MIKELANGELO

D2.10
The First Aerodynamic Map Use Case Implementation Strategy

Workpackage:	2	Use case & Architecture Analysis	
Author(s):	MatejAndrejašič		Pipistrel
	Gregor Berginc		XLAB
Reviewer	NadavHar'El		Cloudius
Reviewer	Peter Chronz		GWDG
Dissemination Level	public		

Date	Author	Comments	Version	Status
2015-08-11	Gregor Berginc/MatejAndrejašič	Initial draft	V0.0	Draft
2015-08-18	Gregor Berginc/MatejAndrejašič	Enhancements and refinements	V0.1	Draft
2015-08-24	Gregor Berginc/MatejAndrejašič	Document ready for review	V0.2	Review
2015-08-31	Gregor Berginc/MatejAndrejašič	Document ready for submission	V1.0	Final

Executive Summary

The purpose of this document is to provide the reader information regarding the first aerodynamic map use case implementation strategy. It represents the first of three deliverables regarding this use case. The aerodynamic map use case is one of the four use cases in MIKELANGELO project, whose vision is to improve responsiveness, agility and security of the virtual infrastructure through packaged applications, using the lean guest operating system OSv and I/O-optimised hypervisor sKVM. The use case will carry out experiments to evaluate and validate the strengths of the Mikelangelo stack.

This report describes the use case, explains the limitations of the state of the art and it explains how the MIKELANGELO project will improve the use cases's workflow. The report comprises the project's key performance indicators (KPI) and stresses their relevance to our use case. The document further explains the implementation plan of this use case, which will mainly focus on improving support inside OSv operating system and integrating it with a cloud-based management system provided by the MIKELANGELO stack. OpenFOAM has already been configured and recompiled in a way suitable for running within OSv as a single process. However, running simulations in parallel is not possible at the moment inside OSv because the parallelism in OpenFOAM is built on top of MPI exploiting multiple processes which are not supported in OSv. Running simulations in parallel will be one of the biggest challenges in this use case.

The report concludes with plans for evaluation and validation. They present the baselines planned to be used in order to demonstrate work progress and eventually the strengths of the entire MIKELANGELO stack: optimised I/O, virtualised RDMA-based communication between workers, flexibility of the guest OS and security.

Table of Contents

Table of Figures

Table of Tables

List of Abbreviations

AoA	Angle of Attack
CFD	Computational Fluid Dynamics
CLI	Command Line Interface
FVM	Finite Volume Method
HC	Heavy OpenFOAM Case
HPC	High Performance Computing
IB	InfiniBand
KPI	Key Performance Indicators
KVM	Kernel-based Virtual Machine
MPI	Message Passing Interface
ORTE	Open Run-Time Environment
RDMA	Remote Direct Memory Access
Re	Reynolds Number
RoCE	RDMA over Converged Ethernet
SC	Simple OpenFOAM Case
sKVM	Superfast KVM (codename of the MIKELANGELO KVM version)
VCPU	Virtual CPU
VM	Virtual Machine

1 Introduction

The aim of this document is to provide the reader with information regarding the first aerodynamic map use-case implementation-strategy. It represents the first of three deliverables regarding the aerodynamic map use case and explains the current state of the use case, its plans and goals. The next two iterations of this deliverable will report on updates of the use case, its implementation and its results.

The aerodynamic-map use-case represents one of the four use cases in the MIKELANGELO project [1], whose vision is to improve responsiveness, agility and security of the virtual infrastructure through packaged applications, using the lean guest operating system OSv and I/O-optimised hypervisor sKVM. The use case will carry out experiments to evaluate and validate the strengths of the Mikelangelo stack. Simple airfoil analysis, which typically takes minutes to converge, will allow for quick evaluation of the tools developed in the project, while the study of full 3D configurations will demonstrate the industrial relevance of the proposed tools.

The use case is presented in detail in Section 2, which includes the use-case description, current limitations and expectations for the MIKELANGELO stack. Section 3 contains the planned use case set-up. It explains the physical hardware and the software that is planned to be used. It also presents the execution workflow, characterizes the input and output data and describes the security constraints. In addition is lists all mandatory requirements that need to be implemented in the MIKELANGELO stack in order to achieve the maximum performance gains. At the end of the Section 3 KPIs are presented with stressed relevancy to the use case.

Section 4 introduces an in-depth analysis of OpenFOAM [2] in the context of this use case, focusing on the simpleFoam solver. The implementation plan is presented in Section 5 and the evaluation and validation plan is introduced in Section 6. The implementation plan explains how OpenFOAM has been configured for OSv and it discusses what else needs to be done in order to be able to run OpenFOAM simulations in OSv in parallel. The evaluation and validation plan presents the baselines and explains their focus on different aspects of running OpenFOAM simulations on top of existing architectures. This will provide the basis for evaluation of the components of the MIKELANGELO stack. At the end of the document key concluding remarks are presented in the Section 7.

2 Use Case Definition

On one hand the aerodynamics use case plans to use cloud computing in order to run as many similar OpenFOAM cases simultaneously as possible. On the other hand it plans to run a single computationally intensive simulation using all available cores in HPC cluster.

The former need arises in an industrial environment when a particular aerodynamic configuration, such as an aircraft, sailboat or a car needs to be analysed under a set of conditions, such as varying the angle of attack, sideslip angle and propeller thrust. While the variation of a single parameter requires only a couple of dozen cases to be run, the number of cases increases exponentially with the number of simultaneous parameters to be varied. Such an application is therefore very well suited for cloud computing, as a single case may not be very demanding (can be run on maximum one node), but the need exists to run a large number of them on independent virtual machines. It should be stressed here that a "case" or an "OpenFOAM case" is a term used to represent a single OpenFOAM aerodynamic simulation of an airflow around a chosen geometry at one set of parameters, whereas a "use case" is used to address the complete Aerodynamic map use case under the MIKELANGELO project.

On the other hand running a single computationally demanding problem is needed when a set of parameters is already chosen but a physically or numerically more precise simulation is required. An example of such a simulation is airflow past a large and diverse aerodynamic geometry where certain accuracy still needs to be assured. OpenFOAM is based on Finite Volume Method (FVM) that needs a mesh (an assemble of 3D cells that represents a volume around an aerodynamic body) within which the airflow is being simulated. A large and geometrically diverse object therefore needs a large mesh (large number of cells) in order to satisfactorily describe its shape. Such a simulation can become even more demanding when incorporating more accurate physical models or numerical schemes. Even though this is a typical HPC problem, it is still planned to be used as a baseline in order to increase the virtualised I/O efficiency of MIKELANGELO cloud stack.

2.1 Use Case Description

The cases to be studied will be from simple 2D airfoil analyses, which we call Simple Case (SC), and from full 3D configurations, which we call Heavy Case (HC). Both SC and HC will be run on as many different sets of parameters as possible in a simultaneous manner, while a heavy HC will be prepared in order to be run in parallel on multiple nodes. An example of a SC geometry, an airfoil, is depicted inFigure 1. This contour presents a 2D shape of a wing as seen in a cross-section at some chosen point along the wing. A complete airplane wing therefore consists of an array of airfoils along the wing in span-wise direction that is lofted between each other. During an aerodynamic design of a wing the designer must first know the characteristics of its basic building blocks, the airfoils. The designer must know how the airfoil behaves at different angles of attack (AoA) and Reynolds numbers (Re) in order to estimate how the wing will behave in different flight regimes (take-off, cruise, landing). Typical quantities that are needed are coefficients of lift, drag and moment.

An example of lift coefficient dependence with respect to the AoA at a single Re is depicted inFigure 2. If the AoA increases, the lift coefficient increases accordingly but only to the point where the airfoil stalls and the lift coefficient drops. Although there is a continuous curve presented on the plot, in reality the designer is limited in time and computer resources and is therefore not able to calculate a desired number of points on the curve. The computational complexity scales exponentially with the number of additional parameters introduced such as Re.

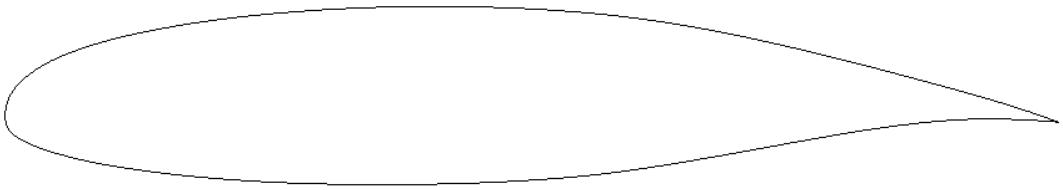

Figure 1: An example of a SC geometry, an airfoil.

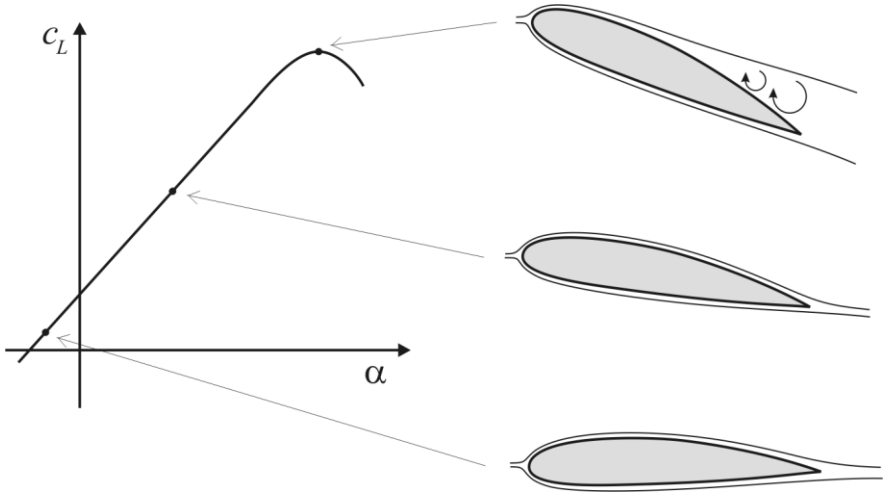

Figure 2: An example of a lift to AoA dependence.

One of the objectives of this use case is therefore to run a large number of simulations simultaneously to obtain results at all needed sets of parameters, that is AoA and Re, at approximately the same time. Pipistrel's in-house cluster currently consists of 2 nodes, each with 8 cores, which results in 16 simulations at a time, if each one is run on a single core. Using the MIKELANGELO stack Pipistrel will be able to employ much larger machines,

which will allow a larger number of simultaneous simulations. Even more important, Pipistrel will gain the know-how to use not only HPC, but also the cloud-based hardware and software, which will introduce greater flexibility to the workflow and possibly also reduce fixed operating costs in the future.

The second use case to be studied under the MIKELANGELO project consists of a wing and a larger number of propellers in front of the wing (Figure 3). In this HC a distributed propulsion system will therefore be studied. Parameters of interest beside AoA and Re are the side-slip angle, the propeller's position and its thrust. Besides computationally more intensive simulations with respect to the SC, there is also a larger number of parameters to vary. The plan is to start with a single propeller and corresponding wing section (Figure 4) in order to be able to run the problem on a single node and to finish with a complete wing run on multiple nodes. Intermediate steps will consist of gradually increasing the mesh size and the physics of the problem. The final full wing simulation will be run with a single set of parameters, chosen according to the lessons learned from previous steps. The objective is therefore to study the position and thrust of propellers in order to obtain satisfactory wing flight characteristics.

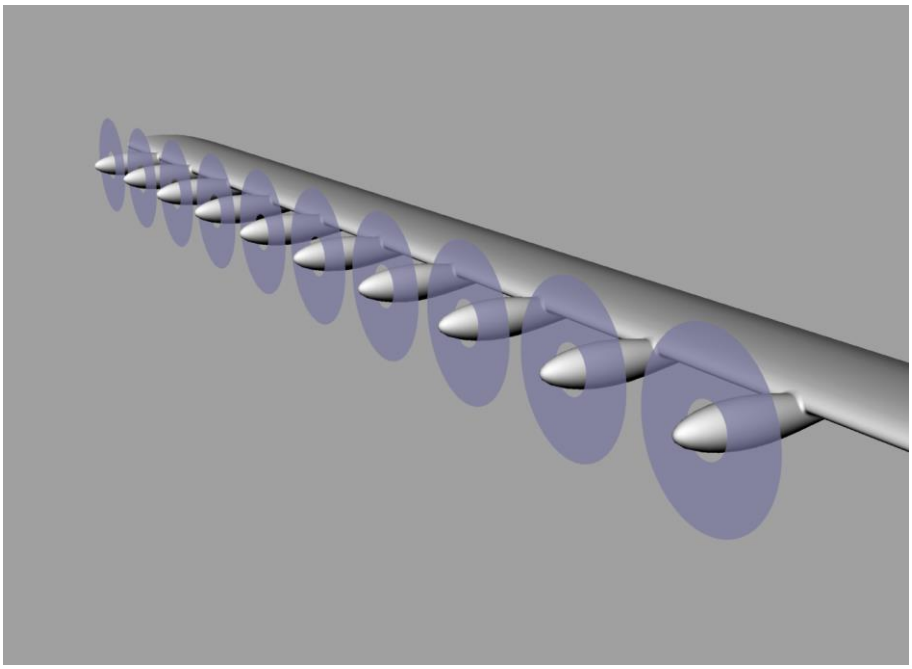

Figure 3: Complete wing with several propellers in front.

The objectives of a case with a single propeller and the corresponding wing section are similar to the SC that is to increase the number of cases run simultaneously to accelerate each simulation with MIKELANGELO stack optimization and to improve the agility of application

deployment. The latter consists of replacement of current bash scripting with a more efficient, simpler and user-friendly GUI interface that will allow the user to choose the parameters to vary with the corresponding values. On the other hand, the objective in the case of the final full wing simulation at a single set of parameters run in parallel on several nodes is to show the increase of efficiency in virtualised I/O of MIKELANGELO stack.

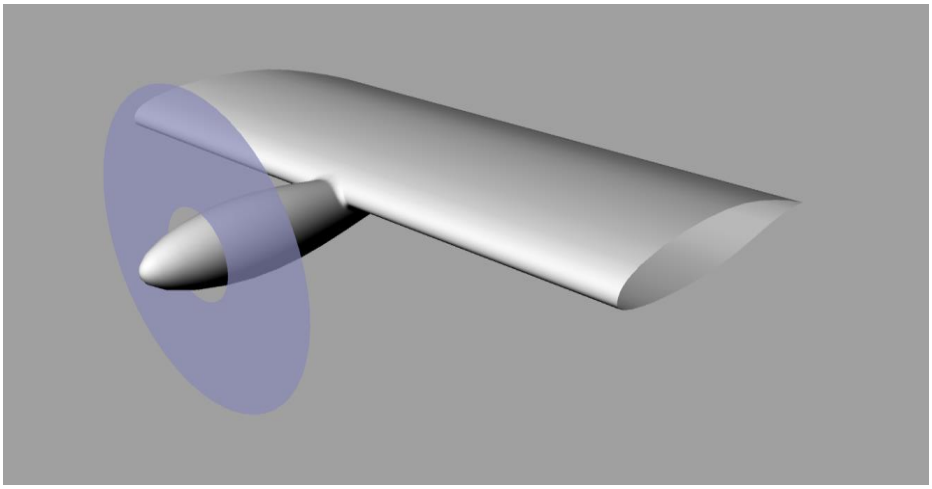

Figure 4: A single propeller with a corresponding wing section.

2.2 Current Limitations

Currently, the workflow is governed by a scripted copying of a template case, adjusting the parameters of each case, running of all cases and collecting the data from all cases. While the process works, it is typically limited to a single node or at best a cluster of nodes if one is available. This scripting approach severely limits the scope of analysis, especially if a number of parameters are being adjusted simultaneously. Typical parametric studies require a large number of OpenFOAM cases to attain the necessary parameter resolution. Furthermore, the addition of extra cases in areas of parameters where greater parameter resolution is needed may prove very cumbersome to achieve, using simple bash script modifications. Increased resolution often means that in the case of only a limited number of available nodes, the cases need to be scheduled in batches, unnecessarily increasing the overall computation time (wall time). Although individual cases do not have high memory requirements, they start competing for memory, disk (I/O) and processor resources when run on a single node with many cores.

Batch systems, such as PBS [3] or its free alternative Torque [4], greatly simplify execution of many cases allowing submission of jobs into a shared queue where they are started by the resource scheduler once resources are available. The end user is notified via email when individual jobs are finished and the results are typically stored in a shared workspace. One significant limitation of this approach is that the user has almost no control of when the jobs

are scheduled for execution, which complicates the planning process since changes in the product design are typically based on the results of previous experiments.

These limitations led to provisioning of more flexible approaches to running intensive simulations on top of high performing clusters. For example, UberCloud [5] is using containers to promote light-weight deployment of OpenFOAM and other HPC simulation frameworks and engines. These are already pre-packaged with all tools required for starting simulations, such aspre-processing, processing and post-processing. This significantly simplifies the management of underlying compute nodes which relieves system administrators from having to maintain different versions of software packages used by end users. However, deploying changes into these packages is cumbersome and requires support from UberCloud and changes requested by users typically affect all others. Furthermore, although container-based technologies have recently been popularised by the advent of the application packaging and management technology Docker [6] there are still important drawbacks, some of which are mentioned in the list below:

- **Security** aspects of using containers are not well researched and are so far considered less secure than environments with full virtualisation. In some cases, containers are even run within a virtual machine, mostly due to security constraints.
- **Hardware support:** containers also have limited support for specialised hardware such as Infiniband, in particular when sharing resources. This limitation will be evaluated by this use case, as part of a baseline experiment.
- **Kernel features:** containers typically rely on very recent kernel updates. Maintaining hundreds or even thousands of compute nodes requires careful planning of infrastructure maintenance and does not allow for such frequent upgrades of core packages.

2.3 Expectations from the MIKELANGELO Stack

The case analysis would ideally consist of an application that builds on an existing template case, in which the parameters to be varied are designated. The application would allow the user to define a set of parameter through a GUI and prepare the cases accordingly. The cases would then be deployed automatically to the cloud. The application would allow the user to monitor the execution and report possible crashes. When the cases are completed, the relevant information would be assembled from all of them and entered into a common database of computational results for the analysis. The application would further allow running more cases in the areas of parameter values of more detailed interest. The full computational results would reside on the nodes for a certain period of time, so that they may be checked for subsequent validity. This may be simplified by pre-selecting only a few cases to be retained, before deploying them.

Running the cases on the cloud allows them to have available all the necessary resources of a single node for each individual case. By using cloud resources, many more cases can then be run simultaneously, significantly increasing the parameter resolution without increasing

computational times. A streamlined workflow with a well-designed results database would allow the analyst to focus on the details of the results and not on preparation of cases.

3 Use Case Set-up

3.1 Physical Hardware

As we have described in the Section 2, the primary goal of the aerodynamics use case is to analyse the 2D/3D model of an aircraft or part thereof under a number of varying parameters. From the construction point of view, the overall experiment consists of a series of rather small simulations running between a few minutes and few hours. The execution time depends primarily on the resolution of the input model.

It is estimated that the actual use case, once in full production, will range from 100 to 1000 input cases with varying input parameters. Each case would be run on 1-8 cores with 1GB of memory. The exact number of processed cases will depend on the number of available resources.

The initial testbed, used during the implementation and initial benchmarking of the use case will be based on XLAB's private small-scale cloud. A high-level overview of the infrastructure is presented in theFigure 5, showing the clusters and network components. The two clusters, namely OpenStack and Woody, are comprised of several nodes. The four OpenStack nodes are interconnected with 10 Gbit network, however neither the switch nor the network cables are high-end.

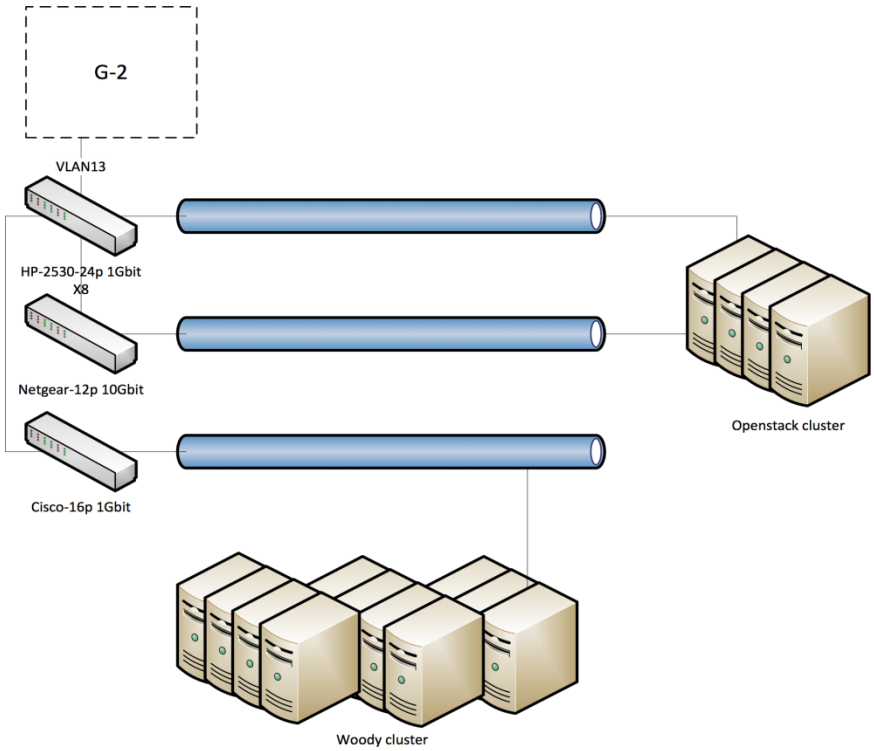

Figure 5: High-level overview of the XLAB infrastructure.

The physical infrastructure provides 29 TB of available disk space, 672 GB of available RAM and 90 CPU cores.

Additional experiments are planned on both the HLRS HPC infrastructure and GWDG Cloud infrastructure. These two are described in reports D2.1 and D2.7.

3.2 Software

The aerodynamics use case has been chosen as an important representative of the broader OpenFOAM user community to help to design, implement, validate and evaluate the whole MIKELANGELO software stack.

OpenFOAM is one of the most popular and commonly used open source packages to perform computational fluid dynamics (CFD). It allows researchers and product designers to analyse the flow of gas or liquids through time. Development of OpenFOAM started in the late 1980s and was initially released as open source software in 2004. Since then it has evolved into a stable and extensive CFD platform with its latest stable release (2.4.0), which will be used within this use case, in May 2015.

One of the crucial benefits of using OpenFOAM is the simplicity of constructing complex CFD tasks by providing basic mathematical, physical and computational building blocks. On top of these blocks few high level applications are already available as part of the OpenFOAM distribution. Furthermore, third-party software developers can use these blocks to tailor simulations to their needs. When using applications that are part of the OpenFOAM distribution, the end user is required to provide the 2D or 3D input case in an appropriate format and configure the simulation provided by the application. Allowing the application developer to modify the entire workflow of the simulation allows them to change the configuration parameters of the input case in the latter case.

The extensibility of OpenFOAM furthermore allows to scale applications from a single node to thousands of nodes without any changes to the application code. The power of parallelism is built into the underlying building blocks that are provided to the application software developer. Jobs that run on a single node, or even a single core, are ideal in the initial phases of the product design when high level decisions for the general direction of the design are made. However, with the increased number of analysed parameters, resolution of the underlying model, complexity of the simulation and number of time-steps OpenFOAM simulations require highly performing clusters of compute nodes to reduce the wall time of the analysis.

More recently, some attempts have been made to support running OpenFOAM in more agile and flexible environments on top of container based technologies, such as Docker. This agility opens completely new business models for infrastructure providers enabling them to run OpenFOAM simulations on demand as a Software-as-a-service model.

MIKELANGELO will simplify deployment, management and execution of distributed simulations on top of an in-house/HPC cluster even more by empowering a cloud-like

interface to the underlying infrastructure. Storage and compute infrastructure will be abstracted by means of OpenStack cloud-services and transparently interconnected for optimal performance. Cross-layer optimisations to the hypervisor and guest operating system will reduce the overhead of virtualised environments and approach the native performance of the hardware, but with much improved manageability and security.

3.2.1 Selection

The aerodynamics use case will perform CFD simulations on some synthetic and real-world cases using the open source software OpenFOAM 2.4.0 or newer, once available. As we have already discussed in the previous section, OpenFOAM is an extraordinary open source project with many different simulation models and processes which we intend to exploit at to the fullest extent throughout the project to obtain best possible results for our business-driven use case.

OpenFOAM does not provide parallel execution on its own. Instead, it provides a high level abstraction of the parallel execution model in terms of a high level interface for various underlying parallel execution implementations. This high level interface is defined in the Pstream library, distributed as part of the OpenFOAM source package. The current version of OpenFOAM only provides a PStream plug-in for MPI (Message Passing Interface) implementations. Although we have mainly worked with OpenMPI [7], an open source implementation of the MPI protocol, other implementations of MPI, such as MPICH [8] and vendor-specific versions. are also supported.

In addition to OpenFOAM, the use case will rely on the following software packages

- Host operating system: for the first iteration of the use case, Ubuntu 14.04 has been chosen as host OS. The primary reason for choosing this version is that it is the latest long-term support release of Ubuntu. This version has also been chosen by the consortium as the first version to be supported by the MIKELANGELO stack due to its slim server image and compatibility with all of our use cases.
- Guest operating system
    - Ubuntu 14.04 will be used as a guest OS for the initial experiments.
    - The most recent version of OSv will be used as guest OS for additional experiments. Upstream OSv changes will also be considered.
- Linux kernel: A kernel with version 3.18.x, which is the latest stable kernel tree with long-term support will be used on the host. The long-term version is particularly important for the work done on the modified hypervisor (sKVM), which relies on a stable kernel code-base. It should be noted that Ubuntu 14.04 does not have Linux kernel 3.18.x by default so it is necessary to upgrade the kernel.
- OpenMPI: OpenMPI with version > 1.6.5 will be used for the first iteration of the use case. This is primarily because the initial analysis of the baselines indicates some changes may be needed to the tools supporting spawning of parallel workers.

Other software packages are currently neither required nor envisioned for the first iteration of the aerodynamics use case. There are also no specific requirements for the C/C++ compiler.

The following section provides an initial analysis of the OpenFOAM package and its relations with the MPI.

3.3 Execution

All input cases will be simulated using OpenFOAM's existing solvers, for example simpleFoam, serving as a proof of concept. Conceptually, the structure of different solvers is very similar:

- read the definition of the input case, simulation parameters and initial values for physical quantities,
- iterate a number of steps of the simulation,
- periodically provide output to the user for visualisation purposes,
- output final results after the execution.

This allows this use case to serve as a solid proof of concept for a broad range of problems that may be addressed with default OpenFOAM solvers as well as custom solvers/applications developed on top of OpenFOAM framework.

The complete workflow of an OpenFOAM application is described in detail in Section 4. Simulations may be executed within a single worker process or in parallel. Parallel execution requires two additional steps:

1. Domain decomposition: this step must be executed before the processing is started, otherwise the simulation will fail. The domain must be decomposed into the same number of sub-domains as there will be parallel processors working on these sub-domains. Contrary to the Bones use case where workers are independent and the final result is produced in the last stage, OpenFOAM workers must be fully synchronised to maintain proper simulation step on the boundaries of all sub-domains at every time step. Although each worker will initially load only its own sub-domain data, all sub-domains must be accessible to all workers. This is because the worker will choose the domain based on its MPI rank assigned during the bootstrapping of workers.
2. Result reconstruction: after all time steps are finished, partial results are stored in separate folders. The final step will take all results from these sub-folders and produce overall result.

Both these steps are efficient and require considerably less time than the simulation itself. They are also not parallelised. Ideally, input and output data are stored in a shared workspace eliminating the need to move inputs and outputs to and from worker nodes.

For development purposes a very small scale example will be used requiring 3-5 minutes on a single node. For the validation purposes, three different models will be provided in this first iteration. The model will have an increasing complexity and consequently run time: 15 minutes, 1 hour, and more than 4 hours when executed on a single core.

A case is prepared on a local machine. A management system is designed to handle parameters and distribute the cases over Virtual Machines (VMs), monitor the execution and collect the data at the end.

3.4 Data

All input and output data consists of text files. An estimated size of a single SC lies between 10MB and 50MB. A size of a single HC lies between 250MB and 4GB. A typical case consists of three folders. In folder "0" a complete mesh is defined together with all initial and boundary conditions. The folder "constant" consists of physical constants, turbulent model selection and all .stl files of the geometry used in corresponding case. The third folder "system" includes files with user-defined parameters that steer the simulation. The contents of this folder define which numerical schemes will be used during simulation, which solver should be used to solve a system of linear equations, how much iteration should be computed, and whether we want to calculate forces on the aerodynamic surfaces during the simulation.

Simulation results are written to separate time stampedfolders that let the expert to review the progress of the simulation. If the simulation converges satisfactorily only the most recent results folder needs to be kept in order to review the results. Although a large number of cases is planned to be run, only a small amount of data needs to be extracted from the cloud back to the local computer. The data we are interested in are forces and moments on specific aerodynamic bodies.

3.5 Security

The input data used in OpenFOAM simulations are typically highly sensitive. These data might include a 3D model of a new airplane or part thereof. Or it could show the design of the hydro plant. Thus, security of data access is of utmost importance when offering simulation software packages such as OpenFOAM as a service. The environment for execution and global storage, such as a SAN, need to be trusted by end users. However, virtual environments and sharing of physical resources open new ways for attackers to exploit the infrastructure to gain information about other tenants' data. For every commercial and even many research organisations it is vital that the underlying infrastructure takes appropriate measures to limit the ability for an attacker to obtain sensitive data.

3.6 Mandatory Requirements

The following is a list of requirements that need to implemented and integrated with cloud management to achieve maximum performance gains. The requirements list is categorised in the same way as it is in the requirements list. Each requirement further describes its importance for this use case.

Application compatibility

> ***Integrate NFS Client in OSv (#102).***OpenFOAM requires that all worker processes have access to the entire input set. Each worker only reads one single domain, i.e. part of the complete 3D model.

OpenFOAM running in OSv and sKVM (#37).OpenFOAM has already been ported to run within OSv, however parallel execution is still an open issue. This requirement refers to support for parallel simulations.

Guest virtio-rdma

DPDK integration on Guest (#70). This use case will benefit from using DPDK in that it will support much more efficient communication between workers using RDMA.

Integration of RDMA components with Ubuntu Guest (#75). Ubuntu is one of main targets for a guest OS for running this use case. The results will be compared to OSv benchmarks.

Integration of RDMA components with OSv guest (#73). Due to its small footprint and fast boot times, OSv may support execution of a large number of cases on virtual infrastructures. Since OSv will most probably not support running multiple workers within one guest, it should be possible to compensate via inter-VM communication through an efficient implementation of the RDMA.

Hypervisor

Hypervisor command line API (#3). The command line API for the hypervisor is important for the evaluation of the Mikelangelo stack. This interface will allow to seamlessly test execution for varying settings of the hypervisor.

Multi VMs shared memory communication (#7). Similarly to requirement #73 above. It is crucial that OSv-based VMs communicate efficiently.

Hypervisor support for Ubuntu guest (#10). Ubuntu is the host OS of choice and is therefore desired that it is supported.

Hypervisor support for OSv guest (#9).

Other low-level requirements for the hypervisor, such as those found in the categories “Hypervisor virtio-blk/scsi”, “Hypervisor virtio-net” and “Hypervisor virtual switch” indeliverable D2.19 The first MIKELANGELO architecture are important because they will support some of the aforementioned high-level requirements of this use case. We therefore do not provide extensive description of these requirements.

Infrastructure

Integration of the modified hypervisor with OpenStack (#18). As it was described in Section 2.2 current execution of simulations does not allow for extensive testing. Cloud integration is an important aspect of the MIKELANGELO project. This use case expects to benefit from the cloud integration because it significantly simplifies the execution of simulations. OpenStack, although not the only possible choice, would allow great improvement to thecurrent workflow by providing a graphical user interface similar to that of the Sahara dig data module.

Monitoring

Capture performance metrics of guest OS - OSv (#42), Capture performance metrics of host Hypervisor - sKVM (#41), Hardware Monitoring (#40). Capturing metrics from all components of MIKELANGELO is particularly important for the evaluation of this use case.

Monitoring GUI (#45). Having the ability to observe monitored values in a graphical user interface is a must for such a high-level business use-case as it gives the user the ability to observe patterns that would otherwise remain hidden.

Services/Applications Monitoring (#43). This is a highly desired capability of a flexible monitoring system allowing application developers to provide metrics specific to the application itself. One example of such a metric would be values provided by OpenFOAM during calculation. It would help to observe these development values through time and observe whether they are converging towards reasonable results.

OSv

All OSv requirements from deliverable D2.19 "The first MIKELANGELO architecture" are important for this use case because they will influence the run time of the simulation. As requirements are rather low-level we chose not to provide additional descriptions as they will be available in the overall architecture document.

Security

Intrusion detection on Hypervisor level (#96). When OpenFOAM simulations are executed in commercial offerings, it is fundamental that security measures are as high as possible. For example, when a new product is being designed it is extremely important to prevent anyone from having access to related 3D models and simulations. Therefore, this use case will pay special attention to this requirement and provide additional information for refinement of features in the intrusion detection module.

It is expected that this list will be revised by a number of additional requirements once more thorough benchmarks are possible in different deployment scenarios.

3.7 Key Performance Indicators

This section describes KPIs from the Grant Agreement of the MIKELANGELO project and proposes specific metrics that will be used to measure progress and provide the basis for the validation and evaluation of the MIKELANGELO stack. Both KPIs as well as metrics will be updated on a regular basis to provide deeper insights into evaluation. Only those KPIs were chosen that are relevant for this use case. We are going to extend the list of KPIs and metrics in each iteration of this report.

KPI1.1: relative efficiency of bursting a number of virtual machines. OpenFOAM cases are typically analysed under different starting conditions resulting in a large number of

processing requests. This KPI evaluates two aspects of bursting a number of virtual machines (processes):

Time required for a number of simultaneous simulations (this number will vary) to start processing.

Subjective evaluation of the workflow for starting a number of simultaneous simulations.

KPI2.1: relative efficiency of virtualized I/O between KVM and sKVM (developed in the project). Although OpenFOAM is a primarily CPU-bound application, I/O plays an important role for reading input files, writing results and for communication between worker processes. The following is a list of metrics that will be used to evaluate this KPI

Execution time in different scenarios over an ordinary network. These scenarios include configurations with a single worker, multiple workers on a single nodeand multiple workers on multiple nodes. This will show how in-house, small scale, clusters will benefit from MIKELANGELO.

Execution time in different scenarios, as above, over specialised interconnects, such as Infiniband and RoCE. The purpose of this evaluation will be to measure performance gains in high-performance clusters.

Additional low-level metrics will be chosen based on the hypervisor and guest OS. For example these metrics may include the number of exits, context switches and wait times.

KPI3.1: The relative improvement of efficiency of MIKELANGELO OSv over the traditional guest OS. During the first project year, Ubuntu 14.04 will be used as a representative of traditional guest OSs. The following metrics will be used to evaluate the progress of MIKELANGELO.

Size of virtual images.

The boot time as measured by the time required to spawn a new VM and start the processing on a single node using qemu or libvirt via its Command Line Interface (CLI).

The boot time as measured by the time required to spawn a new VM and begin processing on the OpenStack cluster.

Run times for simulations under different configurations regarding the number of worker processes, input cases, and available physical nodes.

KPI3.2: The relative improvement of efficiency [size, speed of execution] between the Baseline Guest OS vs. the MIKELANGELO OSv. This KPI focuses on benefits of the MIKELANGELO project for the OSv. Similar metrics as for KPI3.1 will be used to evaluate the progress.

KPI3.3: The relative improvement of compatibility between baseline and MIKELANGELO versions of OSv. The goal of this use case is to support transparent execution of OpenFOAM simulations, regardless of the underlying infrastructure. To measure this KPI the following metrics will be monitored:

OpenFOAM/OpenMPI support in OSv

Number of implemented and/or updated POSIX functions

Number of implemented and/or updated system calls

KPI4.1: A system for appropriately packaging of applications in MIKELANGELO OSv is provided. The following metrics are mostly subjective and will be evaluated by business end-users. This will be critically assessed and compared to other existing systems.

Flexibility of creating new application packages

Ability to merge different packages into single image

Ability to integrate with existing cloud management system, such as Open Stack, Amazon, etc

Ability to work with different package repositories, preferably simultaneously.

KPI4.2: A relative improvement of efficiency [time] between deploying a packaged and non-packaged application.

Time to build the package measured from the perspective of an end-user.

Time to deploy the package into a shared workspace and/or cloud management system from where it is possible to start virtual machines from these packages

Time required to start processing from an already packaged application compared to the time required to prepare the application and start processing in the cloud

Focus will be made on comparing these times running on a multi-node cluster

KPI5.1: Demonstration of a management infrastructure on diverse physical infrastructures. Since this use case may be executed in a heterogeneous cluster, such as a set of idle machines in-house, it is important to facilitate execution of applications on diverse physical infrastructures. No clear metric may be defined, however during the evaluation we intend to setup such a cluster to test the behaviour or the application.

KPI5.2: Relative efficiency [time, CPU, disk overheads] of traditional HPC over Cloud HPC offered in MIKELANGELO. This KPI will provide a high-level overview of most of the other KPIs evaluated on an fully integrated MIKELANGELO stack. Metrics chosen will be similar to KPI3.1 but measured from end-user's perspective.

KPI7.1: All use cases appropriately demonstrated. This KPI will be fulfilled by this use case once all typical uses of OpenFOAM simulations are supported by MIKELANGELO.

The KPI will therefore measure the number of typical uses of OpenFOAM. We expect that between 3 and 5 ways to run cases will be demonstrated, for example

> **Single worker on one case**. This is already supported with changes made to OSv and OpenFOAM build process.
>
> **Multiple workers on different cases (all initiated simultaneously)**. Running multiple workers, each calculating one input case, is already possible using time-consuming scripts. However, this metric will be fulfilled when starting multiple workers will be handled by the cloud management framework.
>
> **Multiple workers on a single case**. As presented in previous sections, this is currently unsupported within OSv and will be the main focus of this use case. During the evaluation we will analyse MIKENAGELO stack running multiple parallel workers on a single node and a number of nodes.

KPI7.2: Documentation of using MIKELANGELO in use cases - Best Practices tutorial. The entire workflow will be thoroughly documented allowing the broadest possible audience to use OpenFOAM and OpenMPI on top of MIKELANGELO. We will communicate intensively with external entities.

KPI7.3: Documentation of using MIKELANGELO in use cases - Documented Benefits. Benefits will be documented in all use case reports.

We have tried to identify relevant metrics for all of the KPIs that will be observed by the MIKELANGELO project as a whole. For those that are relevant we have provided initial metrics that will be used for evaluation of the project. Some metrics are rather subjective but we nevertheless decided to include them because it is important for the uptake of such complex stack to include also the feelings of business stakeholders.

An important outcome of this preliminary study is that the consortium has already provided progress towards the final goal. By extending the OSv kernel we have already been able to execute simple simulations within OSv which is a massive success.

4 Analysis of OpenFOAM Execution Modes

For the purposes of this section, the use of OpenFOAM with the simpleFoam solver has been studied to better understand the generic structure of OpenFOAM applications. It was further studied how the parallelism using MPI is built into the package because. According to the OpenFOAM documentation, most other solvers and applications based on the OpenFOAM library follow a similar approach and also support similar configuration options.

4.1 Parallel Execution

Before we describe the actual code, it is important to understand what it actually means to run an OpenFOAM application in parallel. OpenFOAM operates on a 3D mesh that is one of the most important parts of a particular input case. When OpenFOAM operates in parallel mode, it decomposes the entire model into several sub-regions, called domains in OpenFOAM, it runs the OpenFOAM solver on each of the domains in parallel, it exchanges information between adjacent solvers, and finally it reconstructs the partial solutions into one overall solution.

4.1.1 Domain Decomposition

The first step in running OpenFOAM in parallel is to decompose the entire domain space into two or more sub-domains. OpenFOAM provides a utility called decomposePar aiming to divide the domain with minimal effort while guaranteeing reasonable decomposition. The latter goal is mandatory because parallel OpenFOAM must ensure to provide the exact same solution as in single process execution. Since the decomposition divides the entire 3D mesh, parallel processes have to communicate between each other. However, communication is only needed between adjacent sub-domains. It is therefore important that the decomposition provides minimal interactions between different processes reducing the amount of time spent for communication.

The decomposition is configured in the system/decomposeParDict file part of the input case itself. A thorough description of the file is out of scope of this document, but there are several important parts one must understand prior to using it:

- **numberOfSubdomains** defines the number of sub-domains decomposePar should decompose the original domain into. It is typically set to the number of cores OpenFOAM will be executed on. Since there is no multi-threading within OpenFOAM this assumption is reasonable.
- **method** defines the method used for decomposing the domain. Available options are “simple”, “hierarchical”, “scotch”, “metis” and “manual”:
    - **simple** splits the domain in X, Y, Z directions,
    - **hierarchical** is similar to simple but allows changing the order of directions,
    - **scotch** requires no geometric input from the user and attempts to minimise the number of processor boundaries,

    - **metis** is similar to scotch, but uses a different algorithm: it decomposes the domain using the METIS algorithm, which tries to minimize the communication between processors. It allows specifying weights for the different processors if they have different performance.
    - **manual** allows users to manually decompose the domain and provide sub-domain configurations on their own.
- **<method>Coeffs** is the part of the configuration file where chosen method is configured in more details (for example, for hierarchical, the user may provide the order of directions and number of domains in each direction).

Further information on decomposeParDict is available in the OpenFOAM documentation [9]. Once the configuration is provided, decomposition can be invoked using

```
decomposePar-case/path/to/case
```

The output log provides valuable information, in particular the number of faces of the input 3D model shared with other processors. As a result, the input case contains a set of sub-folders called processorN (N=0, 1, 2, …) for every sub-domain, that is for each processor. If multiple nodes are involved in the OpenFOAM calculation, each node must have access to the entire input case directory, including sub-folders for other processors. It can either be shared via a network share or copied to all nodes involved in the calculation.

4.1.2 Running in Parallel

Once the domain is properly decomposed running OpenFOAM in parallel is just a matter of submitting a job via the MPI infrastructure layer. OpenFOAM defaults to using OpenMPI, which can be started by

```
mpirun-np N simpleFoam-case/path/to/case-parallel
```

In this case, N parallel processes will be spawned working in parallel on the same input case. It is assumed that prior to issuing this command, the domain has been decomposed into N sub-domains. Although mpirun itself is responsible for spawning a number of parallel processes, the last switch (-parallel) is mandatory as it tells OpenFOAM it should be initialised to run in parallel.

4.1.2.1 Running on Multiple Nodes

Running OpenFOAM, or any MPI application, on two or more compute nodes requires some initial configuration of all systems. First, one must distinguish the master node from all other nodes. The master node is the one from which the parallel execution is started or requested. Nevertheless, the master may also be used for actual calculation.

First, the master must have access to all compute nodes that are to be used in the calculation. The list of nodes is specified in a separate configuration file listing names and a number of processors available for MPI processes, for example:

```
host1
```

```
host2

host3cpu=4
```

Hostnames specified in this file must be resolvable on master allowing mpirun to spawn new processes. This can either be achieved by a DNS or by providing entries in the /etc/hosts file.

Once hostnames are resolvable, the master node must also be able to connect to every compute node via SSH protocol. The login uses the same username as the one used on the master to start calculation. Since MPI will not ask for passwords, the public key of the master node must be shared with all slaves. This can simply be achieved by creating a password-less key-pair

```
# ssh-keygen
```

and adding the public key (~/.ssh/id_rsa.pub) into the list of authorized keys (~/.ssh/authorized_keys) on all slaves. You may verify that you are able to connect to slaves via SSH with the following command:

```
# ssh<slave-hostname>
```

It is important that no username should be provided in the above command as the MPI will try to log as the user starting mpirun on the master.

Once the master is able to connect to all slaves it is time to make sure all slaves are configured in the same way with respect to the OpenFOAM. The mpirun command will always use the current working directory on the master on all compute nodes, meaning that all nodes must have OpenFOAM installed at exactly the same location. For example, when compiling OpenFOAM from sources, you may have OpenFOAM stored in ~/OpenFOAM/OpenFOAM-2.4.0 and simpleFoam in

```
~/OpenFOAM/OpenFOAM-2.4.0/platforms/linux64GccDPOpt/bin/simpleFoam
```

Furthermore, you must also ensure that the OpenFOAM'sinit script is executed even for non-interactive (SSH) shells like the one that will be used by MPI. This depends on the Linux version, but on Ubuntu 14.04 it can be done simply by putting

```
.~/OpenFOAM/OpenFOAM-2.4.0/etc/bashrc
```

at the very top of the ~/.bashrc (if one is using default ~/.bashrc, chances are there a condition skipping all settings when not running interactively). This must again be configured on all slaves.

Last step before the processing may start is distributing the input case that has already been decomposed. When NFS or another shared workspace is used, this is not required as all compute nodes will have access to the same input case. However, all nodes must mount the shared workspace at the same mount point. OpenFOAM can be run without a shared workspace by copying the input case to all nodes. Although each process will use only the processorN sub-folder of the decomposed input case, the entire case must be copied to all slaves because MPI will assign processor IDs (ranks) dynamically.

Finally, running OpenFOAM on several compute nodes can be invoked by the following call:

```
mpirun -np N --hostfile=/path/to/hostfilesimpleFoam -case/path/to/case -parallel
```

4.1.3 Reconstruction of Partial Results

Following a successful execution, the user must finally merge partial results into an overall solution. The case is reconstructed by merging the sets of timestamped directories from each processorN directory into a single set of timestamped directories. The reconstruction can be invoked using

```
reconstructPar -case/path/to/case
```

producing the final set of timestamped directories.

4.2 The Orted Daemon

When a single compute node is used for parallel execution mpirun spawns several processes directly. When multiple nodes are used, mpirun connects to each of the slaves listed in the hosts file via SSH and starts the orted daemon, part of ORTE (Open Run-Time Environment), which is finally responsible for spawning processes on remote slave nodes. The exact command that is used to start orted on a slave is

```
ssh -x <slave-name>orted --daemonize -mcaessenv -mcaorte_ess_jobid2826240000 -mcaorte_ess_vpid1 -
mcaorte_ess_num_procs2 --hnp-
uri2826240000.0;tcp://172.16.118.65:345602826240000.0;tcp://172.16.118.65:34560 -mcaplmrsh
```

The following is an example of the orted command line while started with two processors on two different nodes:

```
orted --daemonize -mcaessenv -mcaorte_ess_jobid2826240000 -mcaorte_ess_vpid1 -
mcaorte_ess_num_procs2 --hnp-uri2826240000.0;tcp://172.16.118.65:34560 -mcaplmrsh
```

The IP in the --hnp-uri flag is the IP of the master node.

4.3 Code Analysis

The following listing presents the main structure of the simpleFoam application.

```
#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "RASModel.H"
#include "simpleControl.H"
#include "fvIOoptionList.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
int main(intargc,char*argv[])
{
#include"setRootCase.H"
#include"createTime.H"
#include"createMesh.H"
#include"createFields.H"
#include"createFvOptions.H"
#include"initContinuityErrs.H"

simpleControl simple(mesh);

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Info<<"\nStarting time loop\n"<<endl;

while(simple.loop())
    {
Info<<"Time = "<<runTime.timeName()<<nl<<endl;

// --- Pressure-velocity SIMPLE corrector
        {
#include"UEqn.H"
#include"pEqn.H"
        }

        turbulence->correct();

runTime.write();

Info<<"ExecutionTime = "<<runTime.elapsedCpuTime()<<" s"
<< "  ClockTime = " <<runTime.elapsedClockTime() << " s"
<<nl<<endl;
    }
```

```
Info<<"End\n"<<endl;


return0;
}
```

Although the first #include within the main() function indicates only a simple definition of the input case definition it does a lot more as the following code listing from "src/OpenFOAM/include/setRootCase.H" reveals

```
Foam::argListargs(argc,argv);
if(!args.checkRootCase())
    {
Foam::FatalError.exit();
    }
```

The first line of this listing initialises the argList object passing arguments from main function to the object itself. At the top of the argList constructor there is a check for the "-parallel" switch found in the command line arguments that causes OpenFOAM application to initialise parallelism using the Pstream library. Here is a listing from "src/OpenFOAM/global/argList/argList.C":

```
// Check if this run is a parallel run by searching for any parallel option
// If found call runPar which might filter argv
for(intargI=0;argI<argc;++argI)
    {
if(argv[argI][0]=='-')
        {
constchar*optionName=&argv[argI][1];


if(validParOptions.found(optionName))
            {
parRunControl_.runPar(argc,argv);
break;
            }
        }
```

```
    }
```

Pstream library is a wrapper library allowing OpenFOAM to work with an arbitrary implementation of the MPI standard as well as any other infrastructure layer supporting the required mechanisms. Currently, there is also support for Gamma besides MPI. In the above code listing runPar is a simple function invoking the implementation of the init() function of the underlying infrastructure layer:

```
voidrunPar(int&argc,char**&argv)
    {
RunPar=true;


if(!Pstream::init(argc,argv))
        {
Info<<"Failed to start parallel run"<<endl;
Pstream::exit(1);
        }
    }
```

Finally, the following listing shows the MPI implementation of the init function:

```
boolFoam::UPstream::init(int&argc,char**&argv)
{
MPI_Init(&argc,&argv);


intnumprocs;
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
intmyRank;
MPI_Comm_rank(MPI_COMM_WORLD,&myRank);


if(debug)
    {
Pout<<"UPstream::init : initialised with numProcs:"<<numprocs
<< " myRank:" <<myRank<<endl;
    }
```

```
if(numprocs<=1)
    {
FatalErrorIn("UPstream::init(int&argc, char**&argv)")
<< "bool IPstream::init(int&argc, char**&argv) : "
"attempt to run parallel on 1 processor"
<< Foam::abort(FatalError);
    }



// Initialise parallel structure
setParRun(numprocs);
returntrue;
}
```

The most important part is the MPI_Init call at the beginning of the function making sure the MPI process is properly initialised. Since the OpenFOAM solver is started using the mpirun command

```
$ mpirun-np N simpleFoam-case/path/to/case-parallel
```

each simpleFoam processes spawned by the MPI infrastructure will call appropriate initialisation and get its own rank from MPI.

4.4 Communication Between Workers

The PStream library abstracts input and output layers by means of two additional classes, namely UIPread and UOPWrite. These provide read and write methods respectively. PStream supports three different communication modes: blocking, scheduled and non-blocking. The following presents the mapping between PStream modes and MPI modes as currently implemented within OpenFOAM:

- Read operations
  - blocking and scheduled: MPI_Recv
  - non-blocking: MPI_Irecv
- Write operations
  - blocking: MPI_Bsend
  - scheduled: MPI_Send
  - non-blocking: MPI_Isend

5 Implementation Plan

Since OpenFOAM is already a stable solution and since it relies on MPI to achieve parallelism, the implementation of this use case will mainly focus on improving support inside OSv and on integrating it with the cloud management system provided by the MIKELANGELO stack.

For the purpose of the initial analysis, OpenFOAM has already been configured and recompiled in a way suitable for running within OSv as a single process. Although OpenFOAM is a large open source project, only few minor changes to OSv were required to support additional system calls, such as 204 - sched_getaffinity and 239 - get_mempolicy, and standard functions, such as faccessat, __fxstatat and malloc_hook. Patches for all these modifications and additions to OSv have already been submitted by Cloudius Systems to upstream to OSv's source tree.

These patches already support running OpenFOAM on input cases provided by this use case by a single worker, that is as non-parallel version, resulting in the initial benchmark. We compare the results with those obtained with a Linux guest. Since parallel execution of OpenFOAM and OpenMPI is provided by spawning two or more processes, running simulations in parallel is not possible at the moment inside OSv. Solving this will be one of the biggest challenges for OSv and MIKELANGELO from the perspective of this use case.

The initial idea to support parallel execution was to try to start multiple threads instead of processes within a single OSv instance. However, we have already analysed OpenFOAM and discovered that it uses several global variables. Global variables are shared between all threads of one process, thus simply replacing MPI processes with MPI threads within one OSv process will cause problems. Multiple threads would be changing the same set of global variables instead of multiple versions of them. For example, what each MPI process treats as its input sub-domain would now be shared among all threads. Nevertheless, we intend to further assess whether replacing MPI processes with OSv threads is possible, for example by examining whether global variables may be used differently. This would facilitate execution of parallel simulations within one single VM.

However, running parallel simulations in different VMs would still require some changes to MPI. More specifically changes to the daemon responsible for bootstrapping workers would be required. Thus, the second approach to running OpenFOAM in parallel assumes that each worker is an individual OSv-based VM. This will not create significant overhead because starting a single OSv image that consequently starts the OpenFOAM calculation is almost instantaneous. However, this approach would require changes in the MPI middleware, for example in mpirun and in the orted daemon that have been described in the previous section, to bootstrap workers. With the improved RDMA support that MIKELANGELO will deliver, this separation of workers into VMs should not produce a significant overhead. However, it would significantly improve the flexibility of running experiments. Because this approach is not strictly related to the OpenFOAM, all MPI based applications would again from it.

6 Evaluation and Validation Plan

The aerodynamics use case intends to use several different baselines focusing on different aspects of running OpenFOAM-based simulations on top of existing architectures and comparing them with those available by the MIKELANGELO stack. The following list proposes the initial set of configurations used to benchmark and compare execution times as well as manageability of starting simulations and collection of the results. All Linux tests have been conducted using Ubuntu 14.04.03 (host and guest).

1. **Linux Host (host baseline)**. We evaluate the performance of OpenFOAM running 1, 2, 4 and 8 parallel processes.
2. **Linux Guest (virtualised baseline)**. Performance of OpenFOAM running 1, 2 and 4 parallel processes was examined.
3. **OSv Guest**. Current benchmarks only show results for running single OpenFOAM worker within OSv due to limitations.

All experiments with VMs have been performed using the version of KVM that was available in the Ubuntu 14.04.03. Furthermore, all VMs have been configured the same for Linux and OSv guests.

The experiments in the above list are targeting one single compute node. Once the high-end HPC testbed is available this list will be extended with additional experiments focusing on exploitation of high performing interconnects, such as Infiniband and RoCE.

6.1 Initial Baselines

This section presents the set of initial baselines for those configurations in the previous section that can already be executed The baselines focus on OSv compatibility and on execution times of reference simulations.

6.1.1 OSv Compatibility

One of the most important baselines is the support for different modes of operation of OpenFOAM and OpenMPI. As of writing this report OSv supports running a single worker within OSv. Prior to the MIKELANGELO project, starting OpenFOAM within OSv was not possible due to the nature of OpenFOAM compilation and missing standard functions from the OSv kernel and the standard library. These have already been resolved in the first 6 months of the project. Neither parallel execution on a single node nor multiple nodes is supported in OSv.

All these modes are supported by Linux hosts and Linux guests.

6.1.2 Execution Times

Many different parameters affect the duration of execution, as measured by the wall time, of the simulation. For the initial experiment we will focus on a small number of parameters, which allows us to provide some basic comparisons between different options. No kernel

optimisations have been performed and the same QEMU configuration has been used for Ubuntu and OSv guests. Total durations are shown in Table 1for three chosen input cases. Each experiment has been executed three times and only a single worker was used to solve the simulation.

Table 1: Execution times for three input cases using one worker process.

Configuration	mik3d_15min [s]	mik3d_1h [s]	mik3d_4h [s]
Ubuntu host	633	1634	10070
Ubuntu guest (1VCP)	646	1665	10235
Ubuntu guest (2 VCPU)	643	1656	10159
OSv guest (1 VCPU)	654	1680	10209

The durations for OSv also contain the time to boot and shutdown the VM while the Ubuntu guest was already running prior to starting the benchmark. It takes approximately 5-10 seconds before the processing in Ubuntu can start.

Table 2 shows total durations for different configurations when using two or more parallel workers for each input case.

Table 2: Execution times on the same three input cases using two or more MPI processes.

Configuration	Workers	mik3d_15min [s]	mik3d_1h [s]	mik3d_4h [s]	Average speed-up
Ubuntu host	2	534	1220	7605	1.28
Ubuntu host	4	461	1173	7450	1.37
Ubuntu host	8	450	1140	7311	1.41
Ubuntu guest (2 VCPU)	2	576	1345	8351	1.20
Ubuntu guest	2	511	1238	7718	1.31

(4 VCPU)					
Ubuntu guest (4 VCPU)	4	465	1166	7336	1.40

Although total durations are shorter increasing the number of parallel workers, the durations do not scale linearly. This can be observed in the Average speed-up column showing the ratio between the execution time of a single worker and the corresponding number of parallel workers. Ideally, the speed-up would be equal to the number of parallel workers. Speed-up was cal. Test was conducted on a computer with Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz processor and 16 GB of RAM.

Table 3 shows results on the same three input cases running on a computer with Intel Xeon x5550 @ 2.67GHz processor and 66 GB of RAM. The results are only available for Linux host and are presented in the following table.

Table 3: Execution times on three input cases using different number of workers.

Configuration	**Workers**	**mik3d_15min [s]**	**mik3d_1h [s]**	**mik3d_4h [s]**	**Average speed-up**
Ubuntu host	1	994	2580	16150	-
Ubuntu host	2	520	1340	8390	1.92
Ubuntu host	4	316	809	5066	3.17
Ubuntu host	8	288	729	4401	3.55

Parallelism in this case is much more evident; however the overhead of communication between workers increases significantly. The average speed-up column shows the ratio between execution time of a single worker compared to the corresponding number of parallel workers. Ideally, the speed-up would be the number of parallel workers.

7 Conclusions

The aerodynamic map use case will at the beginning of the project focus mainly on running OpenFOAM in OSv, where the main goal is to run OpenFOAM simulations in parallel. The SCs will later serve as baselines for OSv and sKVM optimisations, graphical user interface design and to tackle security concerns. The HCs will mostly serve as a baseline for the virtualised I/O efficiency improvement.

The bash scripting-based procedure currently used at Pipistrel represents a rather cumbersome workflow for aerodynamic map analysis. Additionally, the number of simultaneously run simulations is limited only to a small number of available computer cores located on the local machine. We intend to upgrade this workflow using cloud infrastructure, which will allow it to deploy a set of simulations more easily and to obtain more results at the same time. The MIKELANGELO project offers Pipistrel an excellent way to achieve this goal. The project will enable Pipistrel to learn how to prepare use cases ready for cloud infrastructure, to gain deeper knowledge of OpenFOAM operation and to gain a large amount of knowledge regarding cloud computing. On the other hand, the aerodynamic use case will deliver a set of baselines used to demonstrate improvements of different MIKELANGELO stack components.

8 References and Applicable Documents

[1] http://www.mikelangelo-project.eu/

[2] http://www.openfoam.org

[3] http://www.pbsworks.com/Product.aspx?id=1

[4] http://www.adaptivecomputing.com/products/open-source/torque/

[5] http://www.theubercloud.com

[6] https://www.docker.com

[7] http://www.open-mpi.org

[8] http://www.mpich.org

[9] http://cfd.direct/openfoam/user-guide/running-applications-parallel/