OpenACC acceleration of the Nek5000 spectral element code

Stefano Markidis¹, Jing Gong¹, Michael Schliephake¹, Erwin Laure¹, Alistair Hart², David Henty³, Katherine Heisey⁴ and Paul Fischer⁴

Abstract
We present a case study of porting NekBone, a skeleton version of the Nek5000 code, to a parallel GPU-accelerated system. Nek5000 is a computational fluid dynamics code based on the spectral element method used for the simulation of incompressible flow. The original NekBone Fortran source code has been used as the base and enhanced by OpenACC directives. The profiling of NekBone provided an assessment of the suitability of the code for GPU systems, and indicated possible kernel optimizations. To port NekBone to GPU systems required little effort and a small number of additional lines of code (approximately one OpenACC directive per 1000 lines of code). The naïve implementation using OpenACC leads to little performance improvement: on a single node, from 16 Gflops obtained with the version without OpenACC, we reached 20 Gflops with the naïve OpenACC implementation. An optimized NekBone version leads to a 43 Gflop performance on a single node. In addition, we ported and optimized NekBone to parallel GPU systems, reaching a parallel efficiency of 79.9% on 1024 GPUs of the Titan XK7 supercomputer at the Oak Ridge National Laboratory.

Keywords
OpenACC, Nek5000, porting NekBone to GPU, optimization of NekBone with OpenACC

1. Introduction
Nek5000 is an open-source code for the simulation of incompressible flow (http://nek5000.mcs.anl.gov). Nek5000 is widely used in a broad range of applications, including the study of thermal hydraulics in nuclear reactor cores, the modeling of ocean currents, and the simulation of combustion in mechanical engines. The Nek5000 discretization scheme is based on the spectral element method (Patera, 1984). In this approach, the incompressible Navier–Stokes equations are discretized in space by using high-order, weighted residual techniques employing tensor-product polynomial bases. The resultant linear systems are computed using the conjugate gradient (CG) solver with convenient preconditioners. The tensor-product-based operator evaluation can be computed as matrix–matrix products.

The Nek5000 code won the Gordon Bell Prize in 1999 (Tufo and Fisher, 1999) and has recently demonstrated scalability up to one million processes on the Mira supercomputer at the Argonne National Laboratory (http://nek5000.mcs.anl.gov/index.php/Scaling). The code consists of 100,000 codes and it is written mainly in Fortran (70,000 lines of code) and in C (30,000 lines of code). Nek5000 uses the message passing interface (MPI) and domain decomposition for parallel computing on distributed memory supercomputers.

Heterogeneous HPC architectures are increasingly prevalent in the Top500 list, with CPU-based nodes enhanced by accelerators or coprocessors optimized for floating-point calculations. This trend will likely increase moving towards exascale and it is vital that relevant HPC applications are able to exploit heterogeneous architectures (Kogge et al., 2008; Dongarra et al., 2009). Whilst accelerators offer a large boost in peak system speed, it is hard to translate this into sustained application performance. For GPU accelerators, applications are typically rewritten in a bespoke, low-

¹HPCViz Department, KTH Royal Institute of Technology, Sweden
²Cray Exascale Research Initiative Europe, UK
³Edinburgh Parallel Computing Centre, Edinburgh University, UK
⁴Argonne National Laboratory, USA

Corresponding author:
Stefano Markidis, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden.
Email: markidis@kth.se
level language such as CUDA. This is a productivity drawback, with developers maintaining multiple versions of their codebase without guarantee of portability or permanence. OpenACC (http://www.openacc-standard.org; Ansaloni and Hart, 2011) enables existing HPC application codes to run on accelerators with minimal source code changes. This is done using compiler directives (specially formatted comments) and API calls. In OpenACC the compiler is responsible for generating optimized code with the user guiding only where necessary.

The GPU architecture has already been shown to be well suited to lattice Boltzmann (Gray et al., 2011) and stencil codes, including S3D (Chen et al., 2009), characterized by several relatively simple operations, which map very easily onto the GPU. By contrast, Nek5000 is a more computationally demanding code that implements a computational fluid dynamics (CFD) solver based on the spectral element method. The problem is spatially decomposed into many elements, which are distributed among the processing cores using MPI (Chen et al., 2009).

To our knowledge, we report here the first attempt of porting Nek5000 to a parallel GPU-accelerated system. The original Fortran source is used as the base, enhanced by OpenACC directives. The initial profiling first provided an assessment of suitability of the code for GPU systems, and then guided the optimization process. The performance results using OpenACC for single- and multi-GPUs are presented.

The paper is organized as follows. In Section 2, the NekBone mini-application is described in detail. The porting and optimization of the NekBone application for single- and multi-GPU systems is presented in Sections 3 and 4, respectively. Section 5 shows the performance results of different implementations. Finally, Section 6 summarizes the results and draws relevant conclusions.

2. NekBone – a skeleton version of Nek5000

A skeleton version of Nek5000, called NekBone, has been used in this paper. Currently, NekBone is configured to resemble very closely the basic structure and user interface of the Nek5000 software. It is in 3D and uses the communication kernel of the standard Nek5000 software. The NekBone code is used by international codesign teams such as CRESTA (http://www.cresta-project.eu/) and CESAR (https://cesar.mcs.anl.gov/).

To our knowledge, we report here the first attempt of porting Nek5000 to a parallel GPU-accelerated system. The original Fortran source is used as the base, enhanced by OpenACC directives. The initial profiling first provided an assessment of suitability of the code for GPU systems, and then guided the optimization process. The performance results using OpenACC for single- and multi-GPUs are presented.

The paper is organized as follows. In Section 2, the NekBone mini-application is described in detail. The porting and optimization of the NekBone application for single- and multi-GPU systems is presented in Sections 3 and 4, respectively. Section 5 shows the performance results of different implementations. Finally, Section 6 summarizes the results and draws relevant conclusions.

2. NekBone – a skeleton version of Nek5000

A skeleton version of Nek5000, called NekBone, has been used in this paper. Currently, NekBone is configured to resemble very closely the basic structure and user interface of the Nek5000 software. It is in 3D and uses the communication kernel of the standard Nek5000 software. The NekBone code is used by international codesign teams such as CRESTA (http://www.cresta-project.eu/) and CESAR (https://cesar.mcs.anl.gov/).

To our knowledge, we report here the first attempt of porting Nek5000 to a parallel GPU-accelerated system. The original Fortran source is used as the base, enhanced by OpenACC directives. The initial profiling first provided an assessment of suitability of the code for GPU systems, and then guided the optimization process. The performance results using OpenACC for single- and multi-GPUs are presented.

The paper is organized as follows. In Section 2, the NekBone mini-application is described in detail. The porting and optimization of the NekBone application for single- and multi-GPU systems is presented in Sections 3 and 4, respectively. Section 5 shows the performance results of different implementations. Finally, Section 6 summarizes the results and draws relevant conclusions.

Table 1. Profiling results of NekBone for Version 8.1 of the CCE on single node with GPU with the CrayPAT profiler. The table shows the most time-consuming subroutines in the NekBone mini-application.

<table>
<thead>
<tr>
<th>Function</th>
<th>Time (%)</th>
<th>Time (s)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>100.0</td>
<td>11.08</td>
<td>747</td>
</tr>
<tr>
<td>ax3D</td>
<td>81.8</td>
<td>9.07</td>
<td>20</td>
</tr>
<tr>
<td>glsc3</td>
<td>6.4</td>
<td>0.71</td>
<td>61</td>
</tr>
<tr>
<td>add2s2</td>
<td>5.9</td>
<td>0.65</td>
<td>60</td>
</tr>
<tr>
<td>add2s1</td>
<td>1.9</td>
<td>0.21</td>
<td>20</td>
</tr>
<tr>
<td>gather-double-add</td>
<td>1.5</td>
<td>0.16</td>
<td>40</td>
</tr>
<tr>
<td>scatter-double</td>
<td>1.4</td>
<td>0.15</td>
<td>40</td>
</tr>
</tbody>
</table>

Figure 1. Scheme of the NekBone computational cycle. The blue blocks of the algorithm are executed on the CPU, while the red blocks are executed on the GPU. After the matrix is initialized, a computation cycle comprising the solution of the linear solver with the CG solver is repeated 20 times.
carried out on a single node, and therefore the gs_op subroutine, that is responsible for the MPI communication, uses less computational time than in the case of multi-GPU systems.

3. Accelerating NekBone on a single GPU

As shown in Table 1, the main computations in NekBone are spectral transforms, that are based on computing several relatively small rectangular matrix multiplications. These independent matrix-matrix multiplications are executed in the ax3D subroutine. The initial Fortran code for the ax3D subroutine in NekBone is presented in Listing 1. In this code, nel is the number of spectral elements (typically 1024), and n is the order of the polynomials (usually chosen to be 16). The subroutine ax3D consists of five nested loops, where the external one has a much higher tripcount.

To port ax3D to GPU using OpenACC is a trivial task. It can be achieved by adding the OpenACC compiler directives to the previous code as shown in Listing 2, where the OpenACC directives appear in green. The compiler directives CopyIn(D,u,g) indicate moving D,u,g variables from the CPU memory to the GPU memory, while CopyOut(w) instructs the compiler to move w (the final result of the tensorial product) from the GPU memory to the CPU memory after the calculations. ACC KERNELS ... ACC END KERNELS indicate the part of code that will be executed on the GPU.

When optimizing the OpenACC ax3D code reported in Listing 2, it is important to understand the computational cost of each single loop of the original ax3D subroutine. For this reason, we carried out a loop-level profiling of the computational cost of the nested loops in the subroutine ax3D.

Table 2 shows the loop nests (rather than just the routines as in Table 1) that take significant portions of the runtime and the number of loop iterations. This is important if we are trying to understand how well the loops will be scheduled on the accelerator. The i,j,k and l loops all have a low tripcount (being n equal to 16). Therefore, the compiler will, by default, schedule all the iterations of the GPU, and no single loop has enough tripcount over a level of parallelism for the iterations to keep the GPU busy.

One optimization of the ax3D subroutine is to explicitly instruct the compiler to collapse all four i,j,k,l loops together. This gives the largest possible iteration space to divide into thread-blocks. This can be achieved by adding Collapse(4) as in Listing 3. We note that OpenACC (http://www.openacc-standard.org) offers two directives to accelerate code sections on GPU: kernels and parallel. Our experience with the Cray compiler, at least, shows that kernels are a very useful way of exploring initial acceleration, but parallel offers greater control over kernel scheduling and performance. For this reason, we used parallel for the optimization of the ax3D subroutine.

4. Porting and optimizing NekBone to multi-GPU systems

When porting NekBone to multi-GPU systems, we focus on the gs_op (gather-scatter operator) subroutine that performs the communication operations using the MPI library. For clarity of exposition, we consider the computation of the u variable on multi-GPU systems using NekBone. On distributed memory systems, the vector u is divided across different MPI processes that compute their part and communicate its boundary values to other neighboring MPI processes so they can perform additional calculations using the u from different MPI processes. In the OpenACC version of NekBone, the values of the u vector are entirely computed on the GPU but the communication to other MPI processes is performed only by the host. For this reason, the
Listing 2. OpenACC code for the ax3D subroutine in NekBone.

```fortran
!$ACC DATA COPYIN(D,u,g)

!$ACC COPYOUT(w)

!$ACC KERNELS
    do e=1,nel
        do k=1,n
            do j=1,n
                do i=1,n
                    temp = 0
                    do l=1,n
                        temp = temp + D(i,l)*u(l,j,k,e)
                    enddo
                    w(i,j,k,e) = g(i,j,k,e)*temp
                enddo
            enddo
        enddo
    enddo

!$ACC END KERNELS

!$ACC END DATA
```

Listing 3. Optimization of the OpenACC code for the ax3D subroutine in NekBone.

```fortran
!$ACC DATA PRESENT(u,w)

!$ACC PRESENT(D,g)

!$ACC PARALLEL LOOP COLLAPSE(4) GANG WORKER VECTOR VECTOR_LENGTH(128)
    do e=1,nel
        do k=1,n
            do j=1,n
                do i=1,n
                    temp = 0
                    !$ACC SEQ
                    do l=1,n
                        temp = temp + D(i,l)*u(l,j,k,e)
                    enddo
                    w(i,j,k,e) = g(i,j,k,e)*temp
                enddo
            enddo
        enddo
    enddo
```

Listing 2. OpenACC code for the ax3D subroutine in NekBone.

Listing 3. Optimization of the OpenACC code for the ax3D subroutine in NekBone.
variable $u$ needs to be moved first from GPU memory to the host memory so that MPI communication can be completed between different hosts. Finally, the correct $u$ variable can be moved back from the host to the GPU. Figure 2 presents a scheme of how the computation of $u$ occurs in multi-GPU systems.

Initially, we use the OpenACC directives simply to move the $u$ variable from the GPU to the host and vice versa. The gs_op subroutine performs the MPI communication between different hosts. This can be simply achieved by instructing the compiler with the directives !$ACC UPDATE HOST(u)$ and !$ACC UPDATE DEVICE(u)$ as in Listing 4.

However, only a small part of $u$ comprising the boundary values of the $u$ vector needs to be communicated from the GPU to/from CPU memory. To achieve this, the gs_op subroutine is split into three parts as in Listing 5. The local operations (the $u$ vector is multiplied by a matrix called Q) that can be computed without communication are performed first on the GPU ($u_G = Qu$). Then only the $u$ boundary values are moved to the host and communicated ($gs\_op(u_G, 1, 1, 0)$). Finally the GPU computes the new $u$ values after the communication with other processes ($u = Q^T u_G$).

This optimization allows us to reduce considerably the amount of data moved from GPU and CPU memories and vice versa, increasing the overall performance of the OpenACC multi-GPU version of the NekBone code.

### 5. Performance results

The performance tests of the NekBone version with OpenACC were initially carried out on a Cray XK6 consisting of four compute nodes comprising a 2.1 GHz AMD Interlagos 16-core processor, 16 GB of memory, and a Kepler K20 GPU. Version 8.1 of the Cray Compilation Environment (CCE) supporting OpenACC was used. The computational performance was measured in GFlops, that is billion floating-point operations per second.

We first tested the performance of NekBone with OpenACC on a single node comprising one CPU and one GPU. We used 512 elements and a 16th-order polynomial with a total number of points equal to 2,097,152. Three code versions were compared: the initial Fortran code, a version with ‘basic’ OpenACC, and an optimized OpenACC version of ax3D. Figure 3 presents the performance results of NekBone using only CPU (gray), the naive OpenACC implementation of NekBone (light brown), and the optimized OpenACC
version (dark brown). On the benchmark system, the initial NekBone version without using the GPU reached 16.27 Gflops, while the naïve implementation with OpenACC and GPU led only to a small improvement: 20.56 Gflops. However, the simple optimization of collapsing the four nested loops into one loop led to a significant performance improvement: 42.67 Gflops, more than twice the unoptimized OpenACC NekBone version.

The computational performance depends critically on the computational workload of the GPU. The more calculations that are completed on the GPU, the higher the performance. This is clear from Figure 4, showing the performance on a single GPU varying the number of elements (nel) and the spectral order (N). By increasing these two values, we increase the tripcount of the four nested loops in the matrix–matrix multiplication. The worst performance of approximately three Gflops is achieved at a low number of elements (32) and a low order (8).

Figure 5 shows the performance on a single GPU varying the number of elements, the spectral order,
In Listing 3, VL is set to 128, that is, the default value. As seen in Figure 4, by increasing the number of elements and the spectral order, the computational performance increases. In addition, it is clear, that the parameter VL does not have a strong effect on the computational performance unless a small value, such as 32, is used.

Finally, we present the scaling tests on a parallel GPU-accelerated system. We carried out weak scaling tests by keeping the workload per node fixed: 1024 elements per node with 16th-order polynomials, giving a total number of points equal to 4,194,304 per node. Figure 6 presents the parallel performance using one, two and four GPUs using the optimized gs_op subroutine. The parallel efficiency on two and four GPUs was 88% and 84%, respectively.

In addition to the Cray XK6, used for the previously presented tests, the Titan supercomputer was also used to study NekBone performance on a much higher number of GPUs. Titan is a Cray XK7 supercomputer, containing 18,688 AMD Opteron 6274 16-core CPUs and 18,688 NVIDIA Tesla K20X GPUs. For the tests on Titan, 512 elements per node and 20th-order polynomials were used for a total number of points per node of 4,096,000. Figure 7 shows the NekBone scaling performance, measured in Gflops for up to 1024 GPUs in semi-logarithmic scaling (red solid line), while the black dashed line represents the ideal weak scaling. The initial results on Titan are encouraging: the parallel efficiency on 1024 GPUs is 79.9%.

6. Conclusions
A skeleton version of the Nek5000 code, called NekBone, has been successfully ported to multi-GPU systems using OpenACC compiler directives. The focus of this work was on porting the most time-consuming routines of the NekBone to GPU system: the ax3D and gs_op subroutines. To port NekBone to GPU systems required little effort and a small number of additional lines of code. In fact, after the porting, the total number of lines of NekBone was 41,953 including 45
OpenACC directives. Approximately, one OpenACC directive was used per 1000 lines of code.

The naïve implementation using OpenACC leads to little performance improvement: from 16 Gflops obtained on the CPU, we reached 20 Gflops with the naïve OpenACC implementation. The optimization of matrix–matrix multiplication required evaluating the computational cost of loop-nesting to assist the developer in guiding the OpenACC loop scheduling. By simply instructing the compiler to collapse four nested loops in the matrix–matrix multiplication, we reached approximately 43 Gflops, doubling the performance of the naïve OpenACC implementation. In addition, we ported and optimized NekBone on a multi-GPU system by working on the gs_op subroutine. The optimized version for a multi-GPU system gave a parallel efficiency of 79.9% on a 1024 GPU system. Our current work focuses on improving overlapping MPI communication and OpenACC CPU–GPU memory transfer and computation.

Acknowledgements
Computing time on the Titan supercomputer at the Oak Ridge National Laboratory within the INCITE program is acknowledged.

Funding
This research received funding from two European Commission projects (grant numbers 87703, http://www.cresta-project.eu/ and 610598, http://www.epigram-project.eu/). KH and PF were supported by the US Department of Energy (contract number DE-AC02-06CH11357 and award number DE-FG02-08ER25835).

References


Author biographies

Stefano Markidis is an Assistant Professor in the High Performance and Visualization (HPCViz) department at the KTH Royal Institute of Technology, Stockholm, Sweden. He was a recipient of the 2005 R&D100 award and is the author of more than 50 articles in peer-reviewed journals. His research interests include parallel computing and computational physics.

Jing Gong completed his PhD in scientific computing on “Hybrid Methods for Unsteady Fluid Flow Problems in Complex Geometries” at Uppsala University, Sweden in 2007. He also holds an MSc in scientific computing from the KTH Royal Institute of Technology and an MEng in mechatronics from Beihang University, Beijing, China. He joined the Center for High Performance Computing as an Application Expert in computational fluid dynamics in January 2012.

Michael Schliephake started working at the Center for High Performance Computing in January 2010. As a System Administrator, he concentrates on the operation of the Center’s HPC systems. After spending several years as a Software Developer creating engineering software, he began working in the area of HPC at the High Performance Computing Center in Stuttgart, Germany. There, Michael gained experience in the installation and operation of cluster systems as well as in several projects which were associated with the design and implementation of supercomputing and grid infrastructures. These projects were the German D-Grid Initiative and the European projects DEISA and PRACE. Furthermore, he brings with him his past experience educating students in HPC. Michael obtained his qualified engineer degree at the Technical University in Leipzig, Germany.

Erwin Laure is Director of the Center for High Performance Computing at the KTH Royal Institute of Technology. He is also the Coordinator of the EC-funded EPiGRAM project and is actively involved in major e-infrastructure projects (EGI, PRACE, EUDAT) as well as exascale computing projects. His research interests include programming environments, languages, compilers and runtime systems for parallel and distributed computing, with a focus on exascale computing.

Alistair Hart is an Application Specialist in the Cray Europe Applications Team and part of Cray’s Exascale Research Initiative in Europe. He has specific interests in large-scale HPC and using it to solve scientific problems. He has an academic background in HPC and lattice QCD and experience in participating in and managing EU-funded projects in both the commercial sectors and in academia. He has a particular interest in the application of novel architectures in HPC, including the GPU-accelerated Cray XK7.

David Henty graduated with a degree in physics from Imperial College London, UK, in 1987, and gained his PhD in theoretical physics from Glasgow University, UK, in 1990. He spent the next four and a half years doing research in lattice field theory at Edinburgh University, UK, before joining the Edinburgh Parallel Computing Centre (EPCC) in 1995. David became Project Manager of the EPCC’s Academic Research, Training and Support (ARTS) group in 1999. He directly manages a wide range of projects which include: the University of Edinburgh HPC Service; EPCC’s MSc in HPC; and the UKHEC project, a three-year HPC research collaboration with Daresbury Laboratory and Manchester University.

Katherine Heisey is the Lead Developer of Nek5000/Nekbone for the CESAR codesign project. She has performed runtime studies on Argonne’s Blue Gene/Q computer and has developed new support tools and methods for Nek5000 that will be able to function in the space of up to 2 billion elements. While still an undergraduate, Heisey worked in the Mathematics and Computer Science Division at Argonne National Laboratory, USA, as a summer student in 2009. After graduating with a BSc from the University of Illinois in Chicago, USA, she joined the Argonne National Laboratory as a Scientific Assistant.

Paul Fischer holds a Blue Waters Professorship at the University of Illinois, USA with the Faculty of Computer Science and Mechanical Science & Engineering. He is Deputy Director (part-time) of the DOE Center for Exascale Simulation of Advanced Reactors at the Argonne National Laboratory. He is Chief Architect of Nek5000, which is an open-source fluid simulation code that has scaled beyond a million-ranks and has been recognized with the Gordon Bell Prize in HPC.