MONC eCSE05-12

Technical report: Enabling large-scale microphysics and optimising solver performance in MONC

Nick Brown, Adrian Hill, Ben Shipway, Michele Weiland

April 27, 2017

Abstract

MONC is a highly scalable modelling tool for the investigation of atmospheric flows, turbulence and cloud microphysics. In this report we detail the work done during eCSE05-12, firstly we coupled PETSc with MONC for optimisation of the iterative pressure solver and contrasted this against the direct FFT solver. It can be clearly seen that the FFT solver scales far worse than the iterative solver and the iterative solver now out performs the FFT solver for coarse solves. Secondly the CASIM micro-physics scheme was optimised, which has not only more than doubled the performance of this sub-model on CPUs but we have also ported the scheme to GPUs and KNL to understand its applicability to these architectures, enable the scheme to take advantage of new HPC machines installations (such as ARCHER KNL) and future directions for hardware. Thirdly significant improvements, bug fixes and optimisations have been developed for the in-situ data analytics which is relied upon by users to produce higher level, diagnostic, information from the raw prognostic data. The result of this work are a number of significant improvements to the model which not only provide significant new functionality to current users on ARCHER, but also future proof and identify avenues for further development.

1 Introduction

The Met Office NERC Cloud model (MONC) [1] is an open source high resolution modelling framework that employs large eddy simulation to study the physics of turbulent flows and further develop and test physical parametrisations and assumptions used in numerical weather and climate prediction. MONC replaces an existing model called the Large Eddy Model (LEM) [4] which was an instrumental tool, used by the weather and climate communities, since the 1980s for activities such as development and testing of the Met Office Unified Model (UM) boundary layer scheme [9][10], convection scheme [12][11] and cloud microphysics [13][8].

MONC has been developed in Fortran 2003 and uses MPI for parallelism. The standard version of this code is designed to be run on many thousands of cores and has demonstrated good performance and scalability on up to 32768 cores [1]. This model has been designed around pluggable components where the majority of the code complexity, including all of the science and parallelisation, are contained within these independent units. Components are managed by a registry and at run-time the user selects, via a configuration file, which components to enable or disable. The majority of a component's functionality is contained within optional callback procedures, which are called by the model at three stages: upon initialization, for each timestep and upon model completion. There are no global variables in MONC, but instead a user derived type is used to represent the current state of the model and this is passed into each callback which itself may modify the model state. Using this approach means that the model's current state is represented in a structured manner and there

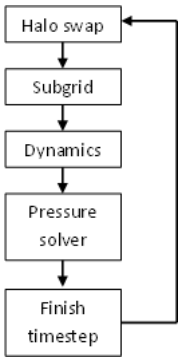

Figure 1: MONC timestep structure

is a single point of truth about the model's status at any point in time. Isolating these components significantly limits interaction between them and as such results in an easy to understand architecture where aspects can easily be added, removed or replaced without having to worry about unintended side effects. Via the component based architecture it is trivial to add significantly distinct functionality without it impacting on other parts of the model. Each component can be given bespoke compiler commands and directives during build time. Once components are compiled into the code then they are configured via a configuration file which specifies aspects such as whether they are enabled, when they should run and other component specific options.

Like many LES models the simulation proceeds in timesteps, gradually increasing the simulation time on each iteration until it reaches a predefined termination time. The model works on prognostic fields, u, v and w for wind in the x, y and z dimensions, θ for the temperature and any number of q fields which represent aspects such as moisture and tracers. Figure 1 illustrates high level groups, each of which contains any number of components, that makes up the structure of a single timestep. Each of these groups must execute sequentially so that one group can not start until the previous group has completed. Initially all prognostic fields are halo swapped between neighbouring processes and then the sub-grid group of components are called to determine model parameterisations. The dynamics group of components, often referred to as the dynamical core, performs Computational Fluid Dynamics (CFD) in order to solve modified Navier-Stokes equations [6] which is followed by the pressure solver, using either an FFT or iterative approach to solve the Poisson equation. The pressure solver is the topic of section two which focuses on optimisations made to the iterative solver in order to improve performance and future proof the model on large core counts.

In section three we consider work done during the first work package of this project in optimising the performance of the Cloud AeroSol Interactions Microphysics (CASIM) micro-physics scheme which calculates interactions between

moisture droplets in the atmosphere. Modelled at the millimetre scale, these droplets represent moisture in many different states such as vapour, liquid water, snow, ice and graupel. The modelling of moisture is such a crucial aspect of weather and climate codes that this specific scheme is designed to be used with MONC (specifically as a component of the dynamical core), the Unified Model, and Kinematic Driver (KiD) model.

The large prognostic, raw data, fields are often not of primary concern to the scientists but instead *diagnostic* values, which are values resulting from data analysis on the prognostic fields, are far more useful. One such diagnostic might be the minimum cloud height and the model proceeds in timesteps with samples taken from the prognostic fields every timestep, analysed, and then averaged over a specific time frame to produce the final diagnotic values. For instance every 5 minutes of model time we might output the average lowest cloud height over that last five minute period, this is an example of time averaging each specific contribution. In MONC data analysis therefore includes direct analysis on the prognostic fields (which produces higher level diagnostic fields), and also time manipulation of both diagnostic and prognostic fields. Section four details the improvements made to the data analytics of MONC which was found to be a critical aspect limiting the uptake of the model before we discuss some miscelaneous activities in section five and draw some brief conclusions in section six.

2 WP1: Optimise computational aspects of MONC

This work package focussed on the computationally intensive areas of the model which had been ported over directly from the existing LEM and not been optimised for large-scale parallelism. The main focus here was the iterative solver which had shown poor performance a smaller core counts but scaling potential [RE] in comparison to the FFT pressure solver. Small amounts of work was completed on other aspects, such as optimising calculation of fluxes, but these were fairly small scale.

2.1 PETSc iterative pressure solver

The component nature of MONC meant that a MONC component could be developed which would integrate with PETSc and use that framework to perform the iterative pressure solve. Upon component initialisation the system is set up with PETSc, where a 3D DMDA is created based upon the grid dimensions of MONC with periodic boundary conditions and a star stencil. The KSP is created and procedure pointers provided for generating the system RHS, initial guess and matrix. A number of other options including the type of solver, pre-conditioner and termination residual can also be specified in the user configuration file and are read from the MONC options database in a standard fashion.

Listing 1 illustrates the Fortran subroutine called by PETSc to setup the

RHS which is based upon source terms which have already been computed for the current timestep. The distributed matrix is first retrieved and then the current processes' global start grid point *xs*, *ys* and *zs* along with the local data size in each dimension *xm*, *ym* and *zm* are determined. A Fortran pointer to the underlying RHS array is retrieved and then the *copy_data_to_petsc_pointer* subroutine, which we developed, will copy the pressure source terms *p_source* (a global variable) into this array. The MONC PETSc component also stores calculated pressure values and these are used as the initial guess for the next timestep.

```
subroutine compute_RHS(ksp, b, dummy, ierr)
  PetscErrorCode :: ierr
  KSP :: ksp
  Vec :: b
  integer :: dummy(*)

  DM dm
  PetscScalar, pointer :: xx(:)
  PetscInt :: zs, ys, xs, zm, ym, xm

  call KSPGetDM(ksp, dm, ierr)
  call DMDAGetCorners(dm, zs, ys, xs, zm, ym, xm, ierr)
  call VecGetArrayF90(b, xx, ierr)
  call copy_data_to_petsc_pointer(xx, zs, ys, xs, zm, ym, xm,
      p_source)
  call VecRestoreArrayF90(b, xx, ierr)
end subroutine compute_RHS
```

Listing 1: PETSc RHS for each solve

Matrix generation is also done via a procedure callback, omitted for brevity, which is called on the first timestep only as once the matrix is initialised it remains unchanged. This matrix implements the Poisson equation in a standard manner and has been tested for correctness in comparison to the existing iterative solver and also the FFT direct solver. Listing 2 illustrates the calling of PETSc at each timestep to perform the pressure solve. The *p_source* global variable is set to be the current source terms which are held in the *p* (pressure) data array and this is then read in the *compute_RHS* procedure called from PETSc. Execution of the *KSPSolve* procedure actually performs the solve and, once this has completed, the *KSPGetSolution* procedure retrieves the solution as a PETSc vector which is then read and copied back into the *p* array, with the *pre˘_p* global variable is set to this value which will be initial conditions for the pressure solve at the next timestep.

```
p_source=current_state%p%data(z_start:z_end, y_start:y_end,
    x_start:x_end)
call KSPSolve(ksp, PETSC_NULL_OBJECT, PETSC_NULL_OBJECT, ierr)
call KSPGetSolution(ksp, x, ierr)
```

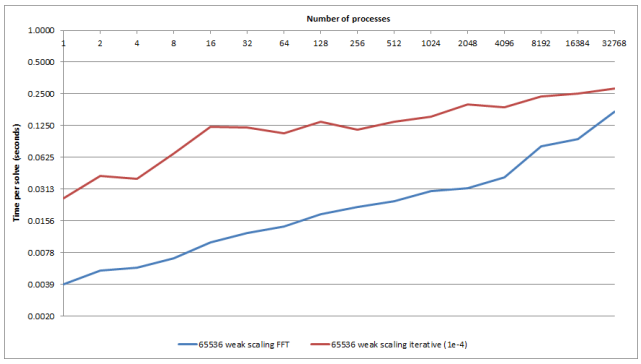

Figure 2: Performance of FFT and iterative solver as the number of processes is scaled

```
call VecGetArrayReadF90(x, xx, ierr)

call copy_petsc_pointer_to_data(xx, z_start, z_end, y_start,
    y_end, x_start, x_end, current_state%p%data)
call VecRestoreArrayReadF90(x, xx, ierr)
prev_p=current_state%p%data(z_start:z_end, y_start:y_end, x_start
    :x_end)
```

Listing 2: PETSc RHS for each solve

Figure 2 illustrates the performance of the FFT and iterative solver with a standard dry boundary layer test case on ARCHER, weak scaling with 65536 local grid points, using the BiCGStab solver with ILU preconditioner. It can be seen that the FFT solver is significantly faster than the PETSc iterative solver at smaller core counts, but as one scales up the differences diminish. A major benefit with utilising an iterative solver in contrast to a direct solver like FFTs is that we can set the termination residual and therefore explicitly choose how accurately to solve the system to. Table 1 illustrates executions of the iterative solver, solving to different relative residual accuracies, against the direct FFT solver. The *Mean difference* column illustrates the average difference per field element the pressure values calculated by the iterative solver and that by the direct solver. The focus here is on computation rather than communication, hence the timing results are over executions of the test case on one core only and the FFT solver took on average 3.78e-3 seconds per timestep. Hence it can be seen that solving iteratively to 1e-2 is significantly faster than a direct FFT solve, although there is the most significant difference in terms of solution accuracy. It is up to the user of MONC to determine the exact termination residual to solve to for their own configurations and our general suggestion is that somewhere between 1e-2 and 1e-4 is most appropriate. Of course it will depend upon the exact problem, but from experimentation it has

been observed that because we use the previous iteration's pressure values are the initial conditions of the next iteration many executions settle where only one or two iterations of the solver are needed.

Relative termination residual	Mean difference	Average time per solve (s)
1e-2	1.68e-5	1.06e-3
1e-4	9.08e-7	5.30e-3
1e-6	4.52e-9	2.71e-2
1e-8	9.91e-11	6.27e-2
FFT	N/A	3.78e-3

Table 1: Iterative solver accuracy in comparison to FFT solver and average time per solve

The performance of the existing iterative solver is similar to that of the PETSc iterative solver, however the PETSc iterative solver provides very many more options for different solvers and pre-conditioners which will suit different problems. From experimentation we found that GMRES is by far the most stable solver, however for some problems it was possible to utilise BiCGStab which afforded a performance benefit but was only stable for a subset of test-cases. Being able to experiment with different solvers and easily take advantage of new solvers and preconditioners for the future will be of significant benefit to users of MONC and help future proof it.

3 WP2: Optimising the CASIM scheme for ARCHER

The Cloud AeroSol Interactions Microphysics (CASIM) model [5] is a bulk microphysics scheme, calculating interactions between moisture droplets in the atmosphere. Modelled at the millimetre scale, these droplets represent moisture in many different states such as vapour, liquid water, snow, ice and graupel. The modelling of moisture is such a crucial aspect of weather and climate codes that this specific scheme is designed to be used as a sub-model by other more general models such as the Met Office Unified Model (UM), Met Office NERC Cloud Model (MONC) [1], Large Eddy Model (LEM) [4] and Kinematic Driver (KiD) [5] model.

Figure 3 illustrates the wallclock time for each hourly run of the COPE testcase, of 500 by 500 grid points with 70 vertical levels, when the original CASIM was coupled with the UM. It can be seen that enabling CASIM resulted in a very significant impact on the overall runtime. There was a similar picture in MONC, where enabling CASIM resulted in a two to three times increase in the overall model execution time.

From profiling CASIM we determined that execution time was spent in a number of key subroutines, specifically *sedmentation*, *condensation*, *autoconversion*, *aggregation*, *accretion*, *sum_processes* and *mphys_tidy*. which focused efforts for the project. Within each of these subroutines there were an obvious

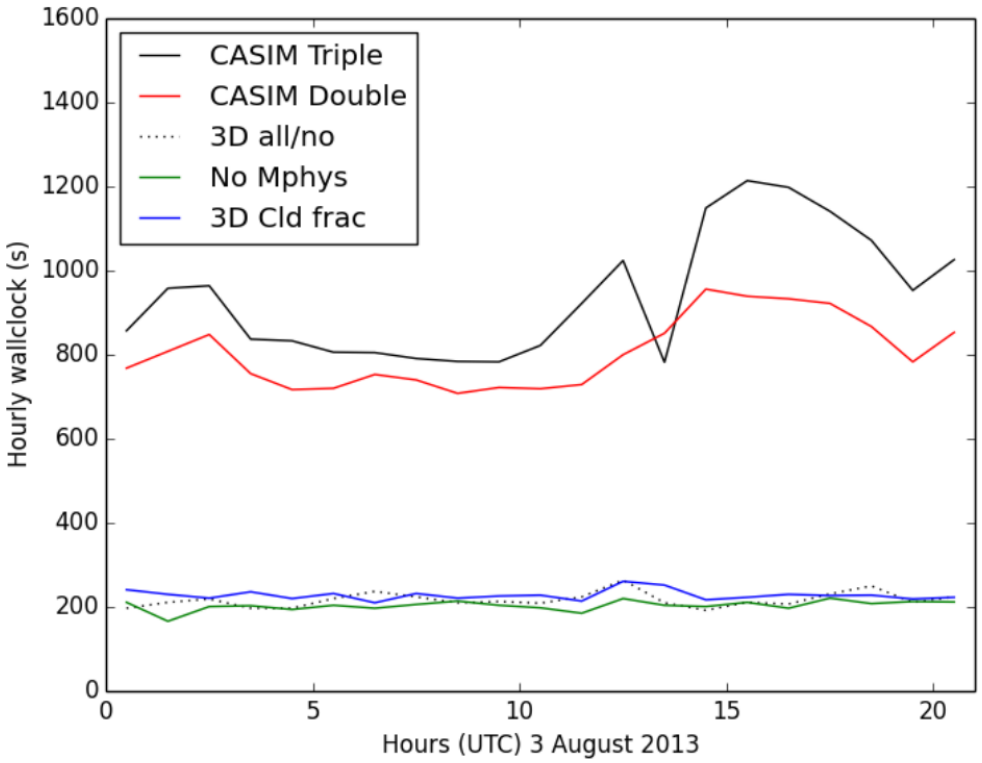

Figure 3: Wallclock time for each hourly run of the COPE UM testcase

number of targets for optimisation. Firstly, many subroutines were dynamically allocating and then deallocating arrays which was wasteful and severely impacted performance. We modified each module in CASIM that contained subroutines allocating temporary variables in this manner, to move the temporary variables as module globals and for an initialisation subroutine, called when CASIM starts up, to perform the allocation once only. It was also the case that many of these arrays were pointers, rather than allocatables. Using Fortran pointers can severely impact the performance of the code in comparison to allocatables and hence we moved all pointers to allocatables.

Numerous computational kernels involved multiple Q variables and much iteration was interwoven and done in such an order that limited the use of the cache. These routines were refactored to simplify the iteration and improve the cache reuse nature of the algorithms to further tune them. It was also found that using fast (i.e. non IEEE compliant) math mode resulted in a significant performance advantage. The cost to this is accuracy of the results and and such this has been provided to the user as a compile time option.

An OpenMP version of CASIM was also developed, which maps processing on columns to threads. The *firstprivate* clause was used for the module global variables and *copyin* to allocate allocatables. We found that by far the best performance was obtained when either the dynamic or guided loop schedules were chosen. There were two reasons for the OpenMP version of CASIM - firstly because it provides additional flexibility in running the scheme on CPUs in the future, but also it supports execution on the KNL.

Description	MONC Runtime (s)	CASIM average per timestep (s)
Original CASIM	238.01	0.050
Optimised CASIM	137.71	0.022
Optimised CASIM fast math	129.46	0.019

Table 2: MONC runtime with kinematic test case under different version of CASIM

Table 2 illustrates the performance of MONC+CASIM for a standard kinematic cold test case running for 3600 model seconds, with 18 Q fields, over one CPU core on ARCHER. It can be seen that our optimisations have significantly improved performance of the overall test case run. From concentrating on the average CASIM runtime per timestep it can be seen that we have over doubled the performance of the scheme and, if the user elects to enable fast maths which is a compiler option that has been added, then a further increase in performance is obtained.

3.1 KNL CASIM

Due to the OpenMP additions to CASIM identified in section 3, execution on the KNL was fairly trivial. The SIMD OpenMP directive was applied to computationally intensive loops in the code that work up or down the column

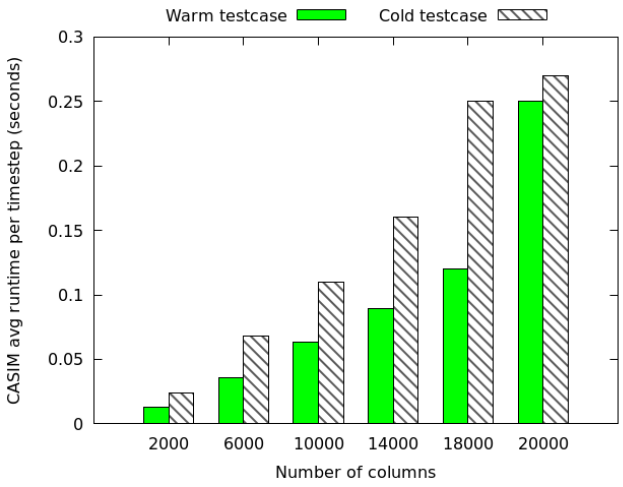

Figure 4: Average CASIM runtime per timestep on the KNL

as is suggested for the KNL in [14]. Warm and cold stratus test cases were executed on ARCHER KNL (64 core model 7210 KNL) with sixty vertical levels (z dimension) and modifying the number of columns in the x and y dimensions to change the system size.

Each ARCHER KNL node is equipped with 96GB of main memory and 16GB of on chip MCDRAM which was configured to use as a cache for accessing the main memory. Figure 4 illustrates the performance of CASIM when running both cold and warm stratus test cases on the KNL and the presented results were averaged over three runs. With the warm test case it can be seen that there is a large jump in runtime between 18000 vertical columns and 20000 vertical columns but this configuration signified another important aspect too. Namely that up to and including 18000 columns running with 256 processes, running 4 processes per core (with hyper-threading) resulted in the best performance, however from 20000 columns using 4 way hyper-threading, where each core still runs one MPI process but each process is executing four threads, gave the best performance. This same hyper-threading behaviour was also true for the cold test case and this point occurred sooner, at 12000 vertical columns.

From in-depth analysis and testing we found that the relative performance on the KNL favoured running over larger domain sizes, figure 5 illustrates a comparison between CASIM running on the 18 core Broadwell CPU (on MONSooN, the Met Office XC40) and the KNL as we scale the number of vertical columns and hence the number of grid points. Whilst the Broadwell CPU and KNL exhibit similar performance on smaller numbers of columns the break even point is 30000 vertical columns and after this point the KNL performance is obviously beating that of the Broadwell. As we reach one million columns (60 million grid points) there is a significant difference and the runtime per CASIM

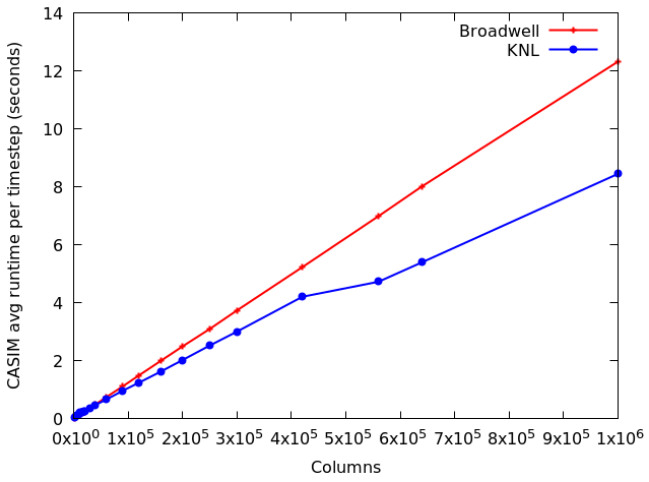

Figure 5: Average CASIM runtime per timestep cold test case comparison between Broadwell and KNL on larger column counts

timestep on the KNL is 65% of that on the Broadwell CPU. From these results we therefore believe an important message for potential users of CASIM on the KNL is that loading it up with a significant amount of computation (and hence number of grid points) is, in comparison to performance obtainable by the Broadwell, optimal.

3.2 GPU CASIM MSc dissertation

Driven by this work the author supervised an EPCC MSc in HPC student's dissertation project where they ported CASIM to GPUs via OpenACC and executed this on Piz Daint, a Cray XC50 with Haswell CPUs and P100 GPUs, the Swiss national supercomputer. The purpose of this project was two fold, firstly to understand whether GPUs can be taken advantage of with these sorts of codes and secondly to understand the applicability and maturity of OpenACC. Unlike many codes, where a kernel can be extracted and the fairly simple code decorated with OpenACC, the nature of CASIM meant that it was not possible to follow this route due to excessive data transfers that would result. Instead, the entirety of the scheme was offloaded, via OpenACC, to the GPU which included 50 Fortran modules and 123 subroutines. This approach is novel and a number of previously unknown issues were identified with the Cray OpenACC implementation which work-arounds had to be developed to handle.

Performance testing was carried using the stratus cloud test-case, where the column height (z dimension) of 60 vertical levels was fixed and the size of the domain in the x and y dimensions (number of columns) was modified. Figure 3.2 illustrates the average time taken by CASIM per timestep for different numbers

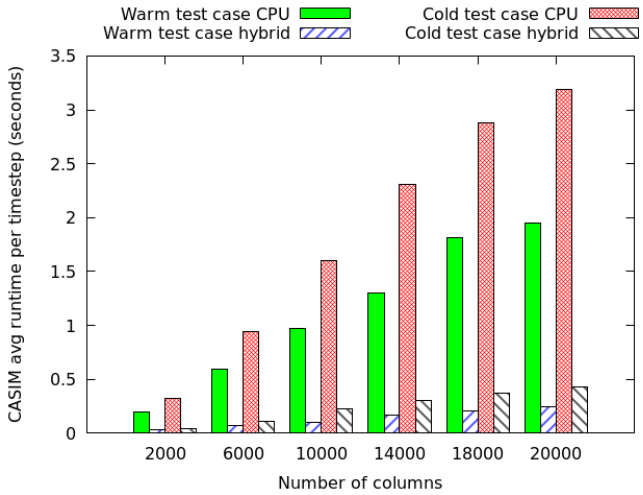

Figure 6: Average runtime per timestep of both cold and warm stratus CASIM test case on Piz Daint

of vertical columns with both the warm and cold stratus test cases on the XC50 using one CPU core. It can be seen from this graph that running CASIM on the GPU is significantly faster than running on the CPU, for instance executing the cold stratus test case over 20000 vertical columns on the GPU is 7.4 times faster than running it on one CPU core only. Even though the warm test case is less computationally intensive, a similar speed up is achieved in comparison to it also running on a single CPU core.

Whilst the GPU performance advantages of figure 3.2 look impressive, the CPU version is only running over a single Haswell core. However the Haswell processors of Piz Daint are equipped with twelve CPU cores per package and as such it is not necessarily realistic to just compare against runs involving one core only, with the rest remaining unused. Figure 7 illustrates the average CASIM runtime per timestep over 20000 vertical columns for the cold stratus test case as we modify the number of CPU cores in use, effectively decomposing the global domain over these cores each running MONC concurrently. In the case of the hybrid version of our experiments, where we run MONC on the CPU and CASIM on the GPU, each core shares the same GPU via MPS's multiplexing, effectively running multiple kernels on the GPU concurrently. From figure 7 it can be seen that the very significant speed advantage to running CASIM on a GPU in comparison to one CPU core doesn't hold as we increase the number of cores, the break even point is around 8 CPU cores and at 12 CPU cores it is actually faster to run CASIM on the CPU instead of the GPU. As CASIM is embarrassingly parallel it scales very well (there is an 11 times speed up over 12 cores in comparison to 1 core), and so we are utilising more of the resources of the CPU whilst the GPU is being utilised exactly the same, just by different

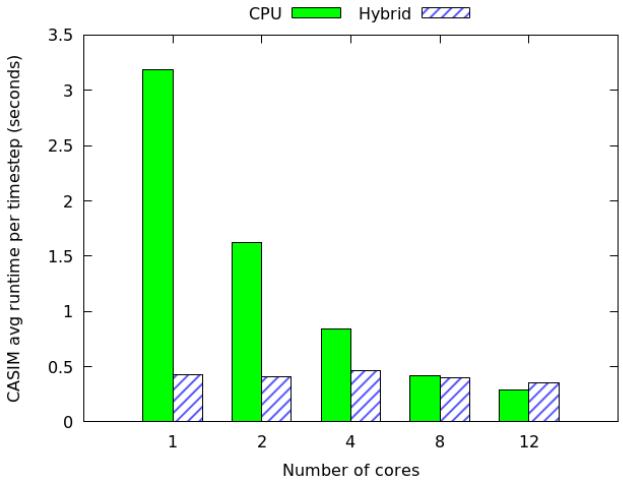

Figure 7: Average CASIM runtime per timestep for cold test case over 20000 vertical columns as the number of CPU cores is modified

CPUs. Whilst the runtime on the GPU is fairly flat, in comparison to being driven by one CPU core the average time per timestep does decrease slightly over twelve cores to 81% of the runtime driven by one core.

Whilst the results in figure 7 do argue that running CASIM over the entirety of the CPU cores, rather than the GPU, is advantageous, it should be noted that our approach to offloading is a hybrid one where the parent model will run concurrently with CASIM. A major limitation identified with this approach was that as each thread on the GPU is processing a column concurrently, all the temporary memory associated with calculations for that column must be duplicated for each thread. This is has a very significant memory impact and on the P100 the maximum number of columns that can be supported is 20000, in contrast to the KNL which can handle up to 100000 columns as the KNL not only can address more memory but also fewer threads (and hence columns) are executed concurrently which means that the temporary memory required is significantly reduced.

More information on both the GPU and KNL versions of CASIM can be found in [2].

4 WP3: General refactoring and code improvement

This work-packaged focused around optimisations to the MONC IO server which, at the start of the project, was limited and significantly impacted the usability and performance of MONC. In the approach adopted by MONC the

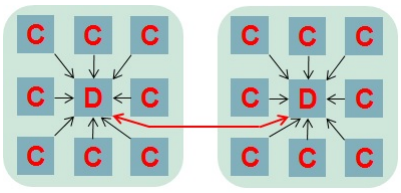

Figure 8: Computation and data analytics cores

cores within a node are shared for both computation and data analysis, as per figure 8, where typically one core per processor will perform the analysis (marked D, running the IO server) for the remaining cores which perform computation (marked C.) Much of the analytics involves combining local values to form some final result which is therefore more communication and IO bound rather than computation bound. In these situations a data analytics core (running the IO server) will compute contributions from its local MONC computational cores before communicating with all other data analytics cores to determine the final result. Determined by the user's configuration, at any timestep a group of raw prognostic fields may be communicated from a MONC computational core to its IO server.

Figure 9 illustrates the IO server and analytics pipeline, the server continuously probes for messages and when a group of fields are received from a MONC computational core (external API in the diagram) a thread is activated from the thread pool to handle this. *Forthreads* [7] is used to access the pthreads library in Fortran. Depending upon the configuration this thread will pass different fields either to the diagnostics federator for analytics or directly to the writer federator.

The second section of the user's XML configuration defines what data analytics the diagnostic federator should perform on what fields and this configuration contains a resulting diagnostic field name, meta data (such as the type and size of this field) and a number of rules to execute in order to generate this diagnostic. A specific rule will execute when its dependencies, either prognostic fields or values generated from other rules are available and it will produce resulting values which are either the final diagnostic itself or an intermediate field that drives additional rules. Rules are made up of operators (such as arithmetic, field coarsening, field slicing and filtering) or communications (such as reductions or broadcasts.) These follow a standard template (Fortran interface) and are registered with the federator when the IO server initialises to match against specific rule names in the configuration, operators for instance must implement procedures (via procedure pointers) for executing the operator and determining whether the operator can run a specific rule. In order to add an operator or communication then the user can follow this template and write their own functionality and easily plug it into the diagnostic federator. Diagnostics often results in values being located on one specific analytics core, the IO server auto

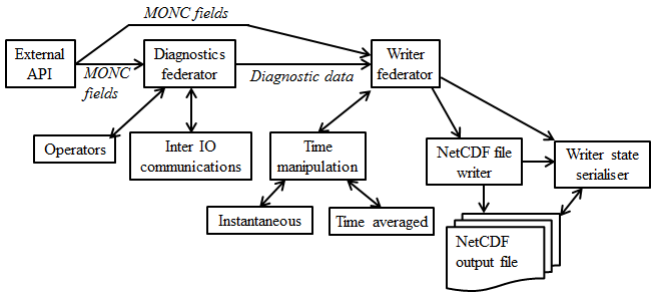

Figure 9: Data analytics architecture

load balances these to spread the values out amongst the analytics cores for optimal parallel IO.

The writer federator in figure 9, defined by the third section of the XML configuration, receives either an analysed diagnostic or raw prognostic field. Firstly time manipulation is performed and based upon an input field the time manipulation will only produce output at specific configured model times - either time averaged values which is an average of all values for that field in the period or an instantaneous value which is the value at that specific point in time. If the time manipulation produces any values then these are stored internally in the writer federator until they are physically written to disk. NetCDF file writing occurs at a specific model time period, for instance every 100 model seconds, and each field written out contains an additional time dimension, which is the result from the time manipulation at the specific output points in model time.

All groups of fields arriving at the IO server also include the model time and timestep, this is checked and when it matches the write period a file write will be initiated which defines the NetCDF file and writes out stored values. It might be that not all fields are immediately available at the time of writing, for instance some might be involved in ongoing data analysis; these are marked as outstanding and written when they are available. MONC timestepping is dynamic, where the exact size of a timestep varies depending on the computational stability of the system, hence we do not know in advance the exact number of time entries for a field as the timestep might become so large it steps over multiple time manipulation periods. Therefore each IO server must keep track of the model times as they arrive so that the collective definition of the time dimensions are correct and file writing can only occur once the model time reaches the trigger point. When all fields have been written then the file is closed and the written fields are removed from the writer federator's in-memory storage.

At the start of this work there were numerous bugs and limitations of the IO server namely bit reproducibility issues, a lack of check-pointing and significant performance issues.

4.1 Bit reproducibility

For simplicity reasons the federators and their subtasks (handlers) of figure 9 mainly process data (i.e. events) in the order in which they are received. This approach works well in many cases of the workflow, however can cause bit reproducibility issues where the handler is accumulating field contributions such as with time averaging. Therefore specific aspects of the workflow, such as averaging field values over time, must be performed in a deterministic order due to the lack of associativity of floating point arithmetic. This relates to processing data events in a deterministic order, instead of the order in which they arrived and because all data events in our approach contain meta-data specifying aspects such as the timestep and model time that the event relates to we can utilise this to determine the correct ordering. In the cases where event ordering matters the handler will order based upon this timestep. Based upon the user's configuration the handler is able to determine the timestep frequency that data will arrive for each field and hence the next expected timestep. Therefore events received out of order are stored in a queue until earlier events have been received and processed. In these event handlers, the processing of a specific event is therefore followed by checking the queue and then processing any appropriate outstanding events which can now be handled without compromising ordering constraints.

4.2 Check pointing

The MONC model can be long running, taking days or weeks of computational time, but often HPC machines have a runtime limit for jobs, such as twenty-four or fourty-eight hours. As such the model proceeds in episodes where MONC is run until a specific walltime, then writes out its state into a checkpoint file and will then restart from this checkpoint, this is known as a continuation run. For these long runs a script schedules two jobs, a job to run the model directly (either starting from initial conditions or a checkpoint file) and another job in a help state which is executed once the other job finishes. This second job will check whether a continuation run is appropriate, if so schedule another job in the held state dependent waiting for this job to complete and execute MONC with the latest checkpoint file.

For checkpointing the computational part of the model it is fairly standard because a snapshot of the model for each core contains the raw prognostic fields and other data such as the model time, timestep and size of the timestep. However it is also necessary to snapshot and restart the state of each IO server core doing data analytics and this is more complex due to the asynchronicity inherent in this approach. During this eCSE it became apparent that the inability to checkpoint the IO server was significantly limiting the sizes of jobs that the model could support. It is made more difficult by the fact that messages from MONC computational cores or between IO servers can be in flight, waiting for additional local values before communication or in the process of being issued, therefore how to handle this reliably was a challenge. We rely on two facts, firstly that the majority of this non-determinism is contained within the diag-

nostics federator and its sub actions, and secondly based upon the timestep metadata associated with each field the writer federator can determine whether it has received all the expected fields up to a specific timestep or whether there are diagnostic or prognostic fields still outstanding.

When a checkpoint is triggered the code waits until the diagnostics federator has completed all its work up until that specific timestep and made these fields available to the writer federator. At this point we only store the state of the writer federator, which is far more deterministic, and its sub activities such as the fields being manipulated in time and waiting to be written to file. The state of the writer federator can be split into five distinct areas, and checkpointing each of these is a two step process. Firstly each IO server will progress through each of the five distinct states, acquiring locks on the different states to serialise and calculating the size of memory required to hold each of these states. Next each stage is processed in tern, where memory to hold that state is explicitly allocated, the state itself is serialised into a stream of bytes which is stored in this memory area and then any locks acquired on that state are released. In the checkpoint file each of these five states is represented by a specific variable which is written into collectively by all IO servers. All IO servers will write their own state at a unique point in the variable and each variable is also associated with a directory of start points and byte lengths for each IO server rank. Once a field is written the memory associated with it is freed and the IO server will progress onto the next variable.

We split this checkpointing approach into two phases for performance reasons, because much of the state requires analysis of sub-states then the only way of determining overall length is by actually walking it. However we do not want to be packaging state into memory, then reallocating memory for sub-states and copying data between these due to the significant overhead that many memory allocations would incur. Therefore instead we adopt this approach of walking all states once to determine the local size required for each, which is then reduced between the IO servers (with a sum) and used to collectively define the appropriate variables and dimensions in the NetCDF file. Between the first stage of determining required memory size and the second stage of physically serialising the state and writing it, it is important that the state is not modified - hence locks are issued during the first state of checkpointing and only released once a state has been serialised and packaged.

Another challenge with this approach was the global size of the serialised variables that are written into the NetCDF files. With many IO servers it is realistic that the size of the field will exceed 2GB and the Fortran interface to NetCDF only supports signed (hence 2GB rather than 4GB) 32 bit integers. This limits not just the dimension sizes, but also the specification of start locations and counts for parallel writes along with the integer data type. Instead we were forced to call directly to the NETCDF C interface via the ISO C bindings for storing and reading much of the IO server state to enable unsigned 64 bit integers.

4.3 Optimising performance

Four aspects were identified as major contributors towards the performance limitations of the IO server which we concentrated upon. Firstly the IO server makes extensive use of data structure functionality, such as lists and maps, provided by the MONC model. Most significantly data is often entered into maps and keyed with some string literal, such as the field name concatenated with the timestep. However searching and retrieval from these maps was naively implemented as a linear search with O(n) time complexity. The MONC map was modified to utilise a hashing function, where a map contains a number of hash key entries, each pointing to a list of contents. When a key value pair is added the key is converted, via a hashing function, into its corresponding hash location which is used to specify where in the map the value should be stored. Because hashs are not necessarily unique each hash location contains a list of matching entries, each with their full individual key strings. This results in a trade-off between performance and memory usage and as such the hashing function in MONC will convert strings to one of 4993 hash key possibilities, but whilst there might be some linear searching still required within a number of records matching that specific hash key this approach drastically reduces the overall amount of searching required.

At the start of this project the IO server required MPI to be run in thread multiple mode and as the IO server was compiled into the MONC executable, effectively this meant that all MONC processes needed to be started in thread multiple mode even if they were (as the vast majority of cases) computational cores. Not only was this seen as a significant performane limitation, but also having to export the *MPICH_MAX_THREAD_SAFETY* variable in submission scripts confused users and was often omitted, resulting in errors. We modified the IO server such that it could be run in thread serialised mode, explicitly protecting the MPI calls internally rather than relying upon the MPI implementation for this. Using a standard stratus cloud test-case the overhead of IO (i.e. the elapsed time between a MONC computational process communicating a value at a specific model time that will induce a file write and that write then being completed) over 32768 computational cores (36045 cores total, including IO server cores) reduced from 12.02 seconds to 8.92 seconds by adopting thread serialised instead of thread multiple mode.

It was also clear that certain NetCDF collective operations, where all IO servers block and wait for every other IO server to participate in the call, such as the closing of a file were resulting in a significant performance impact. An active messaging approach underlies the IO server communications, where communications are provided with a callback which is executed if appropriate when the communication completes. This mechanism was taken advantage of to avoid excessive synchronisation via the barrier active messaging call. Threads in IO servers execute the non-blocking barrier call, providing a unique identifier and callback function. At some point in the future, when all IO servers have executed this active barrier with the corresponding unique identifier, then a thread is activated on every IO server executing the callback subroutine which itself

performs NetCDF blocking calls, such as the definition or closing of a file. Because the semantics of an active barrier is that a specific callback should be executed once all IO servers reach a specific point we can ensure that the blocking calls are all called at roughly the same point in time and there is far less waiting in this NetCDF blocked state which both blocks any other thread in the IO server executing NetCDF calls but also MPI a NetCDF uses MPI-IO.

In addition to writing out diagnostic values, it is also sometimes desirable to write out the raw prognostic fields, for instance when checkpointing or for provenance. However these fields can be very large and this was a very significant overhead, often taking many minutes to write a single field. For performance not only is it important to do the writing collectively, so that each IO server will write out its specific contribution to the overall global field at the same time, but we also want to minimise the number of writes being performed. If each MONC computational core was to do the writing of its prognostic fields itself then this would not be an issue because local prognostic data is contiguous, however with an IO server many computational cores might be sending their prognostic data for writing at different points and these represent different, potentially non-contiguous, chunks of the global domain.

At start-up, once each computational core has registered with its corresponding IO server, a search is performed by each IO server over the sub-domain location of its MONC computational cores. The aim is to combine as many sub-domains as possible into contiguous chunks in order to minimise the number of writes. Because it is often the case that MONC computational cores of a processor are neighbours, working on geographically similar areas of the global domain, it is often possible to result in one or two large writes which is desirable. This is illustrated by figure 10 where the algorithm will search in both dimensions and for each chunk will determine the what contiguous chunks it could be a member of. Starting in the horizontal, the algorithm would identify that chunks A, B and C could form a contiguous block, but it will then progress to the vertical and determine that chunks C, D, E and F can form a contiguous block. Larger chunks are preferred over smaller ones, so in this case two regions (and hence writes) are selected, the first containing chunks A and B, and the second chunks C, D, E and F.

Memory space is allocated for these contiguous buffers and prognostic data from local computational cores are copied into the buffer which is then written to file once all contributions have been made. Because writing of these fields is collective, all IO servers must participate for every write, therefore a reduction with the *max* operator is performed on initialisation to determine the maximum number of collective writes that any IO server will issue. Any IO servers with fewer writes than this will issue dummy (i.e. empty, of zero count) writes so that it is still involved in this collective operation.

4.4 IO server performance

Performance and scalability tests for the IO server have been carried out on ARCHER, a standard stratus test case was used which models stratus cloud in

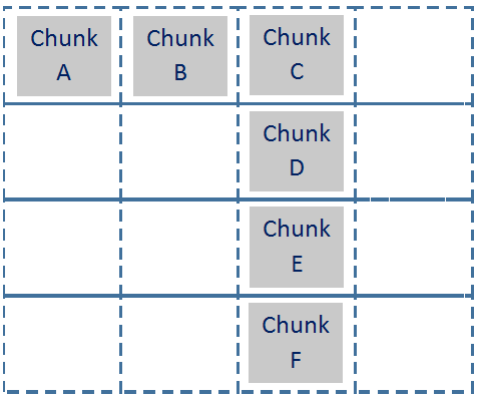

Figure 10: Illustration of contributed data chunks in the global domain

the atmosphere. 232 diagnostic values are calculated every timestep and time averaged with a result every 10 model seconds, the NetCDF file is written every 100 model seconds and the run terminates after 2000 model seconds. Figure 11 illustrates the performance of the model with and without data analytics at different core counts. In this experiment we are weak scaling, with a local problem size of 65536 grid points and on 32768 cores this equates to 2.1 billion global grid points. It can be seen that there is an impact of enabling data analytics but this is fairly minimal, a 8.14 seconds or 2.6% impact at 32768 cores. In all cases the IO server was run in thread serialised mode with hyper-threading enabled (which has been found to afford optimal performance.)

More details about the IO server work can be found in [3].

5 Miscellaneous

As mentioned in section 3.2, this work has been used as a basis for an EPCC MSc in HPC dissertation [16] in 2016 based around porting the CASIM micro-physics scheme to GPUs and understanding their applicability to these sorts of models. Another EPCC student MSc dissertation project around the tracking of clouds generated by CASIM was also proposed by the author and completed by one of our 2015-2016 cohort [15]. The author of this work has also proposed a project for the 2017 MSc in HPC dissertation, which a student is completing, about replacing the non-blocking P2P calls for halo swapping, and collectives in the FFT solver in MONC with MPI RMA equivalents. Whilst MPI RMA has been around since version 2.0 of the standard, version 3 developed a number of very significant improvements and recently MPI implementations, such as that on the Cray, have taken advantage of these in a much more performant manner than previously. It will be especially interesting to see the differences in performance that RMA affords for the FFT solver as, as discussed in section 2,

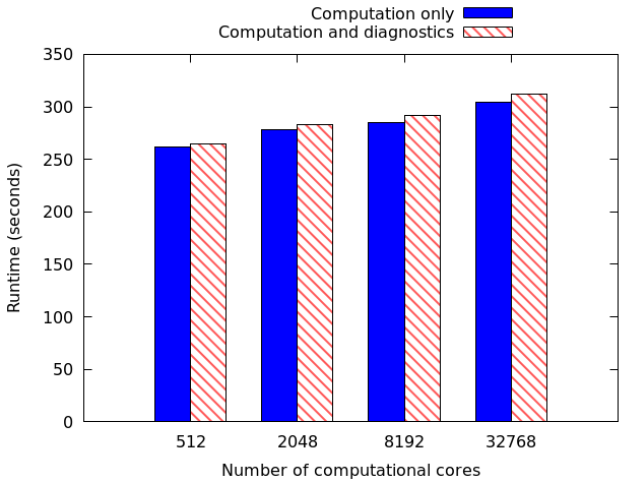

Figure 11: Overall MONC runtime when weak scaling with and without data analytics, 65536 grid points per core

the all to all communications means that the scalability of the solver decreases as one reaches larger core counts, hence could RMA assist here and maintain the computational benefits of this solver (over the iterative solver) for larger numbers of cores.

The author hosted a Summer of HPC student who used MONC and CASIM as part of a project developing an outreach demonstration activity. Utilising the VTK framework, the public can set up initial conditions and then simulate the development of clouds and rain over both land and sea by running MONC on Wee Archie, a mini supercomputer built out of Raspberry PIs, and viewing the weather change in real time. This has been used at a number of outreach events including Bang Goes the Borders, as well as being used by the EU funded Future Emerging Art and Technology (FEAT) project as part of an art piece developed around supercomputing which is on display at the Dundee LifeSpace Science Art Research Gallery between 3th April and 17th June 2017. It is also very likely that this art piece, including MONC running on Wee Archie will be used in further exhibitions throughout 2017 and the author is hosting a Summer of HPC student this year (2017) who will be targetting this demo towards early undergraduate educational uses.

6 Conclusions

In this technical report we have summarised the work done as part of the MONC eCSE project on both the core computational MONC model as well as other supporting facets such as the CASIM microphysics scheme and in-situ data

analytics. All the developments, bug fixes and improvements that we have developed are currently in the trunk on the Met Office science repository and, as at the time of writing, version 1.0 of MONC is ready to be released to the user community as soon as final testing has been completed. This work has been varied, encompassing a number of different areas and both benefiting current users of MONC (such as improvements to the in-situ data analytics and CASIM), and also future uses such as the further development of a pressure solver that will scale to far larger core counts and also demonstrating the applicability of novel hardware for CASIM.

In terms of further work the KNL has potential to be a very useful architecture for these sorts of models, whilst CASIM has shown benefit when running on the KNL the rest of the MONC model can not take advantage of this architecture at the moment, not least, due to the lack of threading elsewhere. A natural next step would be to optimise MONC for the KNL, in [3] we looked at in-situ data analytics on the KNL and found some advantage to running an IO server on the hyper-thread of every core, although this requires further work to test more fully and optimise.

Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)

References

[1] Nick Brown, Michele Weiland, Adrian Hill, Ben Shipway, Chris Maynard, Thomas Allen, and Mike Rezny. A highly scalable met office nerc cloud model. In *In Proceedings of the third International Conference on Exascale Applications and Software*. EASC 2015, April 2015.

[2] Nick Brown, Alex Nigay, Michele Weiland, Adrian Hill and Ben Shipway. Porting the microphysics model CASIM to GPU and KNL Cray machines Accepted into the Cray User Group conference (CUG) 2017

[3] Nick Brown, Michele Weiland, Adrian Hill and Ben Shipway. In-situ data analytics for highly scalable cloud modelling on Cray machines Accepted into the Cray User Group conference (CUG) 2017

[4] A.R. Brown, M.E.B Gray, and M.K. MacVean. Large-eddy simulation on a parallel computer. *Turbulence and diffusion*, (240), 1997.

[5] B.J. Shipway, and A.A. Hill. Diagnosis of systematic differences between multiple parametrizations of warm rain microphysics using a kinematic framework. *Quarterly Journal of the Royal Meteorological Society*, 138.669, 2012

[6] M.E.B Gray, J. Petch, S.H. Derbyshire, A.R. Brown, A.P. Lock, H.A. Swann, and P.R.A. Brown. Version 2.3 of the met office large eddy model. part 2, scientific documentation. 2003.

[7] Omar Awile and Ivo F Sbalzarini. A pthreads wrapper for fortran 2003. *ACM Transactions on Mathematical Software (TOMS)*, 40(3):19, 2014.

[8] A.A. Hill, P.R. Field, K. Furtado, A. Korolev, and B.J. Shipway. Mixed-phase clouds in a turbulent environment. part 1. large-eddy simulation experiments. *Quarterly Journal of the Royal Meteorological Society*, 140(680), 2014.

[9] A.P. Lock. The parametrization of entrainment in cloudy boundary layers. *Quarterly Journal of the Royal Meteorological Society*, 124(552), 1998.

[10] A.P. Lock, A.R. Brown, M.R. Bush, G.M. Martin, and R.N.B. Smith. A new boundary layer mixing scheme. part i. scheme description and single-column model tests. part ii. tests in climate and mesoscale models. *Monthly Weather Review*, 128(9), 2000.

[11] J.C. Petch. Sensitivity studies of developing convection in a cloud-resolving model. *Quarterly Journal of the Royal Meteorological Society*, 132(615), 2006.

[12] J.C. Petch and M.E.B. Gray. Sensitivity studies using a cloud-resolving model simulation of the tropical west pacific. *Quarterly Journal of the Royal Meteorological Society*, 127(557), 2001.

[13] S.J. Abel and B.J. Shipway. A comparison of cloud-resolving model simulations of trade wind cumulus with aircraft observations taken during rico. *Quarterly Journal of the Royal Meteorological Society*, 133(624), 2007.

[14] J.s Jeffers and J. Reinders Intel Xeon Phi Coprocessor High Performance Programming ISBN: 9780124104143

[15] L.Q. Yasel Parallel Post-processing of Met Office Cloud Data At *https://static.ph.ed.ac.uk/dissertations/hpc-msc/2015-2016/Yasel_Quintero_lares-Dissertation-B080173.pdf*

[16] A. Nigay Accelerating the microphysics model CASIM using OpenACC At *https://static.ph.ed.ac.uk/dissertations/hpc-msc/2015-2016/Alex_Nigay-igay-Accelerating-the-microphysics-model-CASIM-using-OpenACC.pdf*