac.uk

Juan Uribe

EDF R&D UK Centre

Manchester, UK

juan.uribe@edfenergy.com

David R. Emerson

STFC

Daresbury, UK

david.emerson@stfc.ac.uk

Abstract—*Code_Saturne* is a widely used computational fluid dynamics software package that uses finite-volume methods to simulate different kinds of flows tailored to tackle multi-billion-cell unstructured mesh simulations. This class of codes has shown to be challenging to accelerate on GPUs as they consist of many kernels and regular inter-process communication in between. In this poster we show how template pack expansion with CUDA can combine multiple kernels into a single one reducing launching latencies and along with the specification of data environments help reduce host-device communication. We tested these techniques on ORNL Summit Supercomputer based on OpenPOWER platform delivering almost 3x speedup over CPU-only runs on 256 nodes. We also show how the latest generation NVLINK™ interconnect available in POWER9™ improves scaling efficiency, enabling consistent GPU acceleration with just 100K-cells per process.

Index Terms—Computational Fluid Dynamics (CFD), GPU, NVLINK, OpenPOWER, Acceleration

I. INTRODUCTION

Computational Fluid Dynamics (CFD) has been one of the most important classes of codes targeted by High-performance Computing (HPC) facilities around the world [1]. *Code_Saturne* is one of these codes comprising C/C++, Fortran and Python, with MPI and OpenMP support [2]–[4]. It is an open-source software package developed and maintained by EDF R&D to target very large and complex CFD problems. EDF is one of the leading energy suppliers in Europe, running several nuclear power plants, and *Code_Saturne* has been instrumental to simulate multiple components related with the operation and design of current and next-generation plants, aiming at the simulation of the full nuclear reactors. Therefore, it is key that *Code_Saturne* is able to deliver the required performance at scale for its simulations.

The use of accelerators, in particular GPU devices, has become the preferred way to boost application performance as technology does not scale anymore with Moore’s law. GPUs can now be found in many top-tier supercomputers, including the ones based on OpenPOWER platforms like Summit (Oak Ridge National Laboratory) OpenPOWER platform. Many CFD applications, in particular the ones that use unstructured discretisations like *Code_Saturne*, have been known to be hard to accelerate on GPUs for the following main reasons: *i)* unstructured codes map the domain into sparse matrices but GPUs are tailored for dense problems, *ii)* domain boundaries need to be shared regularly and rank collectives need to be used to compute residuals and control convergence, and *iii)* complex models have been

developed over many years and require many kernels scattered in the code base with any kind of dependences between them, making GPU-kernel candidate extraction unfeasible without major code-base disruption.

II. PRIOR ART AND GOALS

Relatively good solutions to tackle *i)* have been developed, with many accelerated sparse algebra libraries like cuSparse [5], and therefore it is not a major limiting factor. In our work we use reference implementations of sparse matrix operations and other vector operations [6].

A consequence of *ii)* is the constant need to move data to/from the device to be shared across ranks as well as frequent barriers for application-wide collectives that preclude streaming. The impact of *iii)* is translated into latencies for launching many small kernels and communication of small quantities between host and device. Some attempts have been made to accelerate parts of the code like [7] for conjugate gradient or the use of PETSc as an acceleration engine. However, the practicality in real applications is still hindered by *ii)* and *iii)*, as the problem size per rank/node gets smaller. In this poster we propose a methodology to tackle *ii)* and *iii)*. To test it we use a 3D-laminar driven cavity solving the velocity using Gauss Seidel (GS) and the pressure an Algebraic Multigrid (AMG), which exposes the problems above with relative low amount of compute. If the compute amount increases the relative impact of the problems above decreases as GPUs can easily accommodate more compute.

III. METHODOLOGY

We propose to combine three techniques to tackle the problems *ii)* and *iii)* mentioned above:

a) Use template pack expansion to produce specialised CUDA kernels for any sequence of matrix/vector operations needed in the code: Figure 1 shows how this can be done. `any_kernels` is the only device entry point (CUDA kernel). This templated CUDA kernel takes a list of enumerators `Kinds`, each specifying a vector/matrix operation. It also takes an array `KernelArgsSeriesGPU` in CUDA constant memory with the arguments for each of these operations. The template pack is then expanded along with the right indexing to the array of arguments, and `any_kernel` is called for each operation. The function `any_kernel` contains a switch case statement where each case is a vector/matrix operation. The switch is controlled by the template argument `Kind`, hence each instance of `any_kernel` is optimised into a single vector/matrix operation. The result is a given instance of `any_kernels` to be a specialised sequence of vector/matrix operations without any launching latencies

POWER9 scaling experiments were only possible with Summit Early Access programme sponsored by Oak Ridge Leadership Computing Facility. Work partially funded by UKTC consortium grants EP/L000261/1 and EP/R029326/1.

between them. We also control the amount of rows per block, so that each block always produce results for the same rows and, with this, avoid inter-block synchronisations. The outcome is similar to task abstractions found in programming models like OpenMP [8], but with tasks handled/queued at compile time with CUDA.

```
template<KernelKinds... Kinds>
__global__ void any_kernels(void) {
    auto *KA = reinterpret_cast<KernelArgsSeries*>(&KernelArgsSeriesGPU[0]);
    const unsigned n_rows_per_block = KA->RowsPerBlock;
    unsigned idx = 0;

    int dummy[] = { any_kernel<Kinds>(KA->Args[idx++], n_rows_per_block)... };
    (void) dummy;
}
```

Fig. 1: Template pack expansion to combine multiple operations into a single CUDA kernel.

b) Manage device memory explicitly: When the application starts, a pool of memory is allocated and reused for all data that needs to be on the device. As the amount of variables live on the device is small, the pool management is lightweight. The implementation can with this combine arguments for consecutive kernels and create data environments so that data stays on the device between series of operations, reducing latencies associated with data movements.

c) Use NVIDIA Multi-Process Service to increase GPU utilisation by having multiple ranks using the same GPU: As there are still tasks that need to be handled by the CPU, (e.g. ghost cell sampling prior to communication with other ranks) and data to be moved to/from the device, the GPU could sometimes be idle. Overlapping execution of multiple ranks in the GPU allows the GPU-compute of a given rank to overlap data movements or CPU-compute of some other rank, improving utilisation and efficiency.

IV. RESULTS

We tested our methodology with the configuration in Table I. Figure 2 shows how the proposed techniques improve performance for our target testcase which couples GS and AMG algorithms. Figure 3 compares CPU-only and CPU+GPU runs. Consistent speedup is observed (over 2x) even for mesh partitions as small as 100K-cell per rank. Even better scaling performance are shown using POWER9 versus POWER8 machines - 16x more resources can be efficiently used for a problem that is only 8x larger. NVLINK 2.0 (POWER9) plays an important role as it almost halves data movement latencies comparing to NVLINK 1.0 (POWER8). Efficiencies higher than 100% can explained with more advantageous domain partitions for that specific number of ranks.

	Cores	Ranks	GPUs	Ranks/GPU	OpenMP thd./rank
POWER8 only	20	20	4	-	8
POWER8 + GPU				5	
POWER9 only	42	21	6	-	4
POWER9 + GPU		18		3	

TABLE I: Configuration per node for both target systems. The chosen settings are the ones delivering better performance.

V. CONCLUSIONS

In this poster we propose a set of techniques to improve GPU utilisation in *Code_Saturne* by aggressively reducing idle times and data movement latencies. We combine template pack expansion, explicit device memory pool management and the NVIDIA Multi-Process Service to enable over 2x speedup over CPU-only runs even when using only 100K-cell per rank, improving strong scaling *Code_Saturne* performance and enabling it to tackle larger problems at scale. All the techniques proposed in this poster can be applied

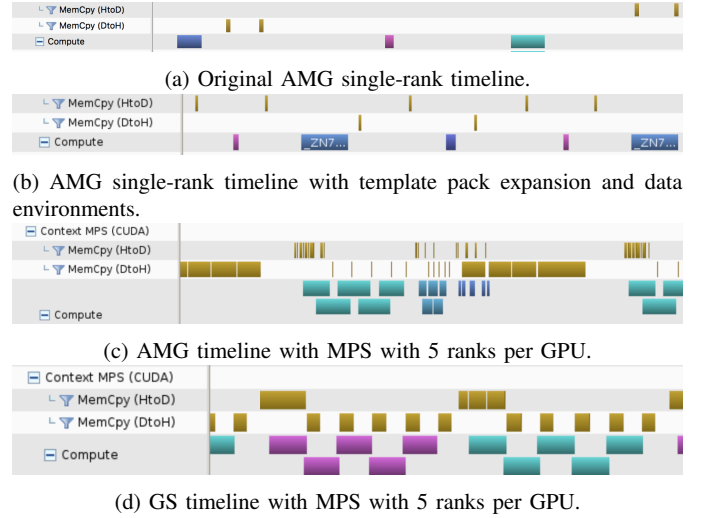
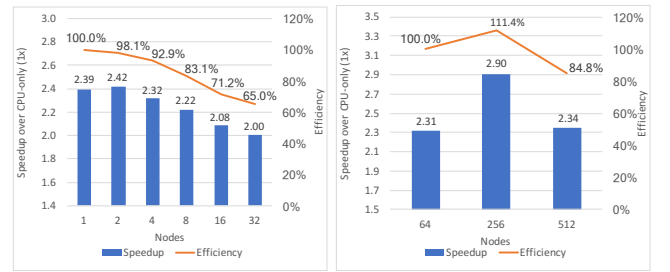


Fig. 2: Execution timelines on POWER8+GPU for AMG and GS. (2a) Large latencies between compute kernels and data movements. (2b) Idle times are greatly reduced. (2c) Data movements is hidden by overlapping multiple ranks. (2d) Data movements almost completely hidden for GS.



POWER8 + P100 GPUs (111M-cells). POWER9 + V100 GPUs (889M-cells).¹

Fig. 3: CPU+GPU speedup over CPU-only and efficiency (strong). Speedup is reported on same number of nodes.

to other codes whose acceleration at scale is limited by latencies resulting from running multiple small kernels.

REFERENCES

- [1] A. Turner, "Parallel software usage on UK national HPC facilities 2009-2015: How well have applications kept up with increasingly parallel hardware?" EPCC, Tech. Rep., 04 2015.
- [2] EDF. (2018) Code_saturne official web page. [Online]. Available: <https://www.code-saturne.org>
- [3] F. Archambeau, N. Méchitoua, and M. Sakiz, "Code_saturne: a finite volume code for the computation of turbulent incompressible flows," *International Journal on Finite Volumes*, vol. 1, 2004.
- [4] Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A. Sunderland, and J. Uribe, "Optimizing code_saturne computations on petascale systems," *Computers & Fluids*, vol. 45, no. 1, pp. 103 – 108, 2011, 22nd Int. Conference on Parallel Computational Fluid Dynamics (ParCFD 2010).
- [5] NVIDIA. (2018) cuSparse official documentation web page. [Online]. Available: <https://docs.nvidia.com/cuda/cusparse/>
- [6] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA, Tech. Rep., 11 2008.
- [7] H. Anzt, M. Baboulin, J. Dongarra, Y. Fournier, F. Hulsemann, A. Khabou, and Y. Wang, "Accelerating the conjugate gradient algorithm with GPUs in CFD simulations," in *High Performance Computing for Computational Science – VECPAR 2016*, 2017, pp. 35–43.
- [8] OpenMP Architecture Review Board, "Openmp application programming interface version 4.5," OpenMP ARB, Tech. Rep., 11 2015.