GPU Accelerated Computation of the ICON Model

Christian Conti, William Sawyer
Swiss National Supercomputing Center (CSCS)
Manno, Switzerland

April-May 2011

Contents

---

1 Introduction

The main objective of this work is to explore the capacity of modern GPUs to accelerate the ICON (ICOsahedral Non-hydrostatic) model [4] developed by the Max-Planck-Institut für Meteorologie (MPI-M) in Hamburg in collaboration with the Deutscher Wetterdienst (DWD). The ICON model is an atmospheric general circulation model suited for both global and regional scale simulation.

To study the potential for a GPU accelerated implementation, different approaches such as C++ with OpenCL [5], CUDA Fortran [9] and accelerator directives will be used.

The work is divided in two phases: a preliminary part where a simple case of advection in the Icosahedral Shallow Water Prototype (ICOSWP) [1] framework is studied and the main phase where the non-hydrostatic model contained in a preliminary test release of ICON is ported to GPU.

Details for the first task can be found in [8]. The rest of this document will focus on the second part and is structured as follows: Section 2 will discuss the relevant feature of the model, Section 3 will present a method for performance assessment and the hardware settings used, Section 4 will show the main results of this preliminary study and Section 5 will conclude with some suggestions to advance in this study.

2 Icosahedral Non-Hydrostatic Model

This section will discuss the main features of the ICON model that are relevant for a GPU implementation. This short overview is neither meant to be exhaustive nor complete as the Mathematics behind the model are not the focus of this document.

2.1 Problem Statement

The ICON model aims at solving the fully compressible non-hydrostatic Euler equations arising from the Energy-Vorticity Theory (EVT) [6] and derived from the conservation laws (conservation of mass, momentum and energy).

This subsection will present the base model equations as shown in [2] as a reference. It will not however go into the details and transformations that

deliver the final equations as this requires specific mathematical knowledge that goes beyond the objective of this project.

The set of conservation laws is composed by the conservation of mass,

$$\frac{\partial p'}{\partial t} - \rho_0 g w + \gamma p \nabla \cdot \mathbf{v} = -\mathbf{v} \cdot \nabla p' + \frac{\gamma p}{T}\left(\frac{\dot{Q}}{c_p} + \frac{T_0}{\theta_0} D_\theta\right), \tag{1}$$

the conservation of momentum,

$$\frac{\partial u}{\partial t} + \frac{1}{\rho}\left(\frac{\partial p'}{\partial x} - \frac{\sigma}{p^*}\frac{\partial p^*}{\partial x}\frac{\partial p'}{\partial \sigma}\right) = -\mathbf{v} \cdot \nabla u + f v + D_u, \tag{2}$$

$$\frac{\partial v}{\partial t} + \frac{1}{\rho}\left(\frac{\partial p'}{\partial y} - \frac{\sigma}{p^*}\frac{\partial p^*}{\partial y}\frac{\partial p'}{\partial \sigma}\right) = -\mathbf{v} \cdot \nabla v + f u + D_v, \tag{3}$$

$$\frac{\partial w}{\partial t} - \frac{\rho_0}{\rho}\frac{g}{p^*}\frac{\partial p'}{\partial \sigma} + \frac{g}{\gamma}\frac{p'}{p} = -\mathbf{v} \cdot \nabla w + g\frac{p_0}{p}\frac{T'}{T_0} - \frac{g R_d}{c_p}\frac{p'}{p} + D_w, \tag{4}$$

and the conservation of energy,

$$\frac{\partial T}{\partial t} = -\mathbf{v} \cdot \nabla T + \frac{1}{\rho c_p}\left(\frac{\partial p'}{\partial t} + \mathbf{v} \cdot \nabla p' - \rho_0 g w\right) + \frac{\dot{Q}}{c_p} + \frac{T_0}{\theta_0} D_\theta. \tag{5}$$

The equations above are presented in a (x, y, σ) coordinate system and also include terms found from the physical parametrization of sub-grid phenomena. A subscript $_0$ indicates the variable in its reference state and a prime denotes its deviation from it. ρ, θ, $\dot{Q}$ and D_θ represent density, potential temperature, heating rate due to diabatic processes and subgrid-scale eddy terms respectively whereas the constants g, f, R_d, c_p and γ are the acceleration due to gravity, the Coriolis term, the gas constant for dry air, the heat capacity at constant pressure of air and the heat capacity ratio.

2.2 Main Characteristics of the ICON Model

As the name suggests, the ICON model grid is built on top of an icosahedral grid, composed of triangular cells or, in its dual representation, of hexagonal and pentagonal cells (shown in Figure 1).

The data necessary to compute a solution to the equations is organized in a C-type staggering meaning that the data lives on three different elements depending on its type: vertices, edges and faces (see Figure 2).

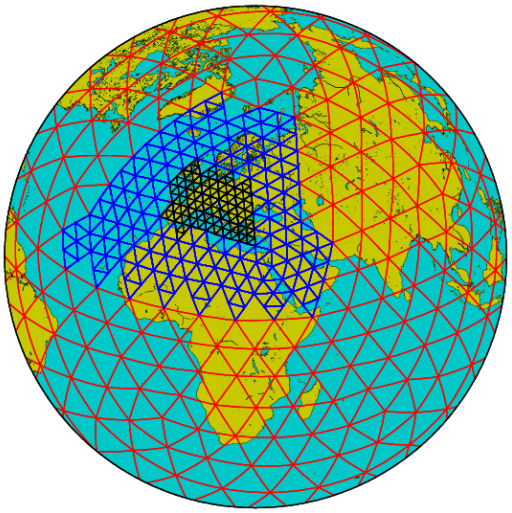

Figure 1: Dual representation of the icosahedral grid used in the ICON model as shown on the ICON webpage [4]. The picture shows a nested grid with multiple levels of resolution.

A vertical dimension is added by extruding the cells in normal direction so that each cell of the triangular grid can be approximated by a prism that lives on a certain level (or altitude). In this settings, the data is placed on the vertices, on the edge midpoints, on the face centers and in the center of the cell volumes.

The icosahedral grid allows for multi-resolution computation so that a regional scale simulation can be performed within the same model without needing an additional global model to find the boundary condition.

A uniform resolution ICON grid is called "$RxBy$" where x denotes the number of divisions of the icosahedron edges and y is the number of following bisecting operations made to refine the grid.

Because of this non-trivial representation of the data, the implementation of the solver is going to experience some performance degradation due to the memory indirections required to access the neighboring cells. The underlying memory layout is also determinant for reaching a good performance.

A second important factor that has a large impact on the performance is the high number of memory accesses per floating point operation and the low potential for caching of the data as most of the data read is used only once. This causes the implementation to be highly limited by the bandwidth

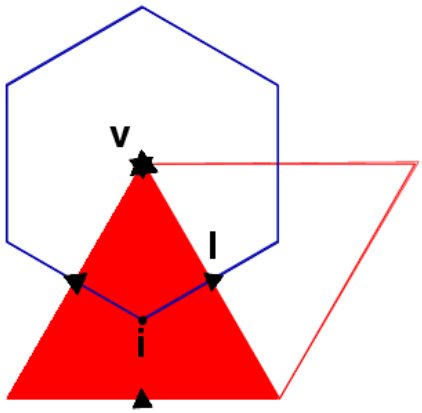

Figure 2: C-staggering of data for the ICON model as shown on the ICON webpage [4] with triangle vertices (v), edges (l) and faces (i).

of the underlying hardware architecture.

3 Performance Assessment

In order to be able to compare the performance of the various hardware settings considered in this work and to assess the efficiency of the kernels implemented for the non-hydrostatic solver, the roofline model is used [10]. This model is useful for optimization as it allows the estimation of the maximum performance in GFLOP/s expected from a kernel even when the computation is memory bound.

3.1 Roofline Model

The roofline model consists in a log-log plot of the operational intensity of the kernel measured in FLOP/B (horizontal axis) versus maximum floating point performance measured in GFLOP/s. For a given kernel, the operational intensity is defined as the ratio of the number of floating point instructions per byte of memory traffic.

The plot is divided in two distinct regions: a left line that defines all operational intensities for which the performance is bound by the memory bandwidth and a right line that denotes values of the operational intensity

with performance limited by the computing capacity of the underlying hardware. Note that the slope of the left region represents the bandwidth.

By estimating the operational intensity of a kernel, the maximum expected performance can be easily found and therefore a better upper bound for the performance can be used to evaluate how much space is left for optimization and therefore help decide how to prioritize the optimizations (together with the timing information).

3.2 Hardware Configurations

For this work, four hardware configurations have been used: a 24 cores AMD Magny-Cours 6174 with an Nvidia Tesla M2050 (based on the Fermi architecture), a 12 cores AMD Istanbul 2427 with an Nvidia Tesla S1070 shared with another identical node (two Tesla T10 GPUs per node), a 12 cores AMD Istanbul 2427 with an Nvidia GeForce GTX285 and a 12 cores AMD Istanbul 2427 with an AMD Radeon 6990 (two Cayman GPUs). The rooflines for the four GPUs are illustrated in Figure 3 together with the Magny-Cours.

Each hardware settings is represented by two rooflines: one for the theoretical peak performance, i.e. the maximum double precision floating point performance and maximum bandwidth computed from the hardware specifications, and one for the measured peak performance where a set of benchmarks (an SSE version of the STREAM benchmark [3], the oclBandwithTest from the Nvidia OpenCL SDK and a high operational intensity benchmark created for this purpose) is used to find the effective peak performance of the machine.

Figure 4 additionally shows the behavior of the device to device bandwidth obtained with `./oclBandwidthTest -dtod -mode=shmoo` of the four GPUs with respect to the size of data that is copied. It should be mentioned that using a 2D or 3D grid of threads further reduces the measured bandwidths considerably and thus, whenever possible, a 1D global work item indexing should be used. These factors decrease the effective bandwidth and should be therefore considered together with the roofline.

All four GPUs show a similar general degradation of the bandwidth with the decrease of the data size. However, the M2050 and the GTX285 shows some anomalous behavior for sizes between 200 and 500 KB. Moreover, even though the M2050 has a higher reachable bandwidth than the T10, for sizes between 500 KB and a few MB it displays an opposite behavior.

The Cayman GPU shows a lower bandwidth for sizes smaller than 1 MB

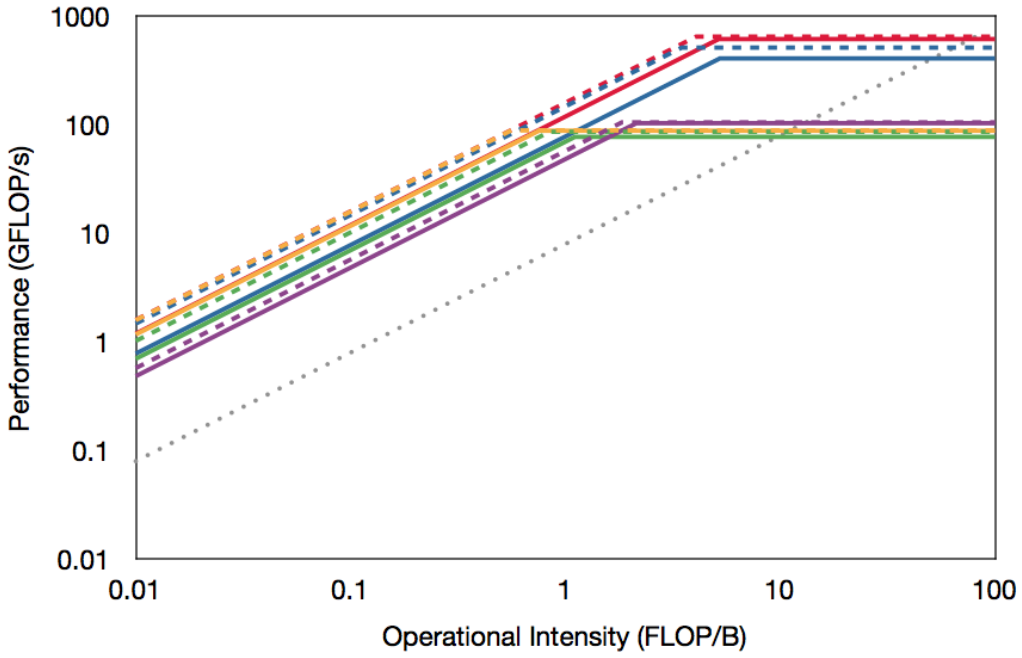

Figure 3: Double precision rooflines for Magny-Cours (purple), Tesla M2050 (blue), Tesla T10 (green), GeForce GTX285 (yellow) and Cayman (red). The theoretical rooflines are represented with dashed lines and the measured ones are shown with solid lines. The grey dotted line represents the theoretical PCIe bandwidth.

but quickly reaches the values measured for the GTX285 after that, although with some fluctuations.

4 Results

This section presents the major results obtained by the preliminary study and should help understanding the relation between GPUs and the solver. Subsection 4.1 is dedicated to define the role of a GPU for the non-hydrostatic solver, Subsection 4.2 analyzes the overall results obtained and uses them to compare the GPU with the CPU. Finally Subsection 4.3 takes a more in-depth look at the kernels that form the solver.

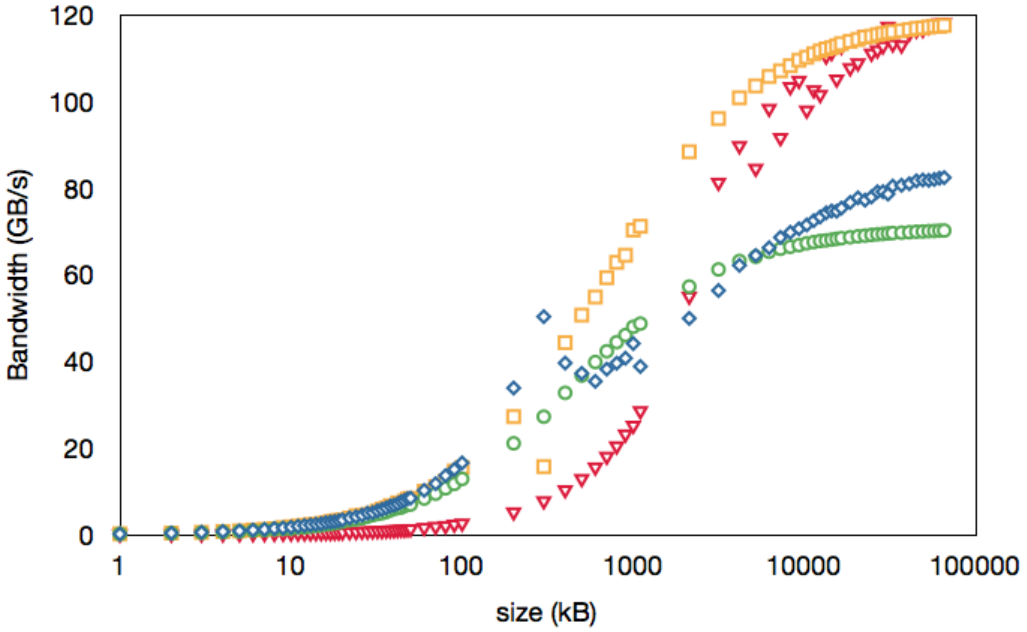

Figure 4: Variation of the device to device bandwidth of the Tesla M2050 (blue diamonds), the Tesla T10 (green circles), GeForce GTX285 (yellow squares) and Cayman (red triangles) with respect to the size of the working data (1kB up to 64MB).

4.1 Role of the GPU

From a purely theoretical point of view, since the solver performance is highly limited by the bandwidth, computation is going to be probably faster on the CPU than a GPU treated as a simple accelerator (and thus requiring frequent exchange of data between CPU and GPU), as the PCIe bandwidth between the CPU and the GPU is much lower than the internal bandwidth of the CPU. Theoretically the PCIe bandwidth delivers up to 8 GB/s but in practice the bandwidth reaches 6 GB/s only in the best cases and 2-5 GB/s in other cases.

If, on the other hand, the data can live mostly on the GPU and therefore the GPU is not just considered as an accelerator but as the main computing device, its higher internal bandwidth is going to make an important difference (see Figure 3).

The results presented in this section were obtained on a R2B3 horizontal

grid (7680 edges, 5120 triangular cells and 2562 vertices per level) and on a R2B4 (30720 edges, 20480 triangular cells and 10242 vertices per level), both with 35 vertical levels and 36 half levels.

Figure 5 shows how time for the Tesla M2050 is divided among the various GPU tasks: transfers to the GPU (H2D), transfers from the GPU (D2H), Fortran and C++ overhead and actual computation. All three tested NVIDIA GPUs show similar results in terms of how the time is partitioned, whereas the AMD Cayman shows much lower time spent on D2H tasks.

It is important to know that while on the left side the proportions will mostly remain stable with respect to the number of iterations, in the ones on the left D2H and H2D overheads will shrink asymptotically to zero. In the latter case it is therefore only of secondary importance to optimize communication through PCIe. The rest of the results presented will consider the second case (GPU as main computational device), as the first case (GPU as accelerator) is not as appealing in comparison to a CPU only computation.

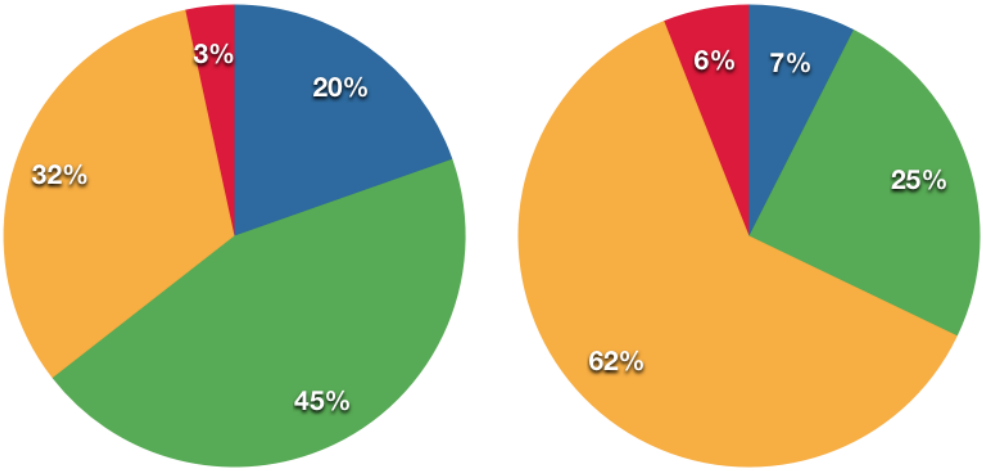

Figure 5: Time distribution over 1000 iterations for host to device (green), device to host (blue), computation (yellow) and C++/Fortran overhead (red) for a Tesla M2050 used as accelerator (left) and as main device (right) for the R2B3 grid.

4.2 GPUs vs CPUs

Tables 1, 2 and Figure 6 show the timing results for 1000 iterations of the solver with the various configurations. The results for the GPUs only consider the kernel execution neglecting CPU-GPU communication, as the improvements would otherwise depend on the number of iterations. The CPU solver provided in the original code is parallelized with OpenMP.

The tables present the timings for the four GPUs considered and three different settings for the CPU run: single thread, twelve-threaded and 24-threaded executions. The first two values for the CPU (1T and 12T) together with the 24T should help understand how the program performance scales with the number of threads, whereas, for a fair comparison with GPUs, only the 24-threaded run should be considered, as it represents the use to full capacity of the CPU node.

	Time (s)	Perf. Variation (w.r.t. 1T)	Perf. Variation (w.r.t. 24T)
Magny-Cours 6174 - 1T	466	1.0	0.1
Magny-Cours 6174 - 12T	54	8.6	0.8
Magny-Cours 6174 - 24T	45	10.3	1.0
Tesla M2050	38	12.1	1.2
Tesla T10	53	8.7	0.8
GeForce GTX285	37	12.5	1.2
Cayman	30	15.7	1.5

Table 1: Time in seconds for 1000 iterations of the solver on CPU, single threaded and 24-threaded, and on the various GPUs (kernel time only) for the R2B3 grid.

At low resolutions the GPUs deliver performance comparable to the CPU, even though the GPUs possess higher bandwidth and, in the case of the M2050 and the Cayman, higher peak double precision floating point performance (see Figure 3). The cause of this phenomenon is the relatively small amount of work that the devices perform.

Evidence of this problem is observed in Table 3 where the ratio between the time measures for grids R2B3 and R2B4 is exposed. The R2B4 grid is four times as large as the R2B3 and thus a ratio smaller than four means that in the smaller grid peak performance is not reached. This seems to be true

	Time (s)	Perf. Variation (w.r.t. 1T)	Perf. Variation (w.r.t. 24T)
Magny-Cours 6174 - 1T	1954	1.0	0.1
Magny-Cours 6174 - 12T	231	8.4	0.8
Magny-Cours 6174 - 24T	179	10.9	1.0
Tesla M2050	93	21.1	1.9
Tesla T10	171	11.4	1.0
GeForce GTX285	118	16.6	1.5
Cayman	88	22.2	2.0

Table 2: Time in seconds for 1000 iterations of the solver on CPU, single threaded and 24-threaded, and on the various GPUs (kernel time only) for the R2B4 grid.

for all the GPUs whereas the CPU scales almost perfectly with the problem size.

	Time (s) R2B3	Time (s) R2B4	Time Ratio
Magny-Cours 6174 - 1T	466	1954	4.2
Magny-Cours 6174 - 12T	54	231	4.3
Magny-Cours 6174 - 24T	45	179	4.0
Tesla M2050	38	93	2.4
Tesla T10	53	171	3.2
GeForce GTX285	37	117	3.2
Cayman	30	88	2.9

Table 3: Timing for the kernels on the various GPUs for the R2B3 and R2B4 grids and ratio of the times (the second grid is four time as large as the first).

For the Nvidia GPUs it should be noted that, even though the Tesla M2050 has a smaller bandwidth than the Tesla T10 for the sizes of the structures in the considered grids (between 700kB and 1MB for R2B3), the former delivers higher performance. This fact is explained by the completely different performance of the local memories (called "shared memories" in CUDA): using local memory for double precision values on the older devices brings only little increase in performance.

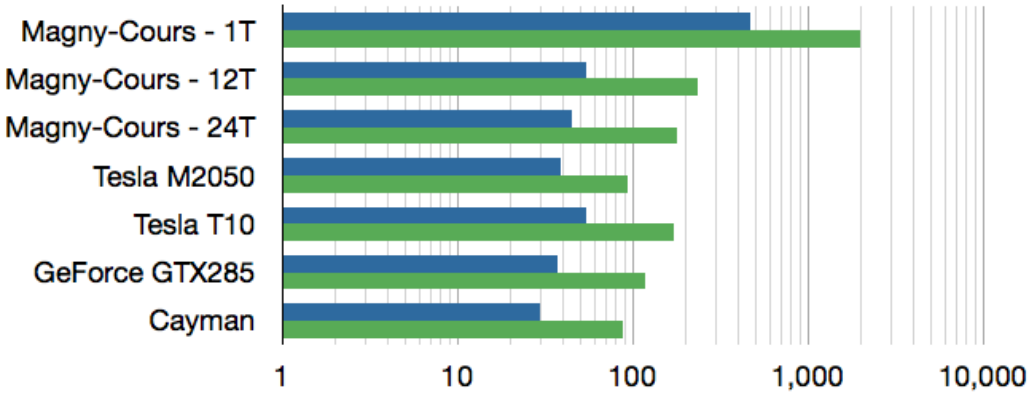

Figure 6: Log histogram showing the measured times from for the R2B3 (green) and the R2B4 (blue) grids.

4.3 OpenCL Kernels

The first step necessary to port the solver from Fortran to OpenCL is to identify what will become a kernel. In general each set of loops on the data structures can be mapped to a kernel where each independent iteration corresponds to the task of a single work item. Figure 7 shows how a set of Fortran `for` loops is mapped to a three dimensional array of OpenCL work items (threads). The calls to `get_indices_c` to get the indices `i_startidx` and `i_endidx` are prepared outside of the OpenCL kernel and stored into integer arrays `localStart` and `localEnd` respectively.

For kernels that use any member of the input data more than once across all the threads, local memory can be introduced to store and share the common data among the work items in a work group. In the solver, local memory is used mainly to share data among work items processing the same vertical column in space. Sharing across the horizontal plane is more complicated as the amount of shared memory is limited and cannot contain the entire plane. It could, however, store some local neighborhood and thus be used as a sort of cache. Using local memory this way requires a deep understanding of the underlying data structures in order to choose the correct data to store so that the cache hits can be maximized.

Figure 8 shows the time distribution between the various kernels of the solver on the M2050 and on the Cayman. Although the bottlenecks are basically the same, there are some important differences in behavior probably

Fortran

```
DO jb = i_startblk, i_endblk

  CALL get_indices_c(p_patch, jb, i_startblk, i_endblk, &
                     i_startidx, i_endidx, rl_start, rl_end)

  DO jk = 1, nlev
    DO jc = i_startidx, i_endidx
```

OpenCL

```
const int jb = i_startblk + get_global_id(0);
const int jc = localStart[get_global_id(0)] + get_global_id(2);
const int jk = get_global_id(1);

if (jk < nlev && jb < i_endblk && jc < localEnd[get_global_id(0)])
{
    const int idx = jc + jk*nproma + jb*nproma*nlev;
```

Figure 7: Mapping of Fortran loops to OpenCL work items.

due to the radically different hardware architectures. The kernel with the major visual difference (called "endphase7") is one of the two kernels (the other being "endphase4") that contain at least one `for` loop that could not be parallelized due to some dependencies between the iterations.

Figure 9 shows a log-log plot with the operational intensities of the various kernels implemented for the non-hydrostatic solver against the expected optimal and measured performance in GFLOP/s for the Tesla M2050 and the Cayman. It has to be considered that the bandwidth is strongly influenced by the size of the working data as shown in Figure 4. In the case of most of the kernels in the ICON solver, the maximum is expected to be around 20-30% less than the highest bandwidth measured by the OpenCL benchmark whereas for some of the smaller ones it is up to 90-95% less.

It is important to remember that the values for the operational intensities are only estimates, as the code that effectively runs on the device cannot be analyzed and some optimizations might change them.

It should be pointed out that, even if a kernel reaches 100% efficiency, it does not mean that it cannot be optimized anymore. A kernel that is in a bandwidth limited region can improve its performance by increasing the operational intensity (for example through the use of caches).

For validation purposes, each kernel input and output is checked against

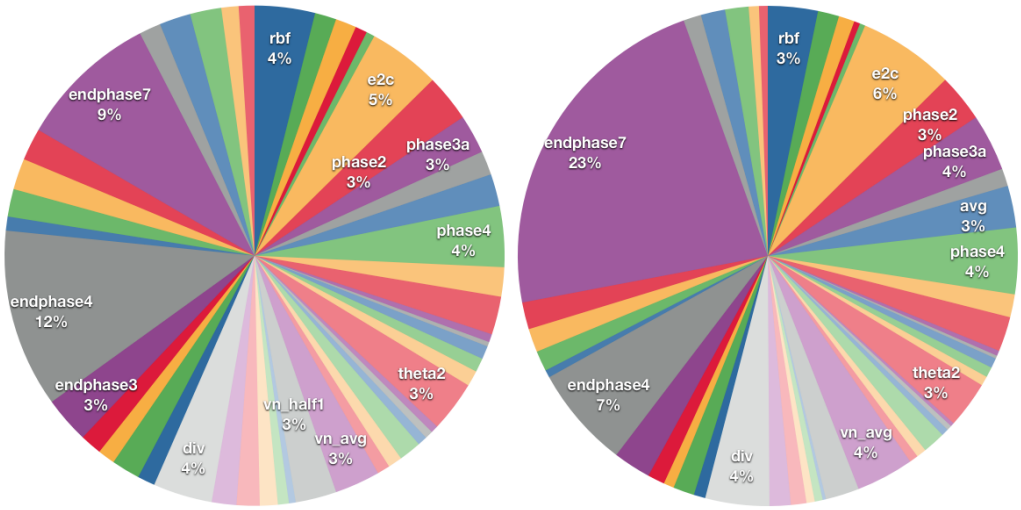

Figure 8: Partitioning of the time spent for kernel computations showing the most significant kernels for the R2B4 grid on the Nvidia M2050 (left) and on the AMD Cayman (right).

the original Fortran code so that it should be guaranteed that the OpenCL kernels perform the same tasks as their Fortran counterparts.

Overall, the GPU solver requires around 0.47 GFLOPs on the R2B3 and 1.88 GFLOPs on the R2B4 grid. The aggregated estimated performance for the kernels is shown in Table 4.

GPU	Grid	Performance (GFLOP/s)	Efficiency (%)
Tesla M2050	R2B3	12.4	36
Tesla M2050	R2B4	20.2	59
Cayman	R2B4	21.2	41

Table 4: Overall performance and efficiency of the solver

5 Future

The current implementation should be tried on a R2B5 or larger grid to harness the maximum capacity of the GPUs and thus to have the maximum

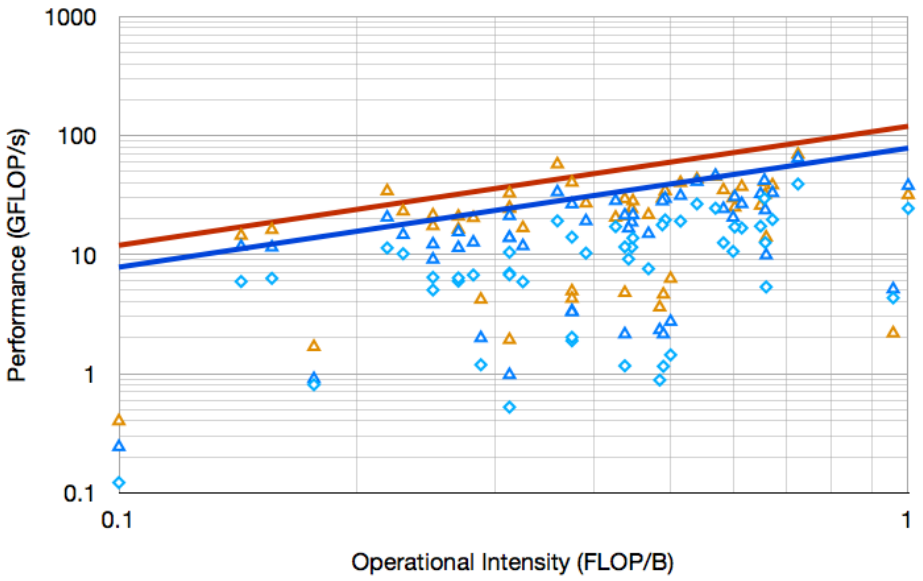

Figure 9: Operational intensities of the various kernels implemented with expected optimal (solid lines) and measured performance with an R2B3 (diamonds) and with an R2B4 grid (triangles) on a Tesla M2050 (blue shades) and on a Cayman (red/orange shades). The expected performance is based on the operational intensity only and does not consider the performance degradation caused by the size of the data structures on which the kernels operate.

improvement. However, a much larger grid (like an R2B7 and maybe even an R2B6, depending on the GPU memory size) might not fit on the GPU and thus additional transfers between CPUs and GPUs are required.

For an implementation of the model on multiple GPUs, possibly across multiple nodes, additional considerations are needed to minimize data transfer between devices as this might grow to be the main bottleneck. The first of these is the use of pinned memory that should be able to deliver a much higher bandwidth over PCIe.

In terms of optimizations, one objective is the increase of the operational intensity of the kernels by use of caches (shared memory and caches) and registers whenever possible. Another target to consider would be the coalescing of the memory accesses. The memory bandwidth can also be increased with the use of one-dimensional work item spaces in place of multidimensional in-

dex spaces which trade ease of representation with performance (when buffers are used). An obstacle towards this goal is represented by "`nproma`", a variable that helps optimizing the code for CPU by enabling the use of SSE instructions, loop unrolling and optimized caching. On GPU however, since each point in the iteration space is mapped to a work item, the types of optimization mentioned above do not apply and this parameter only represents an obstacle that produces overhead (in the form of everything related to the integer arrays `localStart` and `localEnd`) and requires an important quantity of code as well as a third dimension in the work item index space.

Furthermore, some GPU specific optimizations could be applied as Nvidia and AMD GPUs have completely different architectures.

Up to the moment when this report was written, no working API for OpenCL 1.1 was available from Nvidia and thus some of the new features in the latest version of the OpenCL specification have not been tried. However, it might be useful to try some new methods that could directly benefit the GPU implementation (such as `clEnqueueCopyBufferRect`). These methods could help in selecting the regions on which computation has to be done so that coalesced accesses are done more easily and a simple 1D addressing mode of the threads can be used, thus increasing the bandwidth.

Additionally, an OpenCL CPU code could be implemented and compared against the original Fortran program parallelized with OpenMP and against the GPU implementations. The OpenCL GPU program will most likely not be efficient on CPU as the two architectures require different types of optimizations and thus a new implementation will be required.

Finally, unless the algorithm of the ICON solver can be reformulated, high memory bandwidth devices should be preferred over architectures with high computational capacity.

The accuracy of the computation on GPU should also be considered and tested as generally there are some discrepancies in the results between a computation on a CPU and one on a GPU [7].

References

[1] L. Bonaventura, L. Kornblueh, M. Giorgetta, E. Roeckner, T. Heinze, D. Majewski, P. Ripodas, B. Ritter, T. Tingler, and J. Baudisch. The ICON shallow water model: scientific documentation and benchmark tests. http://icon.enes.org/references/publications/icon_scidoc002.pdf, 2004.

[2] J. Dudhia. A nonhydrostatic version of the Penn State-NCAR Mesoscale Model: Validation tests and simulation of an Atlantic cyclone and cold front. In Monterey Workshop, 1993.

[3] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pages 19–25, Dec. 1995.

[4] MPI-M and DWD. ICON. www.icon.enes.org.

[5] A. Munshi. The OpenCL Specification, Version 1.1. Khronos Group Std., September 2010.

[6] P. Nevir and R. Blender. A nambu representation of incompressible hydrodynamics using helicity and enstrophy. Journal of Physics A: Mathematical and General, 26(22):L1189, 1993.

[7] D. Rossinelli, B. Hejazialhosseini, D. G. Spampinato, and P. Koumoutsakos. Multicore/multi-gpu accelerated simulations of multiphase compressible flows using wavelet adapted grids. SIAM Journal on Scientific Computing, 33(2):512–540, 2011.

[8] M. Schraner, C. Conti, and W. Sawyer. Gpu parallelization of the icon shallow water model. Technical report, CSCS, 2011.

[9] The Portland Group. CUDA Fortran. www.pgroup.com.

[10] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM, 52(4):65–76, 2009.