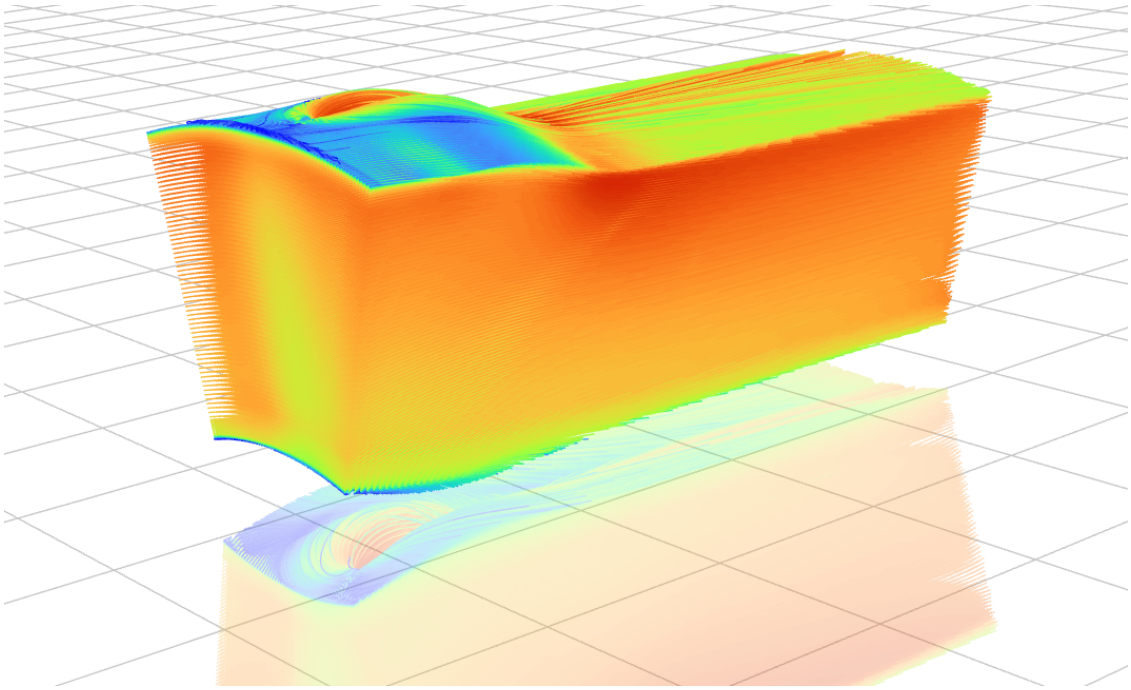

CFD on GPUs in Aerospace Applications

Benchmarking the Fluent native GPU solver on aerospace applications, and how to approach purchasing GPUs for CFD as a business case.

Master's thesis in Mobility Engineering

Filip Gustafsson, Gustav Rönn

DEPARTMENT OF Mechanics and Maritime Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

CFD on GPUs in Aerospace Applications

An evaluation of Ansys Fluent native GPU solver in aerospace applications, and GPUs for CFD simulations as an investment.

Filip Gustafsson, Gustav Rönn

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Fluid Dynamics
Chalmers University of Technology
Gothenburg, Sweden 2025

CFD on GPUs in Aerospace Applications
An evaluation of Ansys Fluent native GPU solver in aerospace applications, and GPUs for CFD simulations as an investment.
Filip Gustafsson, Gustav Rönn

Supervisors: Björn Bragée, EDRMedeso
Jan Östlund, GKN Aerospace
Examiner: Professor Lars Davidsson, Department of Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Division of Fluid Dynamics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Simulation with pathline visualization on the TRS case from Ansys Fluent 2025 R1.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

CFD on GPUs in Aerospace Applications
An evaluation of Ansys Fluent native GPU solver in aerospace applications, and GPUs for CFD simulations as an investment.
Filip Gustafsson, Gustav Rönn
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Running CFD simulations on GPUs is becoming commercially viable, and one major early adopter is the automotive industry, where external aerodynamics cases can be run at a fraction of the time compared to simulations on CPUs. The aerospace industry has not yet adopted GPUs to the same extent, as aerospace cases often require support for more complex physics such as compressible flows and combustion. This study compares the performance of the Ansys Fluent GPU solver with the CPU solver in aerospace applications and is carried out with the support of GKN Aerospace and EDR & Medeso, a reseller of Ansys software. It uses a novel approach to evaluate the attractiveness of purchasing GPUs for a local cluster, compared to purchasing CPUs, from a cost, power consumption, and strategic perspective. A case-based methodology is used to compare the solvers with 3 simulation setups that are representative of typical aerospace applications. The current version of the GPU solver supports all the necessary features to run 2/3 cases, although it requires minor simplifications to the case setups. For the cases it does support, key results include GPU simulations providing a time reduction of 41-98% per iteration, an energy consumption reduction of 88-93% per iteration, a 27-73% reduction in iterations to reach convergence, a cloud computing cost reduction of 83-91% and a total cost of ownership reduction of 48-67% for systems with equivalent simulation capacity on a local cluster. If the simulation capacity demand for simulation setups that the GPU solver supports is sufficient, purchasing GPUs for CFD simulation is a cost-effective and energy-efficient solution to meet simulation capacity demands in comparison to purchasing CPUs. The speedups provided by the Ansys Fluent GPU solver can be leveraged to generate significant value in an engineering process by enabling more design iterations, improved simulation fidelity, and faster simulation turnaround, compared to the CPU solver.

Keywords: CFD, GPU, CPU, HPC, Ansys Fluent, Native GPU Solver, Simulation, Aerospace, Business case, Turbomachinery.

Acknowledgements

We would like to thank EDR & Medeso, GKN Aerospace and the people who have supported us throughout our master's thesis.

Our supervisor, Björn Bragée, for his interest in the project, great discussions, and continuous support. The rest of the staff at EDR & Medeso, especially Klas Johansson and Tomas Jarneholt for their wide CFD knowledge and willingness to always answer all our questions.

Our other supervisor, Jan Östlund at GKN Aerospace, for continuously supporting us in interpreting simulation results and evaluating setups.

Our examiner, Professor Lars Davidson.

Others that have contributed with their knowledge:
Pekka Wikman, Marcus Lejon and Tomas Fernström, GKN Aerospace.

Anders Jönsson, Tobias Berg and Didier Besette, Ansys.

Torbjörn Wirdung, Volvo Cars.

Adam Koc and Jan Wallenberg, GoVirtual.

Lastly we want to thank our friends and families for their ongoing support throughout our entire studies.

Filip Gustafsson, Gustav Rönn, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ALU	Algorithmic Logic Unit
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CSR	Compressed Sparse Row format
FLOPS	Floating Point Operations per Second
CUDA	Compute Unified Design Architecture
FP32	32-bit Floating Point number
FP64	64-bit Floating Point number
GPU	Graphics Processing Unit
HPC	High Performance Computing
IC	Integrated Circuit
SIMD	Single Instruction, Multiple Data
SIMT	Single Instruction, Multiple Threads
SMP	Streaming Multi-Processor
SP	Streaming Processor
RAM	Random Access Memory
RANS	Reynolds-Averaged Navier-Stokes equations
TDP	Thermal Design Power
VRAM	Video Random Access Memory

Nomenclature

Below is the nomenclature of parameters and variables that have been used throughout this thesis.

A	Area
A_f	Area of a specific face, f
a	Acceleration
C	Cost
C_P0	Total pressure coefficient
CV	Control volume
$\hat{\mathbf{e}}_i$	General unit vector
E_{tot}	Total energy consumption
f_{clock}	Clock frequency
f_{memory}	Memory frequency
F_{thrust}	Thrust force
g	Gravitational constant = 9.81
$\hat{\mathbf{i}}$	Unit vector in x
Isp	Specific impulse
Isp_{vac}	Specific impulse in a Vacuum environment
k	Turbulent kinetic energy
$k_{cooling}$	Cooling factor
L	Length
M	Momentum
$\dot{m}$	Mass flow
$\dot{m}_{corr}$	Corrected mass flow
n	Number of
$\hat{\mathbf{n}}$	Normal vector
P	Pressure & Power
P_d	Dynamic pressure

P_{ratio}	Pressure ratio over rotor stage
P_{ref}	Reference or ambient pressure
P_s	Static pressure
P_t	Total pressure
P_{tot}	Total power consumption
p	Price
Re	Reynolds number
S_i	Internal energy source term
S_M	Momentum source term
t_{sas}	Start and shutdown time
t_{sim}	Simulation time
t_{tot}	Total time
T	Temperature
T_{ref}	Reference or ambient temperature
T_t	Total temperature
ts	Time step
u	Velocity in x-direction
U	Velocity vector
v	Velocity in y-direction
V	Volume
V_P	Volume
w	Velocity in z-direction
β	Bandwidth
Γ	Diffusion coefficient
μ	Dynamic viscosity
ϕ	General fluid property vector
Φ	Dissipation function
ρ	Density
τ	Shear stress
υ	Kinematic viscosity

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When complex geometries are introduced into fluid mechanics problems, analytical solutions are not feasible. The problems are instead solved numerically using Computational Fluid Dynamics (CFD), by discretizing the governing continuity and Navier-Stokes equations into a mesh of control volumes. An early paper on a 3D model was published in 1967 by John Hess and A.M.O. Smith [1]. Between then and now there has been major developments in the modeling of fluid flows, including turbulence models, discretization schemes, meshing developments, and multi-physics simulations. CFD codes have traditionally been written for the Central Processing Unit (CPU) of the computer, designed to execute instructions sequentially. Single Instruction, Multiple Data (SIMD) Graphics Processing Units (GPUs) were commercialized in the late 1990s and their performance has been increasing at a high rate. Today, server GPUs have far surpassed the server CPUs in potential computing power and the gap continues to grow larger. For example, the Nvidia H100 GPU has a 40x larger theoretical maximum number of 32-bit floating point operations per second (Flops) compared to the 32-core Intel Xeon Gold 6455B CPU, both of which are used in this thesis. In Figure 1.1 the theoretical performance of some CPUs and GPUs from the last 10 years are shown.

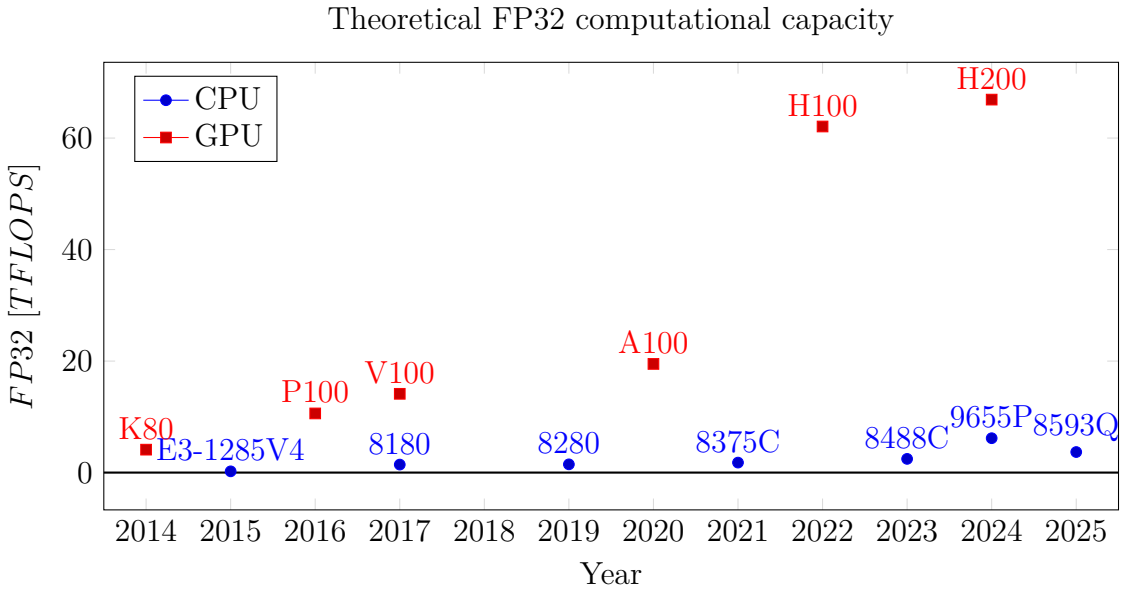

Figure 1.1: Maximum FP32 computational performance measured in TFlops for some CPUs and GPUs throughout the years.

The theoretical maximum GPU computing performance can only be reached if all processing cores are fully utilized, which is not realistic in CFD codes. Read and write operations create a major computational cost in CFD codes and the computational time is therefore normally bottle-necked by the memory bandwidth, β, when solving large CFD problems. The memory bandwidth of some CPUs and GPUs from the last decade can be found in Figure 1.2.

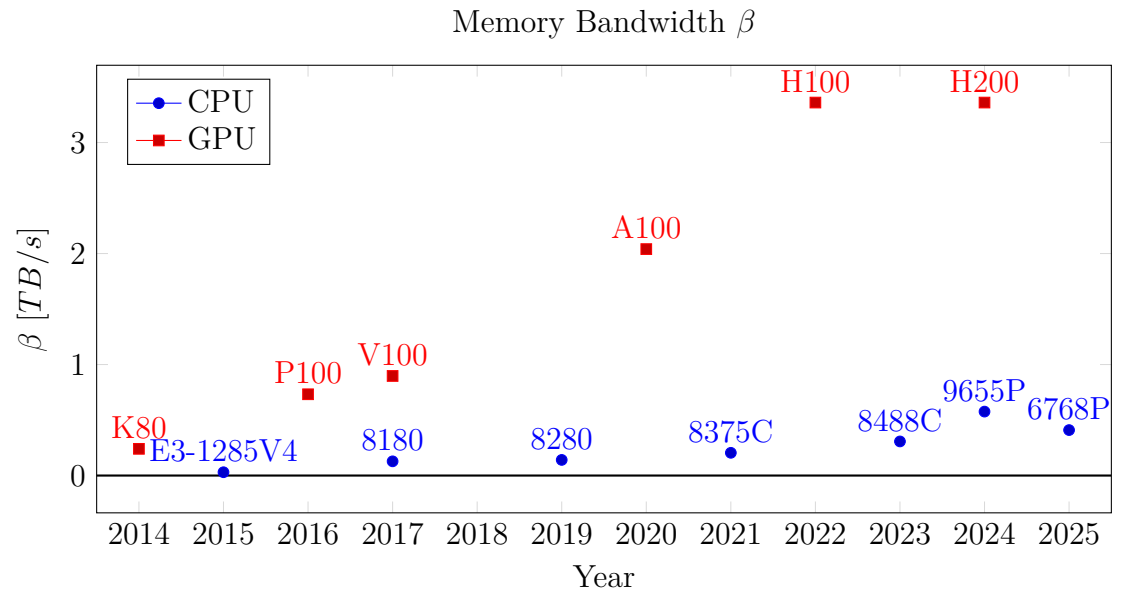

Figure 1.2: Memory bandwidth β of some CPUs and GPUs throughout the years.

GPUs for CFD simulations can be considered a disruptive technology in the fluid

simulation field, showing speedups by up to 32x in the Ansys Fluent GPU solver when running on multiple GPUs, driving automotive manufacturers to invest millions of Euros in GPUs for aerodynamics simulations.[2] [3] The investment in GPUs for CFD is motivated by the strive to optimize engineer and researcher time, since skilled personnel can account for 5-10x the cost compared to the next biggest expense which is normally computing hardware or software licenses. [4] Faster simulations allow engineers to incorporate more simulations earlier in the design process. [4]

EDR & Medeso is supervising this master thesis together with GKN Aerospace to evaluate the Fluent GPU and CPU solvers' performance in aerospace applications and whether an investment in GPUs would be cost-efficient and beneficial for their cluster. EDR & Medeso AB is a reseller of calculation and simulation software for engineers and they deliver training, support and consulting services in this area. Their software suite consists of products from Ansys, Rescale, Neural Concept, ScaleX Enterprise, Trimble, IDEA StatiCa and CSI. They operate in the Nordics, the UK, and Poland, and have around 160 employees worldwide, with about 40 based in Sweden. [5][6] GKN Aerospace Sweden AB manufactures and sells engine components for airplanes and rockets, and performs engine maintenance. They are based in Trollhättan and part of the GKN Aerospace Group based in the UK. They produce engine components for civil aircraft, the RM12 engine for SAAB 39 Gripen and rocket nozzles and turbines for the Ariane 6 rocket. [7]

In the ANSYS Fluent suite, a solver that was natively written for running on GPUs was first included in the 2023 R1 release. The solver is continuously updated for each release that comes out. The current release of the GPU solver is limited in its functionality compared to the CPU solver. The main differences between the CPU and GPU solver are listed in the appendix, in Table A.1. In this project, the 2025 R1 release of ANSYS Fluent is used.
The available literature on CFD on GPUs is limited, and in the case of aerospace applications, non-existent. This is the case since the field itself is rather new, ANSYS rolled out their GPU solver in 2020 to customers [8].
The GPU-accelerated Siemens Star-CCM+ solver shows a 20x speedup for an external automotive aerodynamics case when comparing 2x 64-core AMD EPYC 7742 CPUs with 8x Nvidia A100 GPUs, while the Ansys Fluent Native GPU solver shows a 30x speedup for a different external automotive aerodynamics case with the same CPUs and GPUs. [4] [8] In non-automotive, time reductions in the range 5x to 20x depending on the case and GPU. [9] These were smaller cases with 11 & 16 million cells, respectively.

Power consumption can also be reduced by moving from a CPU server to a GPU server. A 4x reduction in power consumption was found when comparing 64x 16-core Intel Xeon Gold 6242 with 6x Nvidia V100 GPUs without taking cooling, simulation time and the CPU in the GPU server into account. [8] When taking simulation time into account, the results become case-dependent, ranging from a factor of 0.67x to 8x. [9] It should be noted that this study accounts for the necessity of a CPU in every GPU server node but assumes that the CPU runs with maximum power

consumption throughout the whole simulation, rather than idle power consumption, overestimating the GPU node power consumption in comparison to the CPU node power consumption.

The accuracy of the Ansys Fluent Native GPU solver has been studied on a laminar flow over a sphere and it was found to compute the drag coefficient to an error of 0.252%. [8] It has been shown to correlate well with experimental and CPU solver results on a further set of cases. [10] [11]

The cost of CFD simulations on GPUs has also been previously studied with cost reductions of up to 83%. [9] Volvo Cars reached a 2.5x speedup when running on 8x Blackwell GPUs compared to cost-equivalent CPUs with a total of 2016 cores. [2]

Chapter 2

Theory

To understand how GPUs can solve CFD problems faster than CPUs, it is necessary to dive into the calculations that a CFD code performs, the architecture of computers, and how they relate to each other. This is complemented with some background theory on business case methods.

2.1 Fluid theory

The field of fluid mechanics is vast and a commercial CFD code is a ridiculously complex software with a multitude of features and solutions for different problems. The main solution procedure is similar for most codes, however, and the goal of this chapter is to explain how the main calculation procedure is derived and why it can be sped up through parallelization.

2.1.1 Basic fluid mechanics and governing equations

To start off, the characteristics of a flow depends on its velocity U. A Reynolds number $Re > 4000$ indicates that a flow is fully turbulent [12] and a Mach number $M \geq 1$ indicates that compressibility effects are important. [13]

$$Re = \frac{\rho U L}{\mu} \tag{2.1}$$

$$M = \frac{U}{a} \tag{2.2}$$

The governing equations of the flow of a compressible Newtonian fluid can be written as shown in (2.3), (2.4), (2.5), (2.6), and (2.7). [14]
Continuity equation:

$$\frac{\partial \rho}{\partial t} + \text{div}(\rho \mathbf{u}) = 0 \tag{2.3}$$

Navier-Stokes equations, describing momentum in the x-, y-, and z-components:

$$\frac{\partial(\rho u)}{\partial t} + \text{div}(\rho u \mathbf{u}) = -\frac{\partial p}{\partial x} + \text{div}(\mu \text{ grad } u) + S_{M_x} \tag{2.4}$$

$$\frac{\partial(\rho v)}{\partial t} + \text{div}(\rho v \mathbf{u}) = -\frac{\partial p}{\partial y} + \text{div}(\mu \text{ grad } v) + S_{M_y} \tag{2.5}$$

$$\frac{\partial(\rho w)}{\partial t} + \text{div}(\rho w \mathbf{u}) = -\frac{\partial p}{\partial z} + \text{div}(\mu \text{ grad } w) + S_{M_z} \tag{2.6}$$

where S_M is the momentum source term.
Energy equation:

$$\frac{\partial(\rho i)}{\partial t} + \text{div}(\rho i \mathbf{u}) = -p \text{ div } \mathbf{u} + \text{div}(k \text{ grad } T) + \Phi + S_i \tag{2.7}$$

where Φ is the dissipation function and S_i is the source term. The time derivative terms are not considered when solving in steady-state.

2.1.2 Discretization

The governing equations are discretized over a control volume by applying the Gauss divergence theorem. Discretizing the transport equation, which is a common differential form for all flow equations, yields the following steady-state integrated form where ϕ is defined as a general variable representing a fluid property, $\hat{\mathbf{n}}$ is the normal vector and S_ϕ is the source term for variable ϕ. [14]

$$\int_A \hat{\mathbf{n}}(\rho\phi\mathbf{u})\mathrm{d}A = \int_A \hat{\mathbf{n}}(\Gamma \text{grad}\phi)\mathrm{d}A + \int_{CV} S_\phi \mathrm{d}V \tag{2.8}$$

The physical interpretation of this form is that the divergence integrated over a control volume equals the fluxes over its surfaces. Splitting over a finite number of faces yields:

$$\sum_f^M \int_{A_f} \hat{\mathbf{n}}_f(\rho\phi\mathbf{u})\mathrm{d}A_f = \sum_f^M \int_{A_f} \hat{\mathbf{n}}_f(\Gamma \text{ grad } \phi)\mathrm{d}A_f + \int_{CV} S_\phi \mathrm{d}V \tag{2.9}$$

Assuming that the fluid property varies linearly across each face allows the surface integral to be approximated to the value at the point on the face that is between the node and the neighboring node.

$$\sum_f^M \hat{\mathbf{n}}_f \rho_f \phi_f \mathbf{u}_f A_f = \sum_f^M \hat{\mathbf{n}}_f (\Gamma \text{grad}\phi)_f A_f + \int_{CV} S_\phi \mathrm{d}V \tag{2.10}$$

The diffusion term is related to the orthogonality of the mesh. It is divided into an implicit and an explicit term, where the implicit term depends on the orthogonal component Δ of the vector d and the explicit term depends on the non-orthogonal component k of the vector d.

$$\sum_{f=1}^M \hat{\mathbf{n}}_f (\Gamma grad\phi)_f A_f = \sum_{f=1}^M (\rho\Gamma_\phi)_f \Delta(\text{grad}\phi)_f + \sum_{f=1}^M (\rho\Gamma_\phi)_f k(\text{grad}\phi)_f \tag{2.11}$$

This can be discretized through the "minimum correction" approach, the "orthogonal correction" approach or the "over-relaxed" approach. Here is the orthogonal

correction approach.

$$\sum_{f=1}^{M}(\rho\Gamma_\phi)_f|\hat{\mathbf{n}}_f A_f|d\frac{\phi_N-\phi_P}{|d|^2}+\sum_{f=1}^{M}(\rho\Gamma_\phi)_f\hat{\mathbf{n}}_f A_f(1-\frac{d}{|d|})(f_x(\text{grad}\phi)_P+(1-f_x)(\text{grad}\phi)_N) \tag{2.12}$$

$(1-\frac{d}{|d|})$ is zero for an orthogonal mesh, taking the second term out of the equation. This can be applied to all 5 governing equations.
Continuity equation:

$$\int_A \hat{\mathbf{n}}(\rho\mathbf{U})\mathrm{d}A+\int_A \hat{\mathbf{n}}.(\rho\mathbf{V})\mathrm{d}A+\int_A \hat{\mathbf{n}}.(\rho\mathbf{W})\mathrm{d}A=0 \tag{2.13}$$

$$\sum_{f}^{M}\hat{\mathbf{n}}_f\rho_f\mathbf{U}_f A_f+\sum_{f}^{M}\hat{\mathbf{n}}_f\rho_f\mathbf{V}_f A_f+\sum_{f}^{M}\hat{\mathbf{n}}_f\rho_f\mathbf{W}_f A_f=0 \tag{2.14}$$

X-momentum:

$$\int_A \hat{\mathbf{n}}(\rho uu-\mu\frac{\partial u}{\partial x})\mathrm{d}A+\int_A \hat{\mathbf{n}}(\rho vu-\mu\frac{\partial u}{\partial y})\mathrm{d}A+\int_A \hat{\mathbf{n}}(\rho zu-\mu\frac{\partial u}{\partial z})\mathrm{d}A=$$
$$\int_{\Delta V}-\frac{\partial p}{\partial x}\mathrm{d}V+\int_{CV}S_{M_x}\mathrm{d}V \tag{2.15}$$

$$\sum_{f=1}^{M}\hat{\mathbf{n}}_f(\rho_f u_f u_f-\mu_f(\frac{\partial u}{\partial x})_f)A_f+\sum_{f=1}^{M}\hat{\mathbf{n}}_f(\rho_f v_f u_f-\mu_f(\frac{\partial u}{\partial y})_f)A_f+ \tag{2.16}$$
$$\sum_{f=1}^{M}\hat{\mathbf{n}}_f(\rho_f z_f u_f-\mu_f(\frac{\partial u}{\partial z})_f)A_f=-(\frac{\partial p}{\partial x})V_P+S_U V_P+S_P V_P u_P$$

The energy equation and the Y and Z momentum equations are discretized in a similar manner.
The discretized equations are then split into coefficients. For every node P, there are nb neighbours. The value at a face f between a node P and its neighbour node N is calculated through linear interpolation where f_x is the distance from node P to face f.

$$\phi_f=\phi_P(1-f_x)+\phi_N f_x \tag{2.17}$$

Solving the equation system for u gives:

$$a_P u_P=\sum_{nb}a_{nb}u_{nb}+\sum p_f A\cdot\hat{\mathbf{i}}+S \tag{2.18}$$

For generalized fluid property ϕ:

$$a_P\phi_P=\sum_{nb}a_{nb}\phi_{nb}+\sum p_f A\hat{\mathbf{e}}_i+S \tag{2.19}$$

This creates a system of algebraic equations.

$$A\phi=B \tag{2.20}$$

where A is a sparse matrix with a_P on the diagonal and a_{nb} on the off-diagonal corresponding to the neighbouring cell, ϕ is a vector containing a general fluid property (for example x-velocity or pressure) and B is a vector containing source terms. The size of the A matrix will be the square of the number of cells in the domain. The number of a_{nb} coefficients will depend on the number of faces a given cell has. A cubic cell, for instance, has six faces (and six directly neighbouring cells) and therefore has six corresponding a_{nb} coefficients in the A matrix. [15]

Size of matrix A:

$$A_{size} = n_{cells}^2 \tag{2.21}$$

Number of coefficients in matrix A:

$$A_{coeff} = n_{cells}(1 + \bar{n}_{faces\ per\ cell}) \tag{2.22}$$

The number of faces per cell is defined by the number of face connections each face has, which means that a face that connects to cells will have one coefficient for each cell. The size of matrix A will grow faster than its coefficients when the mesh has a large number of cells.

2.1.3 Coupled pressure-based solver

In the pressure-based solver, a flow problem can be solved using a segregated or a coupled approach. The segregated approach solves each momentum and pressure correction equation separately, while the coupled approach solves all equations simultaneously in one matrix equation.

The structure of matrix A and vectors ϕ and B for general $A\phi = B$ in a segregated solver is as follows:

$$A = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \dots & A_{mn} \end{bmatrix}, \phi = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_n \end{bmatrix}, B = \begin{bmatrix} b_1^\phi \\ \vdots \\ b_m^\phi \end{bmatrix}$$

The sparsity of the A matrix increases with an increasing number of cells, since every cell is only connected to its direct neighbours. Examples for 2D 2x2 and 3x3 equidistant meshes are attached in Appendix B. The sparsity of a sparse matrix can be defined by:

$$\frac{n_{cells} - n_{nnz}}{n_{cells}} \tag{2.23}$$

where n_{nnz} is the number of non-zero elements in the matrix. In a coupled solver for a 3D case the A matrix contains all three momentum equations, pressure correction terms, pressure-velocity coupling terms, and the matrix is laid out in the following way:

$$A = \begin{bmatrix} A_{11} & \dots & A_{1n} & 0 & \dots & 0 & 0 & \dots & 0 & -a_{11} & \dots & -a_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{m1} & \dots & A_{mn} & 0 & \dots & 0 & 0 & \dots & 0 & -a_{m1} & \dots & -a_{mn} \\ 0 & \dots & 0 & A_{11} & \dots & A_{1n} & 0 & \dots & 0 & -b_{11} & \dots & -b_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & A_{m1} & \dots & A_{mn} & 0 & \dots & 0 & -b_{m1} & \dots & -b_{mn} \\ 0 & \dots & 0 & 0 & \dots & 0 & A_{11} & \dots & A_{1n} & -c_{11} & \dots & -c_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & A_{m1} & \dots & A_{mn} & -c_{m1} & \dots & -c_{mn} \\ -d_{11} & \dots & -d_{1n} & -e_{11} & \dots & -e_{1n} & -f_{11} & \dots & -f_{1n} & A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -d_{m1} & \dots & -d_{mn} & -e_{m1} & \dots & -e_{mn} & -f_{m1} & \dots & -f_{mn} & A_{m1} & \dots & A_{mn} \end{bmatrix}$$

with vector ϕ $\begin{bmatrix} u_1 \\ \vdots \\ u_m \\ v_1 \\ \vdots \\ v_m \\ z_1 \\ \vdots \\ z_m \\ p_1 \\ \vdots \\ p_m \end{bmatrix}$ and vector B $\begin{bmatrix} B_1^U \\ \vdots \\ B_m^U \\ B_1^V \\ \vdots \\ B_m^V \\ B_1^Z \\ \vdots \\ B_m^Z \\ B_1^P \\ \vdots \\ B_m^P \end{bmatrix}$

The blocks in the A matrix for a coupled solver can be summarized as: [16]

$$\sum_{ij} A_P = \begin{bmatrix} a_{ij}^{uu} & a_{ij}^{uv} & a_{ij}^{uw} & a_{ij}^{up} \\ a_{ij}^{vu} & a_{ij}^{vv} & a_{ij}^{vw} & a_{ij}^{vp} \\ a_{ij}^{wu} & a_{ij}^{wv} & a_{ij}^{ww} & a_{ij}^{wp} \\ a_{ij}^{pu} & a_{ij}^{pv} & a_{ij}^{pw} & a_{ij}^{pp} \end{bmatrix}$$

for cell i and neighbor connection j. a_{ij}^{uv},a_{ij}^{uw},a_{ij}^{vu},a_{ij}^{vw},a_{ij}^{wu},a_{ij}^{wv} contains terms only for cells that have direct contact to boundaries, which is assumed to become computationally negligible on mesh sizes with a number of cells in the order of millions.

An obvious conclusion when looking at the efficiency of the $A\phi = B$ equation for sparse matrices is that it is inefficient to multiply all these zeros together. One way of working around this is the Compressed Sparse Row (CSR) format. This transforms the sparse matrix into three vectors: one to store row indices (or row pointers), one to store column indices and one to store values. This effectively reduces the size of an A matrix block to: [17]

$$A_{blocksize} = n_{nnz} = n_{cells}(1 + n_{nb}) \tag{2.24}$$

with the row and column indices vectors being larger than n_{cells} and smaller than $n_{cells}(1 + n_{nb})$ depending on the sparse format used. For a non-adaptive mesh the

row and column indices are constant and therefore only need to be read from memory the first iteration.

2.1.4 Turbulence Models and Species transport

Turbulence is an unsteady flow phenomenon. To model turbulent regions in steady-state, the flow properties are approximated using a turbulence model, which is commonly done by Reynolds-averaging the Navier-Stokes equations (RANS). The most relevant models are the two equation linear eddy viscosity models $k-\omega$ and $k-\epsilon$, with the transported variables turbulent kinetic energy k, specific turbulent dissipation rate ω and dissipation rate ϵ.

2.1.4.1 $k-\omega$ SST

The $k-\omega$ baseline (BSL) turbulence model is based on the observation that the $k-\omega$ model is accurate in adverse pressure gradients and that the $k-\epsilon$ model is accurate in freestream flow. The $k-\epsilon$ model is rewritten as a $k-\omega$ formulation and a switching function is used to determine which model to use. The BSL model is further improved into the $k-\omega$ shear-stress transport (SST) model by assuming that the principal shear-stress is proportional to the turbulent kinetic energy in the boundary layer. This allows it to account for the transport of the turbulent shear stress. [18]
BSL formulation:

$$\frac{\partial \rho k}{\partial t} + \frac{\partial \rho u_j k}{\partial x_j} = P_k - \beta^* \rho \omega k + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_k \mu_t)\frac{\partial k}{\partial x_j}\right] \tag{2.25}$$

$$\frac{\partial \rho \omega}{\partial t} + \frac{\partial \rho u_j \omega}{\partial x_j} = \gamma P_\omega - \beta \rho \omega^2 + 2\rho(1-F_1)\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_\omega \mu_t)\frac{\partial \omega}{\partial x_j}\right] \tag{2.26}$$

where:

$$F_1 = tanh((max(min(\frac{\sqrt{k}}{0.09\omega y}; 0.45\frac{\omega}{\Omega}); \frac{400\nu}{y^2\omega}))^4) \tag{2.27}$$

SST term:

$$\frac{D\tau}{Dt} = \frac{\partial \tau}{\partial t} + u_k \frac{\partial \tau}{\partial x_k} \tag{2.28}$$

2.1.5 Definitions of variables, coefficients and expressions

$$F_{thrust_{vac}} = v_{exhaust}\ \dot{m} + \int_{A_{outlet}} P_{S_{outlet}} \mathrm{d}A \tag{2.29}$$

$$isp = \frac{F_{thrust}}{\frac{\dot{m}}{g}} \tag{2.30}$$

$$isp_{vac} = \frac{F_{thrust_{vac}}}{\frac{\dot{m}}{g}} \tag{2.31}$$

$$P_{ratio} = \frac{P_{t2}}{P_{t1}} \tag{2.32}$$

$$\dot{m}_{corr} = \dot{m}\frac{\sqrt{T_{t0}/T_{ref}}}{P_{t0}/P_{ref}} \tag{2.33}$$

where T_{t0},P_{t0} is the total temperature and total pressure at the inlet and $T_{ref} = 288.15\ K$ and $P_{ref} = 101325\ Pa$

2.1.5.1 Coefficients

Total pressure coefficient

$$C_{p0} = \frac{P_t - P_{t,\ ref}}{P_{t,\ ref} - P_{s,\ ref}} \tag{2.34}$$

Pressure coefficient

$$C_p = \frac{P_s - P_{s,\ ref}}{P_{t,\ ref} - P_{s,\ ref}} \tag{2.35}$$

2.2 Computer Architecture

Computers contain many components, including a motherboard, processing units, memory, long-term storage, a power supply, and input/output systems. This section focuses on the components and functions that are most relevant for scientific computing applications such as CFD.

2.2.1 Floating point numbers

Floating point binary number formats are standardized through IEEE 754. They are divided into the sign, the exponent and the significand (or mantissa) and are categorized according to the number of bits they utilize. [19]

2.2.1.1 Single-precision numbers

A single-precision floating point number FP32 contains 32 bits. These bits are divided into 1 sign bit, 7 exponent bits, and 24 significand bits. The single-precision floating point is accurate to between 6-9 significant decimals.[19] This creates a risk of inaccuracy when executing operations if many significant decimals are important for the accuracy of the results. Below, in Figure 2.1, an arbitrary number is shown in IEEE 754 Floating point format.

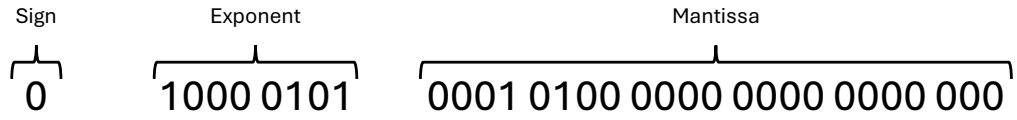

Figure 2.1: An arbitrary number represented in IEEE 754 Floating point format.

2.2.1.2 Double-precision numbers

A double-precision floating point number FP64 contains 64 bits. These bits are divided into 1 sign bit, 10 exponent bits and 53 significand bits. The double-precision floating point format is accurate to between 15-17 significant digits. [19]

2.2.1.3 Floating point numbers in CFD

To exemplify why this is relevant in CFD, consider how the total pressure P_t and static pressure P_S are related to the dynamic pressure P_D.

$$P_D = P_t - P_S \tag{2.36}$$

If $P_t = 100095.73$ and $P_S = 99701.27$ the resulting FP32 difference is 394.453 while the FP64 difference is 394.460. [20] This inaccuracy can cause incorrect results in certain cases.

2.2.2 Central Processing Unit

The central processing unit (CPU) fetches, decodes, and executes instructions that are retrieved from a program sequentially. They can differ in architecture, but most contain some core features: an Algorithmic Logic Unit (ALU), registers, a control unit, and a bus that connect them. A simple machine architecture is the MARIE computer, which contains a CPU and a main memory. The CPU in the MARIE architecture contains an ALU, an accumulator, a memory bus register, a memory address register, an input register, an output register, an instruction register and a program counter. The ALU performs operations such as addition and multiplication. The operations are defined by physical circuits, and every CPU architecture has a set of operations that can be performed. Registers hold input and output instructions, instructions to execute and memory addresses. The control unit contains an instruction register and program counter, and initiates the next cycle loop which depends on what instruction it stores.
The time it takes for a single-core CPU without hyper-threading to run a program can be simplified to:

$$\frac{t}{program} = \frac{instructions}{program} * \frac{avg.cycles}{instruction} * \frac{t}{cycle} \tag{2.37}$$

where $\frac{t}{cycle}$ is the clock-speed of the CPU. Most cycles reads or writes instructions from the main memory, and the memory bus bandwidth is therefore also a factor in CPU performance.

$$FP32_{CPU}\ [FLOPS] = 8 \cdot 2 f_{clock} n_{cores} \tag{2.38}$$

2.2.2.1 Processing cores

Multi-core processors have multiple CPUs on one integrated circuit (IC). They utilize multithreading, which is a parallelization technique that, in essence, divides the instructions from a program and places them into different containers. The instructions in each container are executed independently with local variables and synchronized globally when necessary.

2.2.3 Memory

Memory bandwidth describes the rate of which data can be read and written to the Random Access Memory (RAM) of a computer. [21] Insufficient memory bandwidth

can cause bottlenecks in a system, as the memory cannot write or read the data from the processing unit at a sufficient rate. Insufficient bandwidth leads to the cores being unutilized and remaining at idle, as the memory read and write cannot keep up with the speed of the cores. This is calculated as described in (2.39).

$$\beta\ [B/s] = 8 f_{memory} n_{channels} \tag{2.39}$$

2.2.4 Graphics Processing Unit

A Graphics Processing Unit (GPU) has thousands of small cores, called streaming processors (SPs) or Compute Unified Design Architecture (CUDA) cores. The SPs are organized into streaming multi-processors (SMPs) which batch them together in sets that share fast cache memory and can synchronize. It is optimized to execute many computations in parallel to maximize total throughput. As with multi-core CPUs, GPUs utilize multithreading, but take it one step further. In GPUs, multithreading is implemented in the hardware and applied in a single instruction, multiple threads (SIMT) model. SIMT is an execution model that executes one instruction on multiple data using multithreading for parallel execution. A great example is large matrix operations where one instruction is applied to transform a large dataset, which is common in graphics applications, AI training algorithms, and in CFD solvers.

$$FP32_{GPU}\ [FLOPS] = 2 f_{clock} n_{cores} \tag{2.40}$$

GPUs have dedicated Video Random Access Memory (VRAM) built in. The amount of VRAM available determines how much data can be stored simultaneously, and the memory bandwidth determines how fast data can be transferred between the VRAM and the streaming processors. If the amount of data exceeds the available VRAM space, a memory overload occurs and a fatal error is induced.

2.2.4.1 Utilization

To maximize the utility of GPUs, the computation tasks should have a low dependency on each other, low synchronization requirements and the processes should be oversubscribed, i.e. parallelized in excess of the inherent parallel capabilities of the GPU. Oversubscription is beneficial because threads that are not ready to run can be set aside while fetching data and switched to a ready-to-run thread that can be executed immediately, maximizing core utilization. [22] [23] In the case of the Fluent GPU solver, additional parameters need to be taken into account. When running a case on multiple GPUs, the case should have a minimum of 2 million cells per GPU. To avoid memory overload, all matrices need to fit into the memory simultaneously, and a rule of thumb is that 1 GB VRAM is required per 1 million cells for a hexahedral mesh, with single-precision accuracy. Double-precision requires 50% more memory and other settings such as polyhedra meshes, flow scheme and AMG solver aggregation type will also affect the memory required. [24]

2.2.5 High Performance Computing

Aggregating computational resources by grouping individual computers into a cluster or building a supercomputer is used to maximize computational capabilities.

2.2.5.1 Power Consumption

The power consumption of a HPC system is dependent on the components in the system, the cooling design, and other factors. It can roughly be estimated as:

$$P_{tot}\ [W] = \sum P_{component} + \sum P_{cooling} \tag{2.41}$$

Although CPU temperature correlates closely with system power consumption in most applications, memory-intensive applications such as CFD can have a significantly higher power consumption than other applications with similar CPU temperatures. Therefore, the intensity of RAM operations shows a relation with power consumption in memory-intensive applications. This is not the case for applications that are not memory-intensive.[25]

The total global data center electricity demand for 2022 has been estimated to 240-340 TWh, accounting for 1-1.3% of the total global electricity demand for 2022 of 26 600 TWh. [26] The total primary energy consumption worldwide 2022 was 168 500 TWh [27], therefore the total global data center electricity demand for 2022 accounted for 0.14-0.20% of the total global energy demand. HPC energy use is currently increasing by 20-40% annually. [26] As computational demands are projected to continue increasing, energy efficiency in HPC systems becomes important to limit their environmental impact and electricity demand. GPUs provide an opportunity to perform computationally expensive calculations for a fraction of the energy consumption per computation of CPUs, and are thus an alternative with great potential to reduce the energy demand of HPC systems. [26] It should be noted that while HPC consumes energy, it also drives energy efficiency, renewable energy development, and resource optimization in many fields, including wind turbines, vehicle aerodynamics, and airplane design.

2.3 Business Case

To evaluate an investment as a business case, many factors need to be taken into account. These include costs, benefits, risks, environmental aspects, competitors and other factors.

2.3.1 Cost-Benefit Analysis

The goal of a Cost-Benefit Analysis is to compare an investment with other potential investments, usually presented as a Net Present Value (NPV) of the investment. The comparison aspect is evaluated using a discount rate, which is the expected rate of return on comparable investments. There are different methods for calculating the discount rate. These include the capital asset pricing model (CAPM), the build-up method and the Fama-French three-factor model.[28] In the context of HPC

CAE applications, these do not make much sense to use, as the benefits of an HPC investment are hardly quantifiable. It may instead make more sense to divide the cost-benefit analysis into a Total Cost of Ownership (TCO) analysis and then treat the benefits in the discussion, preferably set in context with a comparison of another similar investment. A Pugh matrix is one way to present the potential benefits objectively. [29]

2.3.1.1 Total cost of ownership

Total Cost of Ownership (TCO) can be divided into acquisition costs, operational costs, end-of-life costs, and indirect costs. [30] Acquisition costs are typically one-time capital expenditures, with the most significant being initial purchase prices, delivery and shipping fees, and installation costs. Operational costs are operational expenses that include maintenance and repairs, energy consumption, and labor costs. End-of-life costs are disposal fees and residual value. Indirect costs include training costs, downtime or lost productivity, and compliance or regulatory costs.

A TCO analysis helps to discern between capital expenditures and operational expenditures. In some cases, it can be beneficial to pay the initial purchase price of the asset upfront, for example, if the residual value of the asset is high. If the asset on the other hand depreciates in value quickly, it may instead make more sense to transfer it into an operating expense by leasing it.

In the case of HPC applications, ownership models can be divided into three main types:

- On-premise infrastructure financed through capital expenditures
- On-premise infrastructure financed through operating expenses (i.e. leasing)
- Cloud computing financed through operating expenses, which can be set to vary on-demand

The expected life span of on-premise HPC systems are 3-6 years for >80% of systems. [31]

Chapter 3

Methods

A case-based methodology is used to evaluate the GPU solver performance and capabilities in typical aerospace simulation scenarios. Multiple simulations were run using various types of hardware to differentiate and evaluate the performance of the different solvers. In this chapter, these hardware configurations and any changes to the physical models will be brought up. Furthermore, the results of the performance evaluations are used as input in the business case analysis.

3.1 Hardware

The simulations were run on a cluster called Rescale. The hardware used was limited by the selection of processors and graphics cards available on Rescale. To draw conclusions from the simulations carried out, the hardware had to be carefully evaluated before being selected to be used.

The latest Nvidia card, the H200 GPU, was not available at the time of this study. Thus, the best alternatives available to use were the Nvidia A100 and H100 cards, Nvidias flagship server GPUs from the Ampere and Hopper architectures. The main differences between the cards are discussed in section 3.1.1.

For a fair comparison, CPUs from Intel's Ice-Lake and Sapphire Rapids architectures were selected, which highlights the performance difference between CPUs and GPUs with similar release dates. The exact CPUs were chosen with the same core count to avoid different CPU parallelization capabilities affecting the results. Another factor that could influence the results and that is highly relevant for this type of application is the hardware's ability to scale performance. This relates to the performance loss of using multiple components in parallel. By using two GPU cards, the simulation time can be reduced by at most 50%, and is likely to scale worse than that. The same principles apply to CPUs. This means that running simulations with varying numbers of units is of relevance, as the scalability of the hardware can be investigated and analyzed. By scaling better, running more complex tasks in parallel can reduce the overall simulation time. The scaling is also influenced by the interconnect bandwidth between components, which differ greatly depending on whether the components are placed on the same node or on different nodes.

3.1.1 Hardware specification

	Cores	Clock speed [GHz]	Released	W_{TDP} [W]
Intel Xeon Platinum 8375C	32	3.48	Q3 2021	300
Intel Xeon Gold 6455B	32	3.00	Q1 2023	∼275

Table 3.1: CPUs used.

with core counts and clock speed taken directly from the *process_output.log* file in rescale.

	Memory [GB]	Memory Bandwidth [GB/s]	FP32 SPs	SMPs	Released	W_{TDP} [W]
Nvidia A100 PCIe	80	1845.7	6912	108	Q2 2020	300
Nvidia H100 NVL	94	3836.4	16896	132	Q3 2022	400

Table 3.2: GPUs used.

with memory, memory bandwidth, and core counts taken directly from the *process_output.log* file in Rescale.

All hardware is tested with 1x and 2x component configurations, to test scaling. All configurations are run on one node only, to avoid interconnect bandwidth between nodes influencing the results. In Figure 4.3, the interconnect speeds influence on total simulation time can be seen. Both simulations were done on the same CPU, but with varying interconnect speeds.

3.2 Calculation methods

Equations to measure the elapsed time were defined to quantify and compare results from the simulations. The simulation time is normalized to the same amount of iterations for all cases. The normalized simulation time can be combined with the iterations required for convergence to get the true simulation time. Another metric that was compared between the simulations was the power consumption of the different hardware components during the simulation, which was estimated based on the simulation time and the hardware data.

3.2.1 Time

The run time of each simulation was determined from the output log files. The startup time is defined as the time between the timestamp when ANSYS is started (the first timestamp in the log file) and the timestamp of the first iteration. The

simulation time t_{sim} is defined as the time between the timestamp of the first iteration and the last iteration, as shown in (3.3). The shutdown time is defined as the time between the timestamp of the last iteration and the last timestamp of the log file. The start and shutdown time t_{sas} is defined as the sum of the startup time and shutdown time, as defined in (3.2). The total time is then obtained by summarizing the time for loading, initializing, solving, and saving the case, as seen in (3.1).

$$t_{tot}\ [s] = t_{sas} + t_{sim} \tag{3.1}$$

where

$$t_{sas} = (ts_{firstit} - ts_{fluentstarted}) + (ts_{end} - ts_{lastiteration}) \tag{3.2}$$

$$t_{sim} = ts_{lastit} - ts_{firstit} \tag{3.3}$$

3.2.2 Energy consumption

The energy consumption for the different cases and setups is calculated differently depending on the hardware used for that setup. For the simulations running on the CPU solver, the systems tested only have CPUs, thus only making the power draw and number of CPUs relevant, along with the total time it is active. This results in (3.4). The base power draw of the rest of the computer is not accounted for in the energy consumption equations, as it cannot be estimated as accurately as the energy consumption of the processing units without access to the computers.

$$E_{tot_{CPU}}\ [Wh] = (1 + k_{cooling})\frac{t_{tot}(W_{TDP_{CPU}} n_{CPU})}{3600} \tag{3.4}$$

where $k_{cooling} = 1.5$ and n_{CPU} and n_{GPU} represents the number of CPUs and GPUs in the current setup.

For the GPU solver, both the GPU and CPU will be active, but at different times during the simulations. During the startup and shutdown time, the CPU will be running as this process is not handled by the GPU, requiring the CPU to be used. During the iteration time, the CPU will have minimum power draw, here noted as $W_{idle_{CPU}}$. This results in equation (3.5)

$$E_{tot_{GPU}}\ [Wh] = (1 + k_{cooling}) \cdot \frac{t_{sas}((W_{TDP_{CPU}} n_{CPU}) + (W_{idle_{GPU}} n_{GPU})) + t_{sim}((W_{idle_{CPU}} n_{CPU}) + (W_{TDP_{GPU}} n_{GPU}))}{3600} \tag{3.5}$$

The factor $k_{cooling}$ is slightly conservatively estimated as 1.5 in order to include the power consumption of the cooling necessary to keep the simulation hardware running at maximum performance over extended periods of time. [32]

In table 3.3 the power draw of the components used in the simulations are listed.

The two AMD CPUs are paired with the A100 and H100 GPUs, respectively, on Rescale.

	W_{TDP} [W]	W_{idle} [W]	Paired with
Intel Xeon Platinum 8375C	300	50	N/A
Intel Xeon Gold 6455B	275	50	N/A
Nvidia A100 PCIe	300	150	AMD 7v13
Nvidia H100 NVL	400	150	AMD 9684X
AMD 7v13	240	50	A100
AMD 9684X	400	100	H100

Table 3.3: Power consumption metrics for the hardware used.

3.3 Simulations

All simulations were solved in Ansys Fluent 2025 R1 on both the CPU solver and the native GPU solver. The simulations were run on the Rescale HPC cluster.

3.3.1 Porting to GPU solver

It is possible to load cases from the CPU solver straight into the GPU solver. When the case is loaded into the GPU solver, some unsupported features are automatically unselected and some need to be replaced manually. If, after that, the case is still not able to be run, the simulation file needs to be rebuilt from scratch. To do this, the mesh file is loaded into the GPU solver, and great care is taken to implement all supported features and settings exactly as they were implemented in the CPU solver. If the case still does not run, the same case is run on the CPU solver to check if removing features that are unsupported on the GPU is viable with respect to physics. If the CPU solver converges to a solution but the GPU solver does not converge, the initialization and solver settings can be tweaked according to the suggestions in the Ansys Fluent User Guide.

After porting a case to the GPU solver, it is recommended to verify its accuracy, especially if features and settings have been modified. This is done by choosing some relevant variables and coefficients to study and comparing the original CPU solver stock case, the CPU solver case with modified features and settings, and the GPU solver case with modified features and settings.

3.3.2 TRS case

The first case, shown in Figure 3.1 is a Turbine Rear Structure (TRS) case with outlet guide vanes (OGVs) that is validated against experimental results from a test rig at Chalmers University of Technology. This test rig is a closed-circuit low-speed large-scale 1.5 stage LPT-OGV facility which can achieve operating conditions that are realistic for up to large turbofan engines with Reynolds numbers of up to 435 000. For this case, the average Reynolds number for the inlet flow is 305 500 and

is based on the height of the inlet channel (outer radius minus inner radius). The total and static pressure coefficients are normalized against reference values that are measured in the bulk flow region between the OGVs. [33]

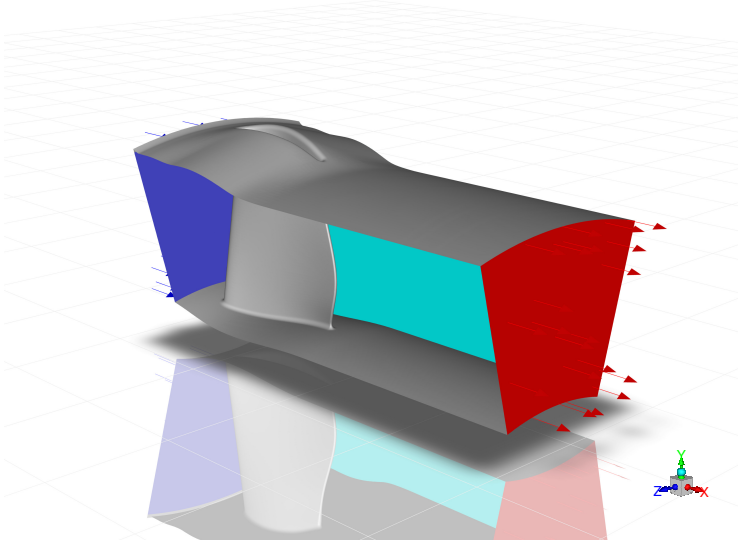

Figure 3.1: Isometric view of the TRS case

The inlet boundary is modeled as a pressure inlet through profiles of total pressure with respect to the radius, and the velocity is divided into profiles of its axial, radial, and tangential components. The outlet boundary is modeled as a pressure outlet with Radial Equilibrium Pressure Distribution.

- Solver
  - Pressure-based
  - Steady-state
  - Low-Re Correction
  - Coupled
  - Distance-based Rhie-Chow
  - Green-Gauss cell based spatial discretization gradient
- Physics
  - $k-\omega$ SST
  - Viscous Heating
  - Low-Re Correction
  - Energy Equation
- Materials
  - Aluminum
  - Air
    - Ideal gas
    - Sutherland viscosity
    - Cp and thermal conductivity defined through polynomials
- Inlet Boundary
  - Pressure Inlet
  - Total pressure profile

  - Cylindrical coordinate system velocity profiles
- Outlet Boundary
  - Pressure Outlet
  - Radial Equilibrium Pressure Distribution

3.3.2.1 GPU solver setup

As mentioned earlier in the report, all features and physics models are not available in the GPU solver, resulting in the need for changes in simulation setup in order to be able to run it. For the TRS case a number of changes have been done in order to comply with the models available in the GPU solver. The case file that was provided was run on the regular CPU solver in order to establish a baseline result that can be used to compare with the results from the GPU solver. Along with this, experimental data was provided by GKN, allowing for validation of the changes and simplifications made to the simulation file.

First, the case file was not able to run as it was stock, so the mesh was imported and then the case was set up from scratch using the same mesh file as the stock case. To achieve the same results in both solvers, the boundary conditions needed to match as close as possible. The stock case has a profile file attached that is used to generate the correct swirl on the inlet air. This cylindrical coordinate profile data was not possible to use as the GPU solver does not support profiles for cylindrical coordinates. The only type of profiles supported was for cartesian coordinates. In order to obtain the correct values for this type of coordinate system, the stock case was run on the CPU solver and the inlet velocity vector field was then exported in cartesian coordinates.

One feature that was supported for cylindrical coordinates was to re-write the profile data as mathematical expressions. This was done and the data was curve fitted as polynomials, but this caused results that did not align with the experimental data nor with the simulations from the CPU solver. After some investigation comparing the expressions in the GPU and CPU solver, it was concluded that the polynomials were a good approximation, but a bug in the GPU solver resulted in an incorrect interpretation. This bug was logged to Ansys and the chosen method was to export the profile data from the stock CPU solver case in cartesian coordinates and use that as inlet boundary condition.

All other boundary conditions were set up with the same values as in the stock case, such as the inlet velocity, temperature etc. The case was only tested using the $k-\omega$ SST model. This model is the most robust and also proved to work reliably in the GPU solver. For the scope of this thesis, it was deemed enough to test with one turbulence model in order to compare the performance between the solvers. The realizable $k-\epsilon$ model is also supported, but is not considered necessary to test for the scope of this thesis. The transition SST model is not yet supported by the GPU solver. For a more complete list of supported features, see Appendix A.

The TRS case is a relatively small case with 2 million cells, accounting for a 30°

slice of the turbine rear structure. To unleash more of the potential in the GPU solver, a full 360° case with 24 million cells was constructed and tested. To further understand the performance scaling on GPUs, it was rotated to 60°, 120°, 180°, 240°, and 300° as well.

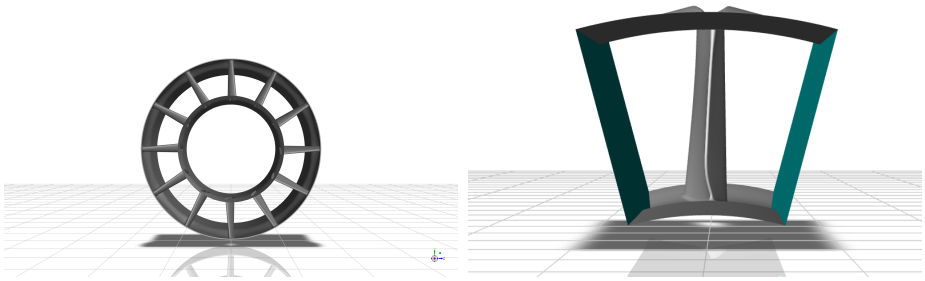

Figure 3.2: The full 360° and the 30° section of the TRS case.

3.3.3 Nozzle case

The second case is a truncated ideal convergent-divergent nozzle for a first stage launcher that is optimized from the open-source RFZ model, which is modeled after the SpaceX Falcon 9. [34] It has supersonic flow and contains combustion, which is modeled using species transport and volumetric reactions with 23 species. Turbulence is modeled using the $k-\omega$ SST model. The case takes a long time to solve, and the bulk of the calculation time is spent on calculating the volumetric reactions. Listed below are some of the most important settings used in the simulation setup.

- Mesh
  - 41.7 e+03 cells
  - 125 e+03 faces
  - Periodic boundaries
  - Resolved walls
- Physics
  - $k-\omega$ SST
  - Viscous heating
  - Production limiter
  - Energy equation
- Species transport
  - Volumetric reactions
  - Stiff chemistry solver
  - ISAT
  - Diffusion energy source
  - Finite-rate/NO TCI
  - 23 species Chemkin
- Inlet boundary
  - Pressure inlet
  - $P_t = 10.5\ MPa$

  - $T_t = 3614K$
  - Species fractions
- Outlet boundary
  - Pressure outlet
  - $P_t = 101.3\ kPa$
  - $T_t = 3615\ K$
  - No species
- Solver
  - Coupled
  - Least squares cell based
  - Rhie-chow momentum
  - Global time step
  - Double-precision
- Initialization and solution
  - Hybrid
  - 500 iterations without reactions to establish flow field
  - 15000 iterations with reactions

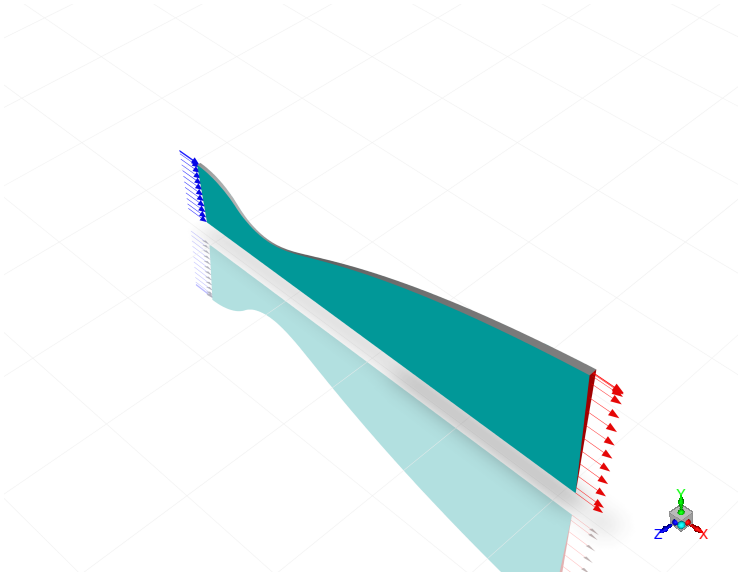

Figure 3.3: Isometric view of the Nozzle case.

3.3.3.1 GPU solver setup

The nozzle case features a quite complex set of physics, using the stiff chemistry solver with volumetric reactions to solve the combustion in a rocket nozzle. Some of these features have been implemented in recent updates. A Chemkin database was imported to define correct physical properties for the species involved in the combustion process.

A number of changes to the stock simulation file was made in order to try to make it run on the GPU solver. The inlet had to be changed from a pressure inlet to a velocity inlet or to a mass flow inlet in order to get the chemistry solver to run at

all, avoiding floating point errors. The values used for the velocity inlet were taken from a simulation run on the CPU solver.

An attempt was made to rebuild the file from the mesh, similar to the method used for the TRS case in Section 3.3.2. The original mesh was created as a 2D-mesh in ICEM. The GPU solver only supports 3D-meshes, so the 2D mesh was rotated slightly to create a 3D mesh. Since the mesh was created in ICEM it posted challenges to work with the mesh and redo it in Fluent.

Attempts to make the case run were also made by changing all available under-relaxation factors, and also by letting the initial flow field solution converge further before turning the volumetric reactions on.

After struggling to reach convergence in the stiff chemistry solver on the GPU solver with a case setup that worked in the CPU solver, a bug report was filed to Ansys and the case was investigated by their combustion experts.

3.3.4 Rotor case

The Rotor case is a large 360° model of an axial compressor without the stator. This kind of high-fidelity model may be of interest when investigating transient effects or rotor-stator interaction, for example. The Rotor case set up in CFX is converted into a Fluent solver case which is meant to be as equivalent and representative as possible. The goal is not explicitly to find a setup which gives perfect accuracy between the two different solvers, but rather to test the speedup and feature support of the GPU solver on a compressor application. The flow in the case is transonic.

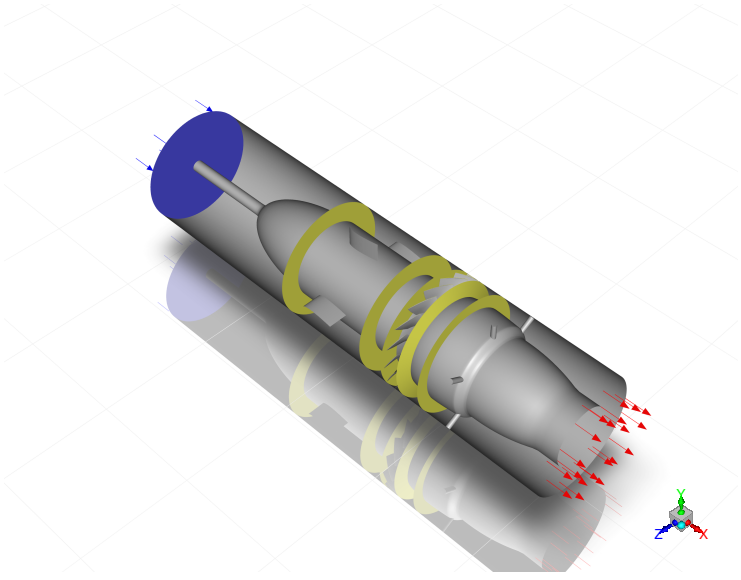

Figure 3.4: Isometric view of Rotor case.

3.3.4.1 CFX setup

- Mesh
  - 32.75 million cells
  - 100.1 million faces
  - 3 cell zones: inlet duct, rotor, outlet duct
  - Inlet-rotor interface
  - Rotor-outlet interface
  - Resolved walls
- Physics
  - $k-\omega$ SST
  - Energy equation
  - Viscous work
- Inlet boundary
  - Pressure inlet
  - $P_t = 25.00\ kPa$
  - $T_t = 288.15\ K$
- Rotor cell zone
  - Mesh motion
  - 17000 rev/min
  - Stationary walls: Shroud and static hub
  - $P_{ref} = 0\ kPa$
- Outlet boundary
  - Pressure outlet
  - $P_s = 25.25kPa$
  - $T_t = 288.15K$
- Inlet duct to rotor and rotor to outlet duct interfaces
  - Transient Rotor Stator frame change
- Solver
  - Second order backward euler
  - Double-precision
- Initialization and solution
  - Initialized with $u = 10\ m/s$
  - 2304 time steps
  - 3.06e-06 s time step size
  - 2 full rotations
  - Residual target 1e-05

3.3.4.2 Fluent GPU and CPU setup

The mesh was imported into Fluent and the setup was redone with the objective of setting it up as closely as possible as the CFX case. As the case is very large, it was not possible to iterate on the case setup in a trial and error fashion, and this resulted in some settings differing. Most importantly, the operating pressure was set to 101.325 kPa in Fluent and 0 kPa in CFX. The case setup on the CPU and GPU solvers in Fluent are identical, however.

- Mesh

  - 32.75 million cells
  - 100.1 million faces
  - 3 cell zones: inlet duct, rotor, outlet duct
  - Inlet-rotor interface
  - Rotor-outlet interface
  - Resolved walls
- Physics
  - K-ω SST
  - Production limiter
  - Energy equation
- Inlet boundary
  - Pressure inlet
  - P_t=25.00 kPa
  - T_t=288.15 K
- Rotor cell zone
  - Mesh motion
  - 1780 rad/s
  - Stationary walls: Shroud and static hub
  - $P_{ref} = 101.325\ kPa$
- Outlet boundary
  - Pressure outlet
  - P_s=25.25 kPa
  - T_t=288.15 K
- Solver
  - SIMPLE
  - Least squares cell based
  - Rhie-chow momentum
  - Second order implicit transient formulation
  - Double-precision
- Initialization and solution
  - Hybrid
  - 2304 time steps
  - 3.06e-06 s time step size
  - Maximum 20 iterations per time step
  - 2 full rotations
  - Residual target 1e-04

3.4 Business case

To build a business case for the investment of GPU solutions in an HPC environment, a wide range of factors must be considered and evaluated. Some relevant costs and benefits include the following:

- Costs
  - Direct costs
    * Purchasing price or leasing cost (including the complete computer system)

        * Licensing cost
        * Cooling system cost
    - Indirect costs
        * Electricity
        * Facilities
        * Personnel
    - Intangible costs
        * Reduced simulation capacity of CPU
    - Opportunity costs
        * Purchasing CPUs instead
        * Reserving computation capacity off-site
        * Accessing GPUs on-demand
- Benefits
    - Direct benefits
        * Reduced cost per simulation
    - Indirect benefits
        * Shorter time per simulation
    - Intangible benefits
        * Faster turnaround
        * More design iterations
        * Increase simulation capacity
        * Environmental impact
    - Competitive benefits
        * Shorter delivery times
        * Optimized products

The main opportunity cost to compare to is purchasing CPUs instead as GKN have a policy against sending simulations off-site for security reasons. Cloud computing solutions will be investigated lightly anyway as cost data was acquired throughout the project.

Computer system costs are primarily estimated from the pricing of real components with the Exxact configurator. [35] This is used in conjunction with the manufacturers' recommended prices.

Benefits are compared and summarized with a Pugh matrix. The reasoning behind the use of a Pugh matrix for this use case is that the benefits are hard to quantify, the importance of each benefit is subjective and varies between use cases and existing infrastructure, and that a Pugh matrix allows for a useful overview of the benefits that are expected.

3.4.1 Assumptions

To simplify calculations, the server cost is split into 3 parts: CPU, GPU, and base system. The base system includes other components needed for a complete server computer and is the same for both CPU and GPU servers for the sake of simplicity. The base system power draw is estimated from a server build with the AMD Epyc

9684X CPU and an Nvidia A6000 GPU. [36]

Base system cost: €8 000
Base system power draw: 300 W

Base system breakdown:
Rack-mountable server
Motherboard
192 GB RAM
2 TB SSD
10 TB Harddrive
2x 1GBase-T Ethernet
1x 1GbE Dedicated Management Port (IPMI)
2x 2600 W (1+1) Redundant - 31.5" Depth
Linux OS

The installation cost is fixed to €1000 and the delivery and shipping costs are estimated to €200 per server computer.

The prices for the CPUs and GPUs are estimated according to Table 3.4.

	Price [EUR]	Paired with
Intel Xeon Platinum 8375C	1 650	N/A
Intel Xeon Gold 6455B	2 800	N/A
Nvidia H100 NVL	25 000	AMD 9684X
AMD 9684X	6 000	H100

Table 3.4: Estimated purchasing prices for the hardware used.

3.4.2 Definitions

The cost benefit analysis calls for data that needs to be calculated, estimated, and assumed through different equations and definitions. This is not an exact science, but rather a balance needs to be struck between accuracy and simplicity. For completeness, the factors not included in the calculations are also explained.

Leasing cost:

$$C_{leasing} = \sum_{1}^{3} C_{leasing,year} \tag{3.6}$$

Electricity cost:

$$C_{electricity} = 24 \cdot 365 P_{tot} p_{electricity} \tag{3.7}$$

where the electricity price $p_{electricity} = €0.087$ per kWh

Regarding the licensing cost, Ansys has a license package called Ultimate, that includes both the license for Fluent and the HPC packs. Thus the difference between CPU and GPU license cost will be zero, as the same Ultimate license can be used to run simulations on both solvers, making the total licensing cost indifferent and not needed in the cost comparisons. In addition, the cost calculations in this project compares adding systems into an already-existing cluster, while the Ultimate license package covers the entire cluster.

Power consumption:

$$P_{tot_{CPU}}\ [W] = n_{CPU}(\frac{W_{Basesystem}}{n_{CPUpersystem}} + W_{TDP_{CPU}}))(1 + k_{cooling}) \tag{3.8}$$

where $k_{cooling} = 1.5$ and $n_{CPU/GPU}$ is the number of CPUs and GPUs in the current setup.

The power consumption calculation for the GPU system assumes that the GPUs are running at full load, the CPU runs at idle and that all GPUs fit into one system.

$$P_{tot_{GPU}}\ [W] = (W_{idle_{CPU}} + W_{TDP_{GPU}} n_{GPU} + W_{Base\ system}) \cdot (1 + k_{cooling}) \tag{3.9}$$

The cooling system cost is calculated as:

$$C_{cooling\ system} = \frac{C_{cooling} P_{tot} k_{cooling}}{1 + k_{cooling}} \tag{3.10}$$

where the cooling cost $C_{cooling}$ is estimated to €1.5 per W of cooling needed. This assumes liquid cooling, that the systems evaluated have a small number of nodes, and that the customer already has a cooling tower to produce 35°C water into the Coolant Distribution Units.

Chapter 4

Results

The simulation results are split by case and used to build business cases of GPU purchases and CPU purchases that account for equivalent simulation capacity. As the GPU solver does not support all features of the CPU solver, the business case is designed to evaluate replacing a minority of the CPU simulation capacity with GPUs in a first-stage adoption of GPUs for CFD simulations.

4.1 TRS case

The TRS case compares all CPUs and GPUs in Tables 3.1 and 3.2 in a 1x and 2x configuration. The comparisons shown are for the 30° case and 360° case.

4.1.1 Simplifications and issues

The TRS case required some setup modifications to run on the GPU solver, as outlined in section 3.3.2.1. To summarize, the original pressure inlet was replaced with a velocity inlet, the cylindrical coordinate system velocity profiles in the inlet were replaced with a cartesian velocity vector field imported from the CPU solver solution, the radial equilibrium pressure distribution setting in the outlet was removed, the Low-Re correction was removed from the turbulence model, and finally, the cell based spatial discretization was changed from Green-Gauss to least squares. In addition to this, the 30° TRS case was not possible to run on 2x Nvidia H100 GPUs.

4.1.2 Time

In the simulation time comparison of the 30° TRS case shown in Figure 4.1, it is observed that the time to perform 300 iterations is reduced on every Nvidia GPU setup compared to every Intel CPU setup. The start and shutdown time is slightly higher on the GPUs than on the CPUs. The total time reduction ranges between 41.2-77.5%. The 300 iteration time reduction ranges between 69.7-90.3%.

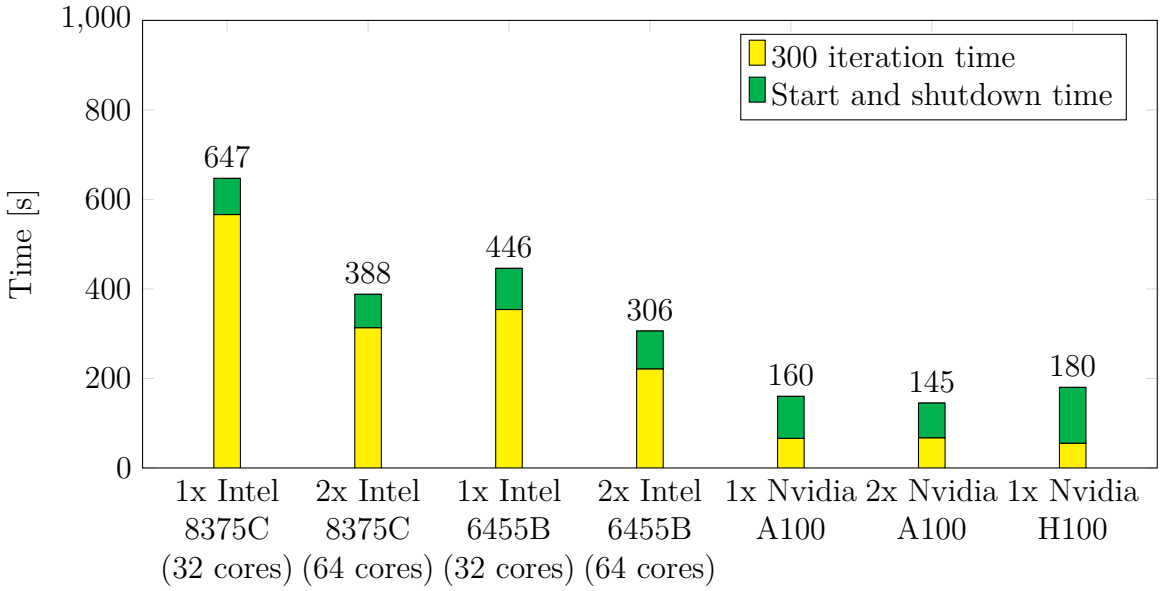

Figure 4.1: 30° TRS case in single precision.

In the simulation time comparison of the 360° degree TRS case shown in figure 4.2, the reduction of the 300 iteration time on GPUs is larger than in the 30° case. The total time reduction ranges between 78.9-95.2%. The 300 iteration time reduction ranges between 86.8%-98.0%.

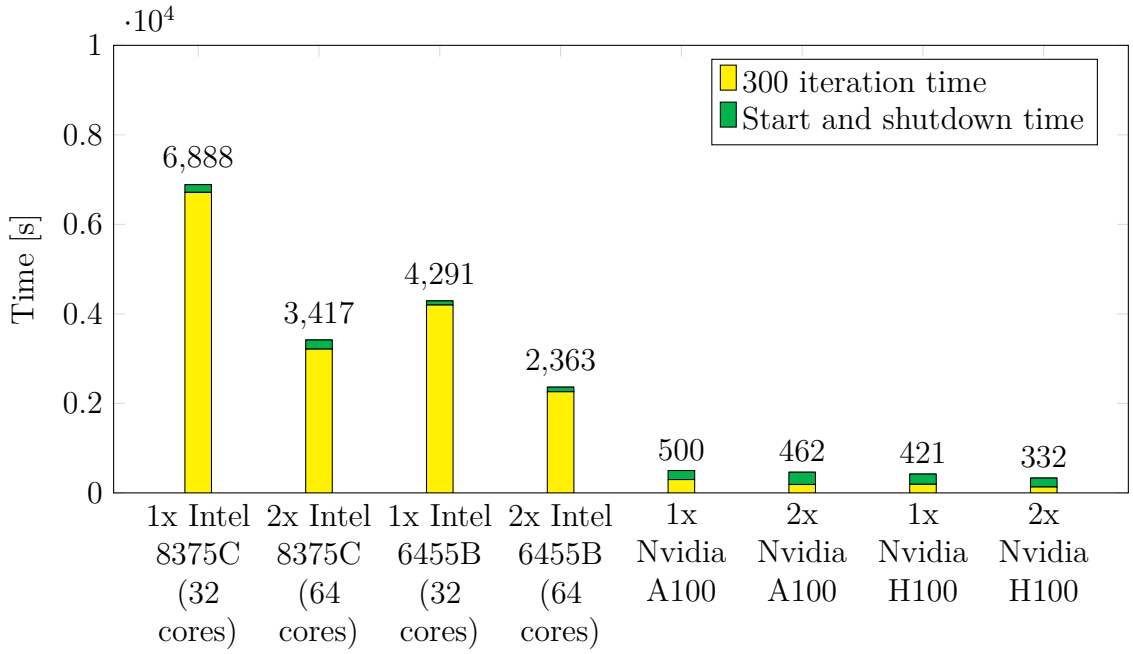

Figure 4.2: 360° TRS case in single precision.

In Figure 4.3, it can be seen how varying interconnect speeds between nodes affect the overall simulation time for the 30° simulation of the TRS case. Both simulations

were run using four Intel 8375C CPUs, both with 32 cores each. Each node contains two CPUs each, resulting in two total nodes in use.

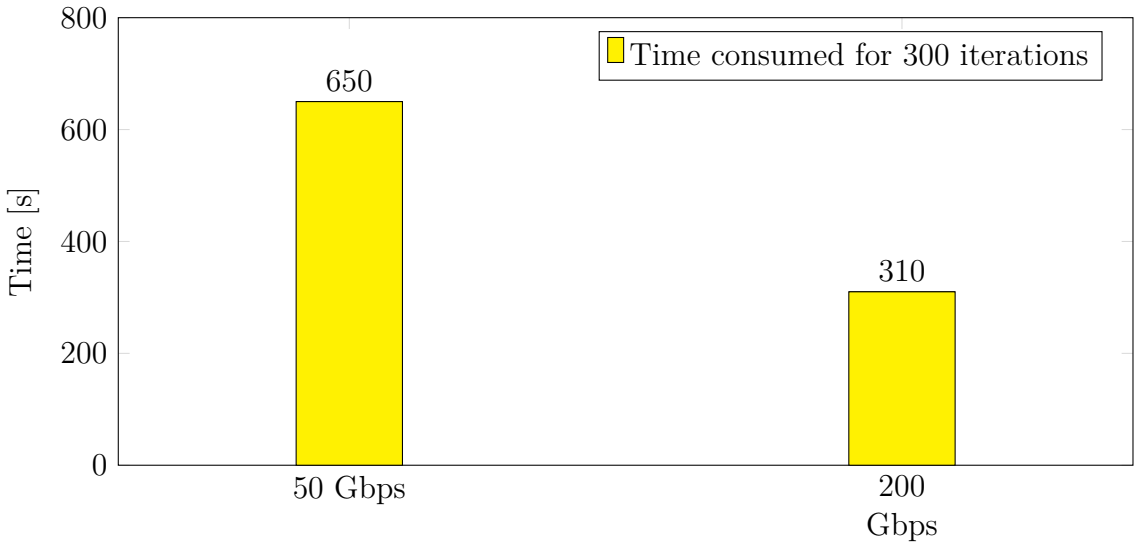

Figure 4.3: Interconnect speed on the 30° simulation of the TRS case.

Figure 4.4 shows the simulation time of the TRS case on single A100 and H100 GPUs for different mesh sizes. It has been obtained by copying and rotating the 30° slice of the turbine, resulting in mesh sizes ranging between 2-24 million cells.

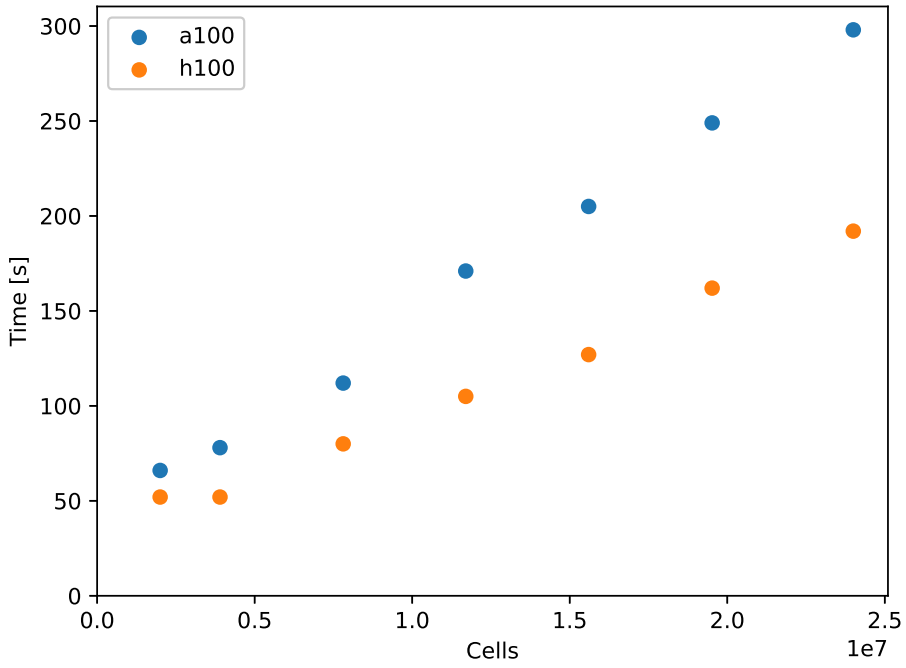

Figure 4.4: GPU solver simulation time over mesh size for the TRS case.

4.1.3 Energy consumption

Figure 4.5 shows the energy consumption for running 300 iterations on the 360° TRS case in single precision. The reduction in energy consumption on the tested GPUs compared to the tested CPUs ranges between 87.7-93.4%.

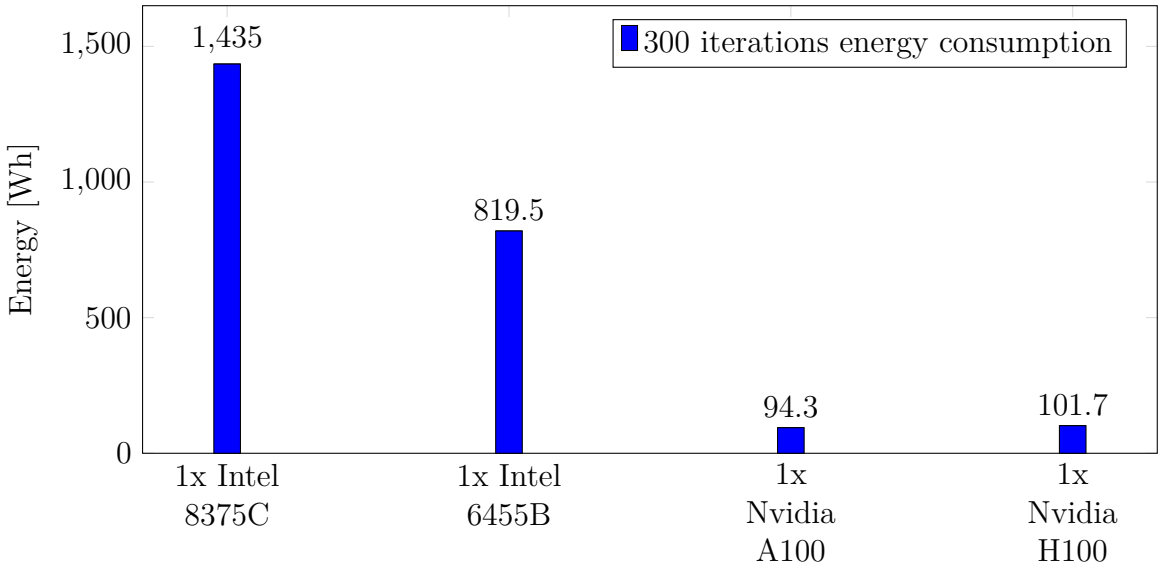

Figure 4.5: 360° single precision energy consumption for running 300 iterations.

4.1.4 Accuracy and comparison

The single-precision accuracy comparison in Figure 4.6 shows circumferentially-averaged data from the stock 30° CPU TRS case and compares it with the simplified GPU case, and with experimental data from the test rig. It compares the total pressure coefficient C_P0 in an outlet plane. The simplified GPU setup correlates closely with the stock CPU case for the total pressure coefficient $Cp0$, although the physics settings differ slightly, as discussed in Section 4.1.1. The simplified GPU case has also been compared against CPU case with the same simplifications and C_P0 is equal up to $\sim$2 significant digits, while the total pressure is equal up to $\sim$5 significant digits. When plotting the simplified GPU case against the CPU case with the same simplifications in the same manner as in Figure 4.6, they look identical to the naked eye.

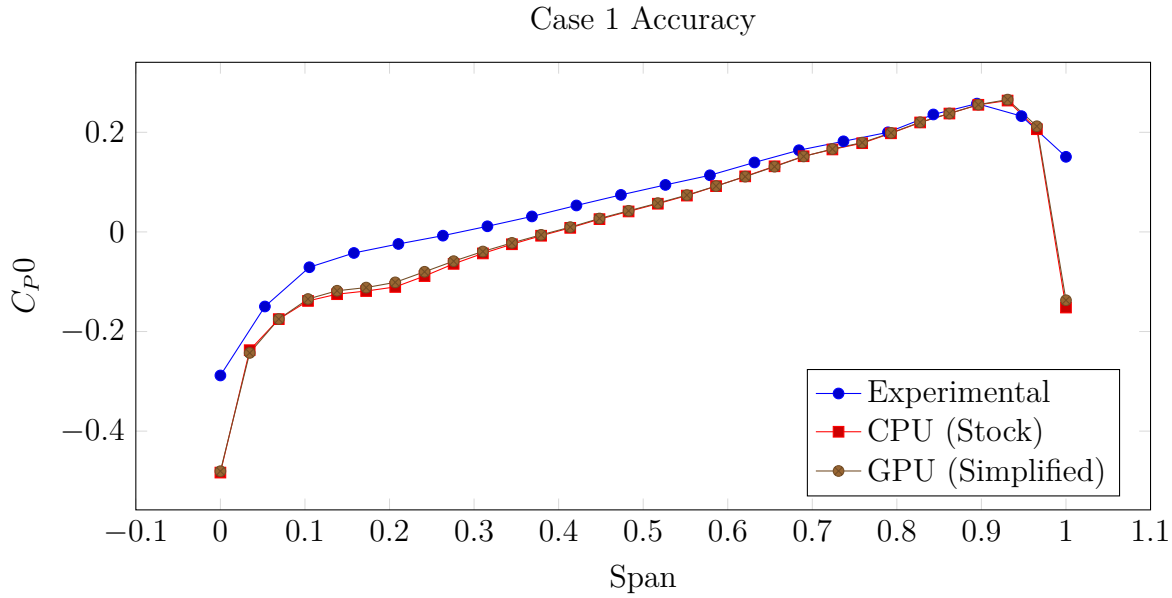

Figure 4.6: Circumferentially-averaged C_P0 over span in the outlet.

4.1.4.1 Convergence

Convergence was tested on the 30° version of the TRS case, using the same convergence criteria for both solvers, run in both single and double precision. Figures 4.7 and 4.8 show the number of iterations required for the solution to converge. The criteria was set as $1e^{-04}$ for all residuals, and all residuals needed to reach this value for the solution to converge. The residuals were continuity, x, y, and z velocities, energy, k, and ω. The data from the convergence comparisons done shows that the GPU solver requires 51.4% fewer iterations compared to the CPU solver in single precision, and 26.7% fewer iterations in double precision.

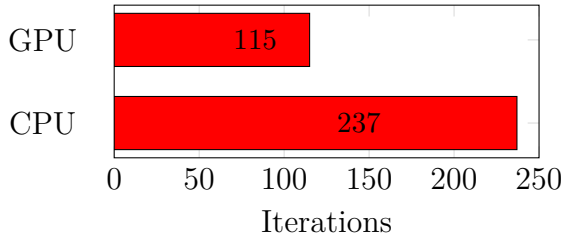

Figure 4.7: No. of iterations until converged for 30° TRS case in single precision.

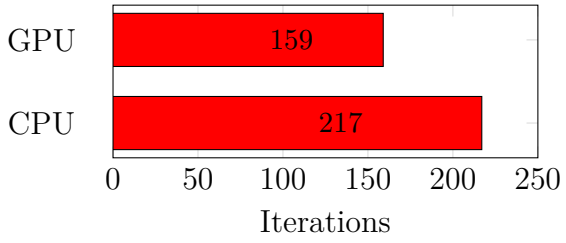

Figure 4.8: No. of iterations until converged for 30° TRS case in double precision.

4.1.5 Cloud cost

Running the 360° TRS case on the GPU solver on the Rescale cloud results in a cost reduction of 83.3% for the simulations in Figure 4.2 compared to running it on the

CPU solver on the Rescale cloud. The GPU solver is run on 2x H100 GPUs, and the CPU solver on 2x 6455B CPUs.

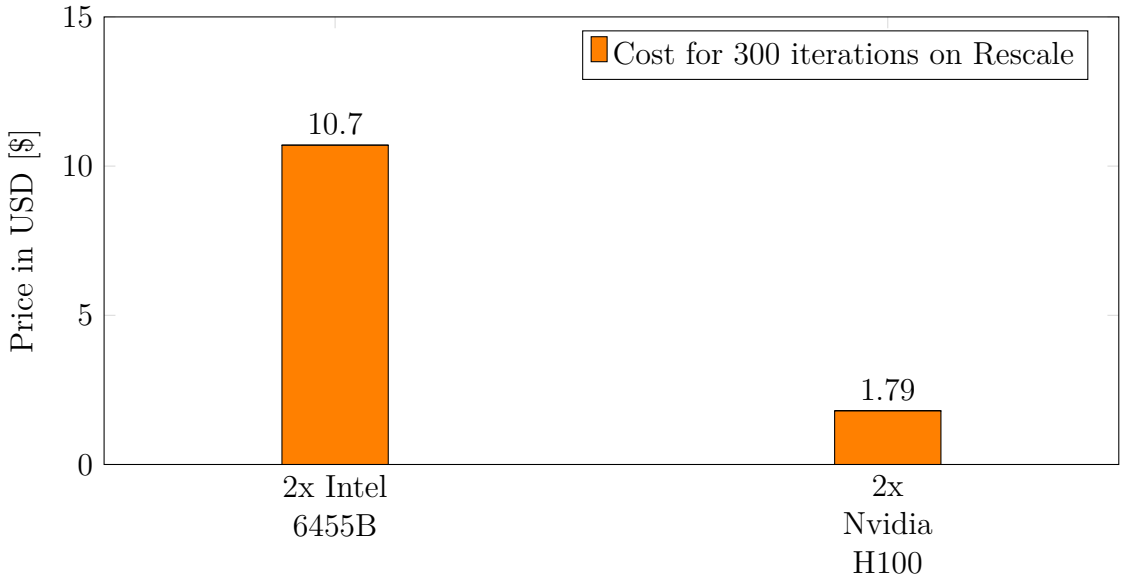

Figure 4.9: 360° TRS case Rescale cost for total time of simulation.

4.2 Nozzle case

Thorough testing of the nozzle case was done, but convergence could not be reached on the GPU solver without complete changes to the case setup. The same GPU solver setup of the case was successfully solved on the CPU solver. The conclusion is that the nozzle case is not possible to solve on the GPU solver. After Ansys tested the case, an Ansys combustion expert confirmed that this nozzle case from GKN does not run on the GPU solver in version 2025 R1. The closest modifications that can be made to reach convergence is to convert the case to transient and rotate the mesh to 90°. This makes the case take a significantly longer time to run, completely erasing the potential speedup benefits of running the case on the GPU solver. It should also be noted that the nozzle case would likely utilize GPUs poorly anyway, since the mesh only contains 41 k cells and because the bulk of the calculation time is taken by the stiff chemistry solver, which solves different equations compared to the fluid flow equations, and those equations seem to be less fit for parallelization.

4.3 Rotor case

The Rotor case was run on double precision and since the case is computationally expensive, only two computing setups were compared: 8x Intel 8375C CPUs totaling 256 cores, and 2x Nvidia H100 GPUs. As discussed in Section 4.3.5, the Fluent case was setup differently than the CFX case. However, this does not affect the comparisons between the Fluent CPU solver and the Fluent GPU solver.

4.3.1 Simplifications and issues

The rotor case ported into Fluent also required some simplifications. The most relevant is that the Fluent GPU solver only supports that SIMPLE pressure-velocity coupling scheme, while CFX is a coupled solver. It also did not run on 1x H100 GPU, due to memory overload on the VRAM.

4.3.2 Time

The Fluent GPU solver is compared with the Fluent CPU solver. Figure 4.10 shows a 66.6% total time reduction between 8x Intel 8375C CPUs totaling 256 cores and 2x Nvidia H100 GPUs per 10 000 iterations. It can be seen that the start time becomes negligible for cases that require this scale of computation. Figure 4.10 does not take into account the significantly faster convergence of the GPU solver.

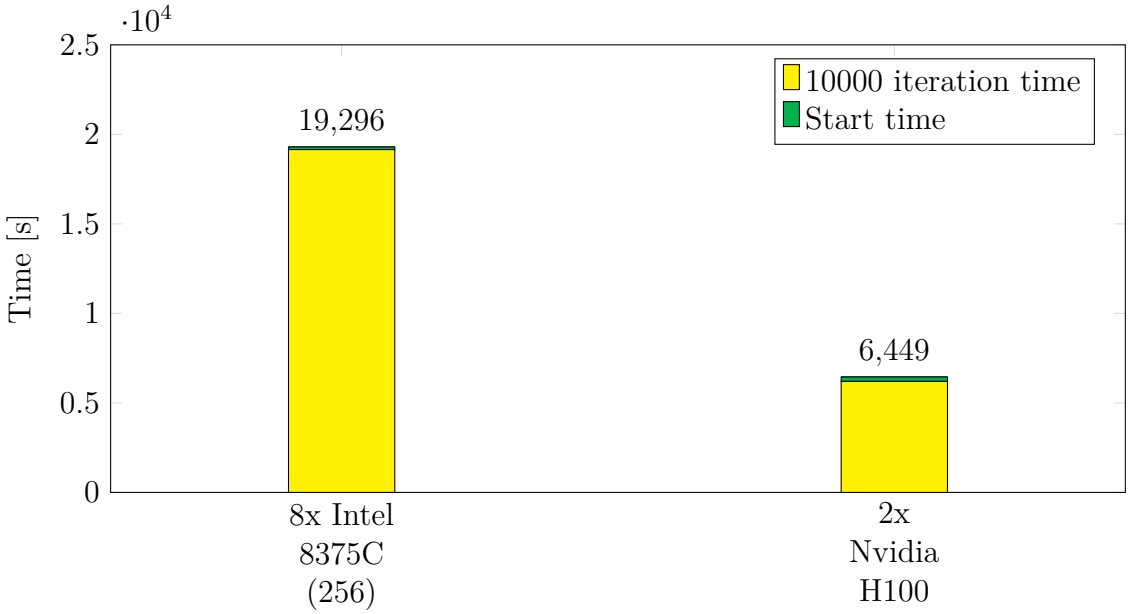

Figure 4.10: Rotor case double precision time per 10 000 iterations.

4.3.3 Energy consumption

Figure 4.11 shows energy consumption per 10 000 iterations for the rotor case in double precision. The energy consumption is reduced by 88.2% .

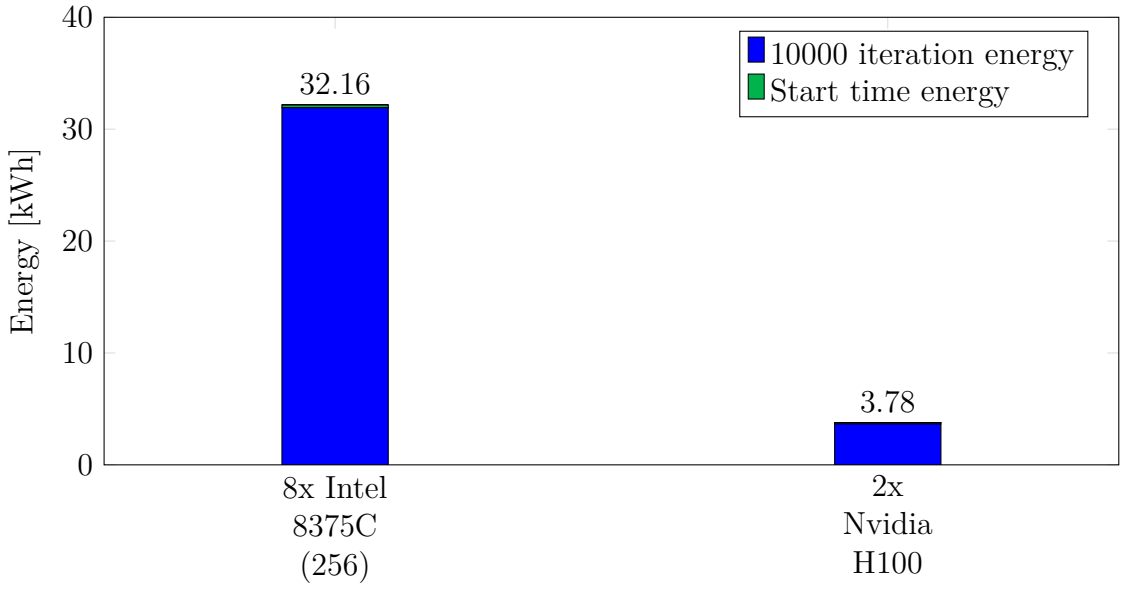

Figure 4.11: Rotor case double precision energy consumption per 10 000 iterations.

4.3.4 Convergence

Figure 4.12 shows a 73.1% reduction in the number of iterations required for the Rotor case in double precision to converge, with the convergence criteria set to $1e^{-04}$ for all residuals, and with all residuals needing to reach this value for the solution to converge, each time step allowing a maximum of 20 iterations. The residuals were continuity, x-, y-, and z-velocities, energy, k, and ω. It is noted that the continuity equation converged significantly slower on the CPU solver. Figure 4.12 is a comparison between the Fluent CPU solver and Fluent GPU solver with the SIMPLE pressure-velocity coupling scheme. The Rotor case was also tested with the PISO pressure-velocity coupling scheme in the CPU solver, and similar convergence behavior to the SIMPLE scheme in the CPU solver was observed. It should also be noted that the CPU solver consistently hit the maximum of 20 inner iterations per time step for most time steps for both the SIMPLE and PISO coupling scheme. Therefore, the reduction in the number of iterations required may have been even more dramatic had the 20 inner iteration maximum been removed.

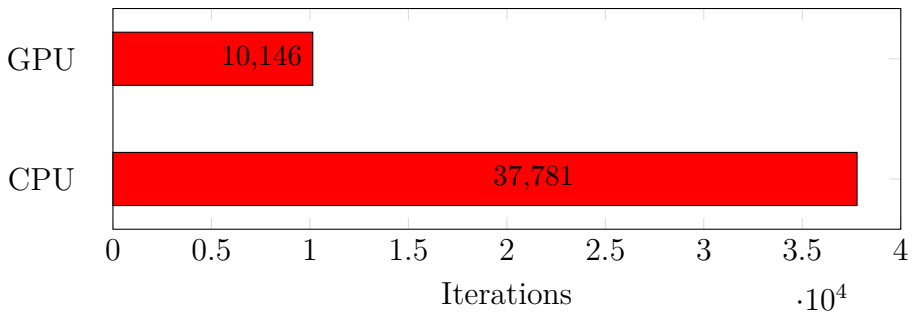

Figure 4.12: No. of iterations until converged for the Rotor case.

4.3.5 Accuracy and comparison

In Table 4.1, the results from the Rotor case from the original CFX file are compared to the results obtained from the simulations in Fluents GPU solver. The results from the Fluent CPU solver were not saved as the simulation reached the maximum walltime before it finished running. The ambient pressure was set 5x higher in Fluent, making the actual fluid flow case compared different from the CFX case. However, it was decided to be worthwhile to compare the pressure ratio and corrected mass flow between the case setups anyway. The pressure ratio is defined in (2.32), and the corrected massflow in (2.33). It can be seen that the pressure ratio P_{ratio} over the rotor is 12.8% larger in CFX, and the corrected massflow $\dot{m}_{corr}$ is 33.4% larger in CFX. P_{t1} is the total pressure in the interface between the inlet duct and rotor, and P_{t2} is the total pressure in the interface between the rotor and the outlet duct.

	CFX	Fluent GPU
P_{t1} [kPa]	24.84	124.1
P_{t2} [kPa]	27.55	135.9
P_{ratio}	1.109	1.095
$\dot{m}_{corr}$ $[\frac{kg}{s}]$	2.651	1.988

Table 4.1: Comparison between results from CFX and Fluent GPU solver.

4.3.6 Cloud cost

Running the Rotor case on the GPU solver on the Rescale cloud reduces the cost by 90.8% per 10 000 iterations compared to the CPU solver. The GPU solver is run on 2x H100 GPUs and the CPU solver is run on 8x 8375C CPUs.

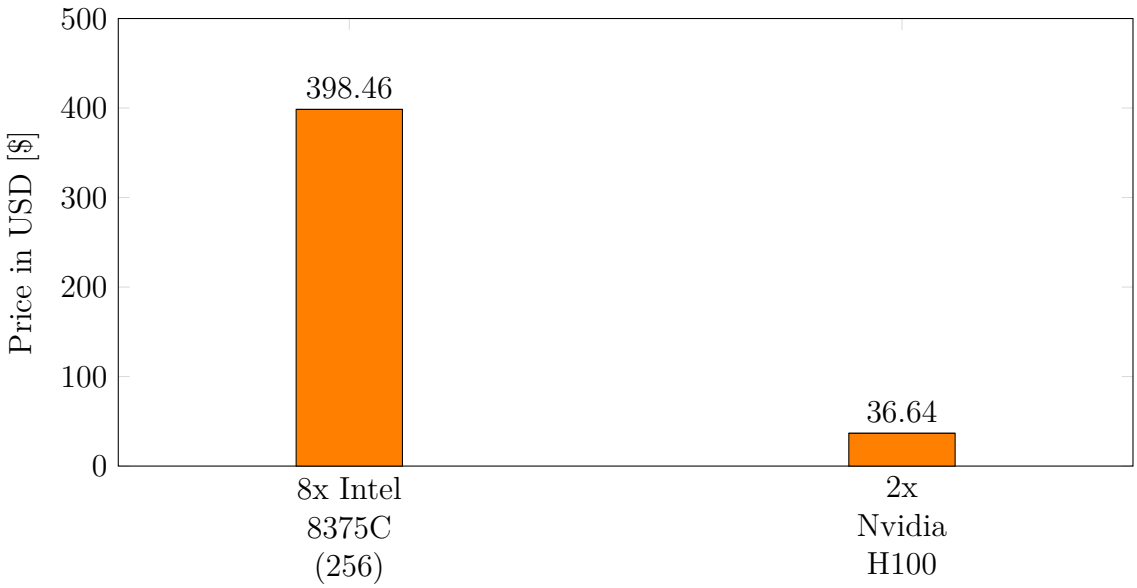

Figure 4.13: Rotor case Rescale cost per 10 000 iterations.

4.4 Business case

The faster convergence of the GPU solver is not taken into account in the business case as it is likely to vary depending on the convergence criterias that are chosen and the case that is run.

4.4.1 Total cost of ownership

An estimation of total cost of ownership (TCO) is most useful if set in relation to an opportunity cost. In this case the TCO on 2x H100 GPUs is compared to CPUs of equivalent simulation capacity, meaning that the two different investments provide the capacity to perform the same simulations in the same timeframe. In the TRS case comparison, the CPU cases are run on one node each. The simulations on the TRS case presented in Figure 4.2 can be interpreted to show that 7 simulations can be run sequentially on the two H100 GPUs in the same timeframe that 7 simulations can be run in parallel on 14x Intel 6455B CPUs, this interpretation is visualized in Figures 4.14 and 4.15, and serves as a basis for the TCOs in the business case. Since it is possible to mount a maximum of 2 CPUs in one computer, 14 CPUs require 7 computers to house them.

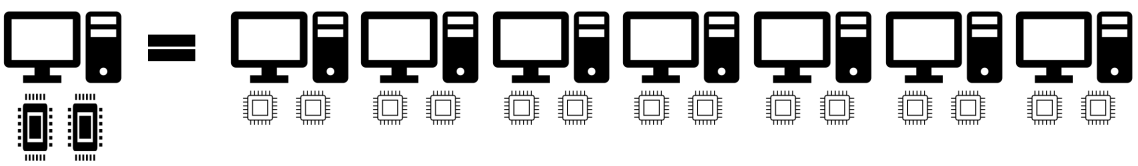

Figure 4.14: Visualization of equivalent simulation capacity for 2x Nvidia H100 compared to 14x Intel 6455B.

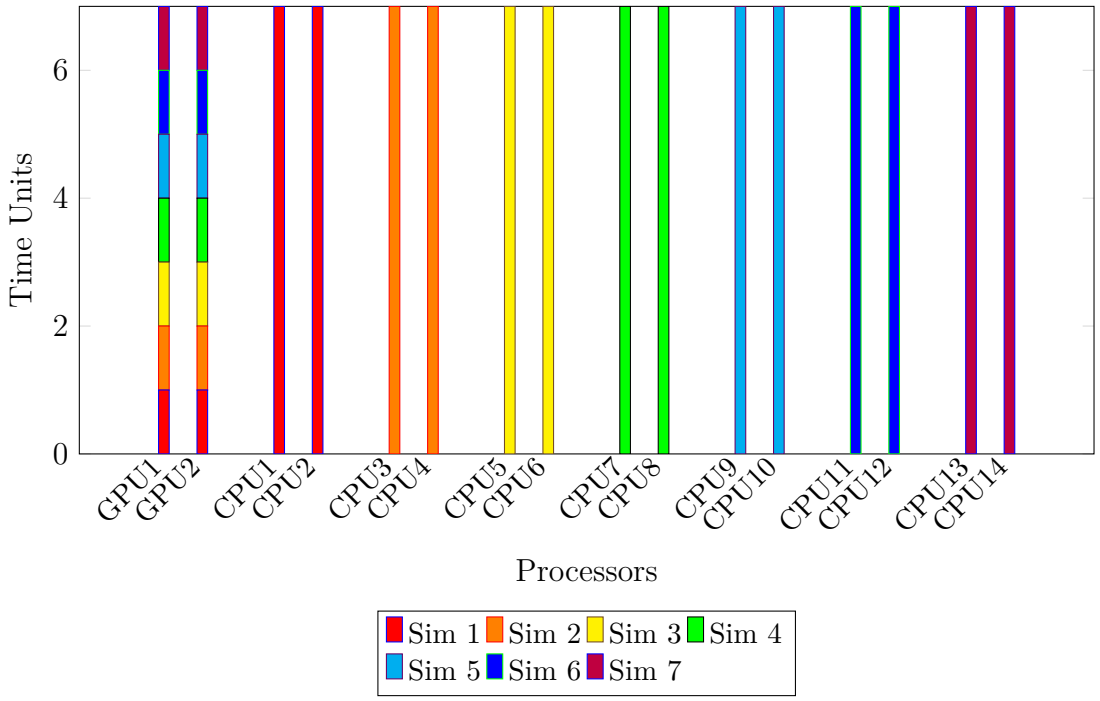

Figure 4.15: Equivalent simulation capacity for the TRS case when comparing 14x Intel 6455B with 2x Nvidia H100.

Table 4.2 shows the TCO of 2x Nvidia H100 GPUs. It assumes that they are fit in one computer system with one AMD 9684X CPU. It includes the cost of expanding the cooling system, the shipping, delivery, and installation costs, and the cost of electricity. It does not include other operational costs, end-of-life costs, and indirect costs as they either require rough estimations or disclosure of sensitive information.

	Year 1	Year 2	Year 3	3-year cost
Acquisition costs				
GPU lease	€17 000	€17 000	€17 000	€51 000
CPU lease	€2 000	€2 000	€2 000	€6 000
Rest of computer	€2 670	€2 670	€2 670	€8 000
Cooling system cost	€1 250			€1 250
Delivery and shipping costs	€200			€200
Installation costs	€1 000			€1 000
Operational costs				
Electricity	€2 300	€2 300	€2 300	€6 900
Maintenance and repairs				Not included
Labor costs				Not included
Facilities				Not included
Licensing costs				Not included
End-of-life costs				
Disposal				N/A
Residual value				N/A
Indirect costs				
Training costs				Not included
Downtime or lost productivity				Not included
Compliance or regulatory costs				N/A
Sum				**€75 350**

Table 4.2: Total Cost of Ownership of 2x Nvidia H100 over a 3 year period.

The TCO of 2x Nvidia H100 shown in Table 4.2 shows a 48.0% reduction in cost compared to the TCO of 14x Intel Xeon 6455B shown in Table 4.3. It can be seen that while the GPUs themselves are more expensive, the cost of the rest of the computers and electricity is a lot cheaper for a GPU system compared to CPU systems of equivalent capacity.

	Year 1	Year 2	Year 3	3-year cost
Acquisition costs				
CPU lease	€13 000	€13 000	€13 000	€39 000
Rest of computer	€18 700	€18 700	€18 700	€56 000
Cooling system cost	€13 400			€13 400
Delivery and shipping costs	€1 400			€1 400
Installation costs	€1 000			€1 000
Operational costs				
Electricity	€11 300	€11 300	€11 300	€34 000
Maintenance and repairs				Not included
Labor costs				Not included
Facilities				Not included
Licensing costs				Not included
End-of-life costs				
Disposal				N/A
Residual value				N/A
Indirect costs				
Training costs				Not included
Downtime or lost productivity				Not included
Compliance or regulatory costs				N/A
Sum				**€144 800**

Table 4.3: Total Cost of Ownership of 14x Intel Xeon 6455B (448 cores) over a 3 year period.

In the rotor case, the relation between simulation speed and processing units differ and the opportunity cost in the business case is therefore modified to account for this. 24x Intel Xeon 8375C CPUs totaling 768 cores are estimated to represent equivalent simulation capacity to 2x H100 GPUs. The TCO of 2x Nvidia H100 shown in Table 4.2 shows a 66.5% reduction in cost compared to the TCO of 24x Intel Xeon 8375C shown in Table 4.4.

	Year 1	Year 2	Year 3	3-year cost
Acquisition costs				
CPU lease	€13 200	€13 200	€13 200	€39 600
Rest of computer	€32 000	€32 000	€32 000	€96 000
Cooling system cost	€24 300			€24 300
Delivery and shipping costs	€2 400			€2 400
Installation costs	€1 000			€1 000
Operational costs				
Electricity	€20 600	€20 600	€20 600	€61 700
Maintenance and repairs				Not included
Labor costs				Not included
Facilities				Not included
Licensing costs				Not included
End-of-life costs				
Disposal				N/A
Residual value				N/A
Indirect costs				
Training costs				Not included
Downtime or lost productivity				Not included
Compliance or regulatory costs				N/A
Sum				**€225 000**

Table 4.4: Total Cost of Ownership of 24x Intel Xeon 8375C (768 cores) over a 3 year period.

Below, in Figures 4.16, 4.17 and 4.18 the cost for using a local cluster solution compared to using a cloud service over a three year period is shown. The cloud cost is based on a fixed sum for each hour used for the specific hardware of each case on Rescale. The hardware setups are matching in terms of simulation capacity, as calculated in Section 4.4.1.

The cost for the local cluster solution is based on the calculations from Section 4.4.1. The only running cost is the electricity price. For each of the hardware configurations, the break even point is shown in the respective graph. This point marks where it becomes cheaper to have a local cluster compared to using an available online cloud solution. The x-axes range from 0 hours to 26280, which is equivalent to three years.

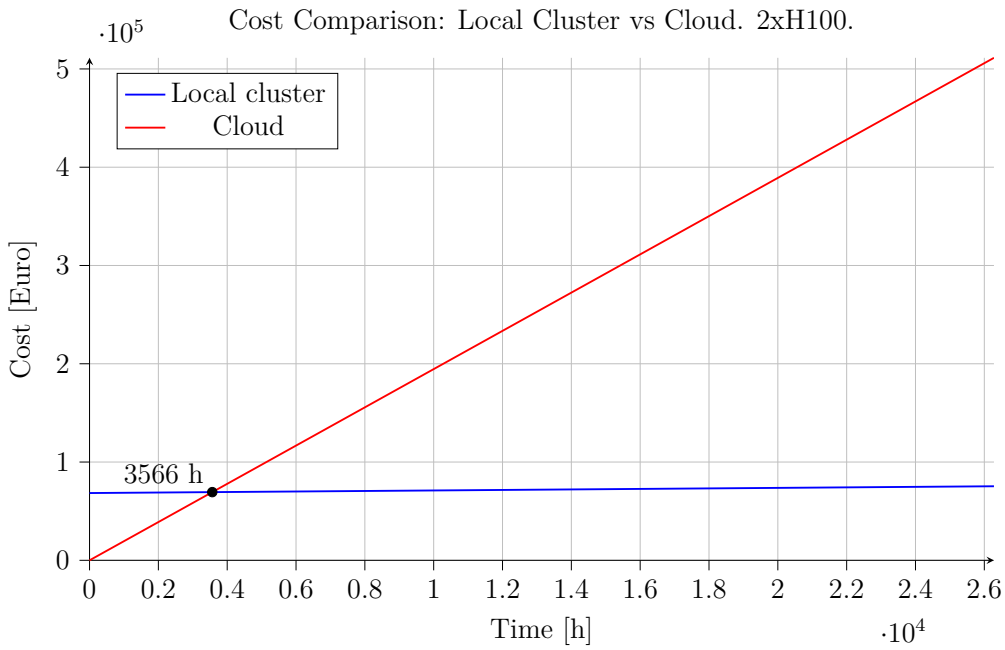

Figure 4.16: Local vs cloud cost over a 3 year period for 1 2x H100 machines.

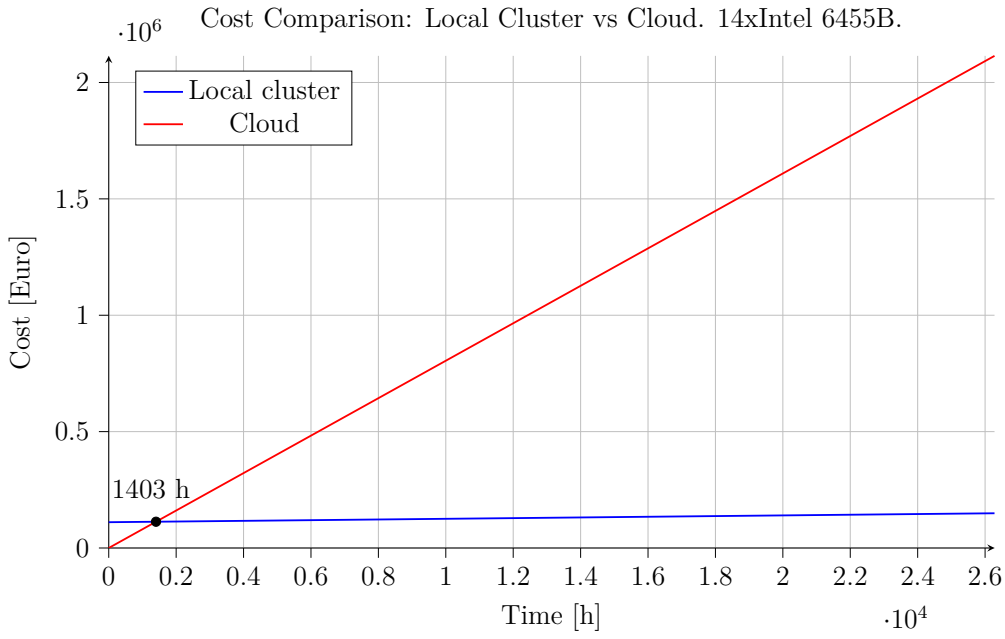

Figure 4.17: Local vs cloud cost over a 3 year period for 7 2x 6455B machines.

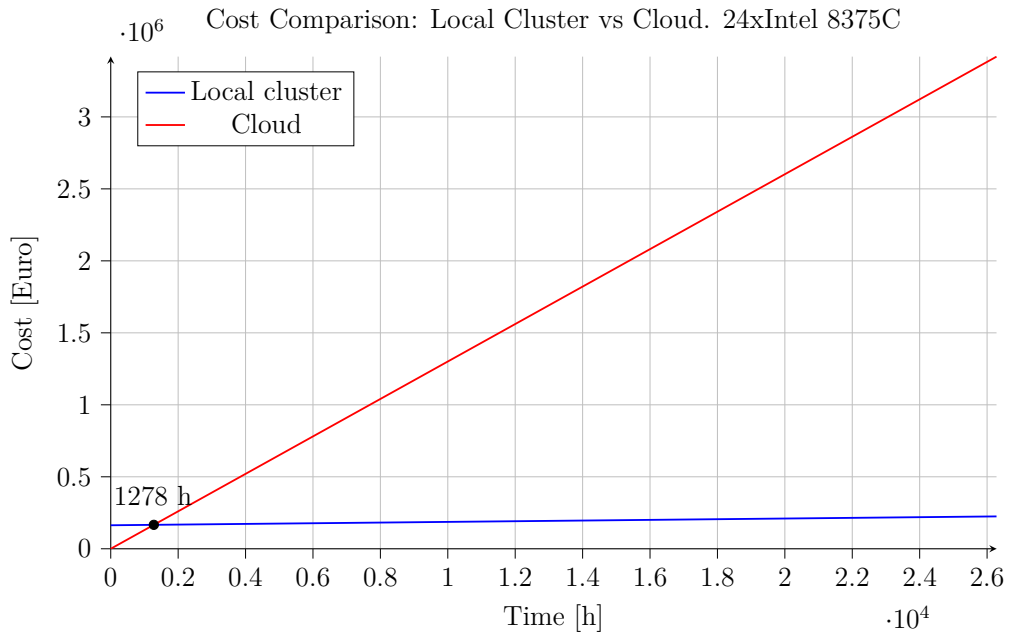

Figure 4.18: Local vs cloud cost over a 3 year period for 12 2x 8375C machines.

4.4.2 Benefits

The benefits of CPUs and GPUs for CFD simulations are compared in Table 4.5. How these benefits should be weighted is quite subjective and use-case-dependent, therefore, no weights are applied. There is also some overlap between some of the benefits.

	CPU	**GPU**
Direct benefits		
Cost per simulation	-	+
Indirect benefits		
Time per simulation	-	+
Supported physics	+	-
Power consumption	-	+
Intangible benefits		
Turnaround time	-	+
Number of design iterations possible	-	+
Simulation capacity	-	+
Environmental impact	-	+
Ease of implementation	+	-
Flexible scaling	+	-
Potential scalability	-	+
Competitive benefits		
Delivery time of projects	-	+
Product optimization	-	+

Table 4.5: Pugh matrix of benefits for CPU system vs. GPU system investment.

4.4.3 System power consumption

The power consumption of the different systems outlined in Section 4.4.1 is shown in Figure 4.19. The 2x H100 system shows a 79.9% reduction in power consumption compared to the 14x 6455B system, and a 88.9% reduction in power consumption compared to the 24x 8375C system.

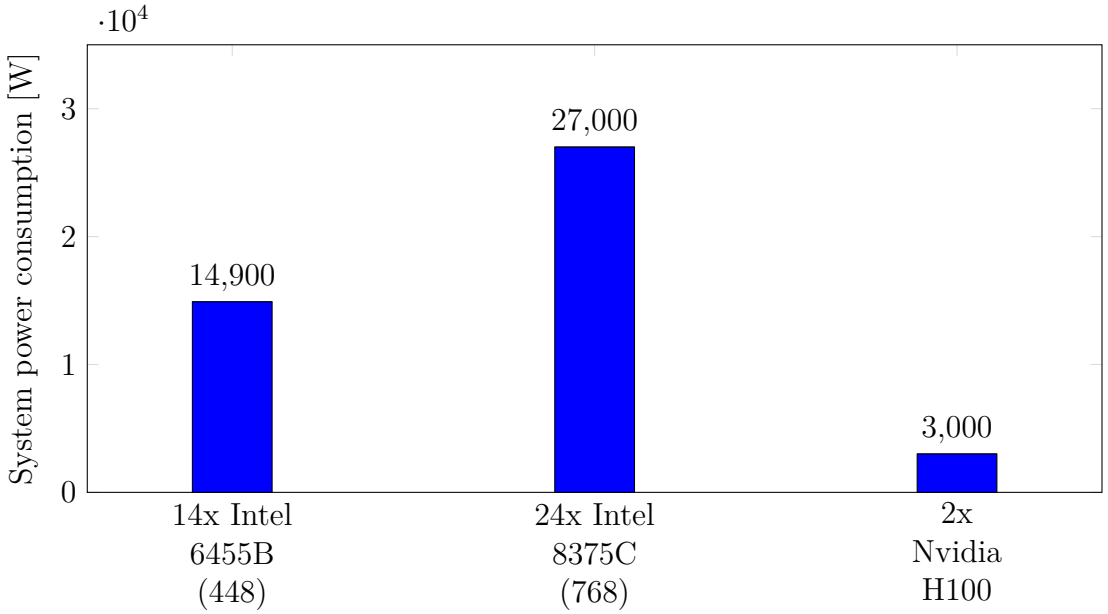

Figure 4.19: System power consumption.

Chapter 5

Discussion

Of the three cases that are representative of typical aerospace applications, the TRS case and Rotor case were possible to run on the Ansys Fluent GPU solver, while the Nozzle case was not able to be run. In addition, the TRS case required multiple simplifications and the Rotor case had to be ported from CFX to Fluent. The rather complicated process of porting the cases requires engineering time and necessitates testing and validation to ensure simulation accuracy. However, after this hurdle was overcome, the GPU solver proved its worth. Of the parameters tested: *simulation time*, *energy consumption*, *convergence*, and *cost analysis*, all were consistently better on the GPU solver with the hardware specified in Section 3.1.1 for both the TRS case and the Rotor case. Section 4.1.4 shows that even though the TRS case required multiple simplifications to run on the GPU solver, it still provided accurate results. While the GPU solver does not seem mature enough yet to reliably handle combustion problems, it seems ready to be implemented on real-world turbomachinery problems today.

Purchasing GPU systems is estimated to be cheaper than purchasing CPU systems of equivalent simulation capacity. To reach these numbers, many assumptions and price estimations had to be made, all of which are accounted for in Sections 3.4.1 and 3.4.2. Applying these results to other businesses is possible as well, but as local prices and existing infrastructure varies from business to business, and as the equivalent simulation capacity may differ on a simulation-by-simulation basis, the TCOs should be recalculated for every business application. The calculations provided in this study can then serve as a blueprint for what to take into account.

Identifying simulation cases that benefit from being run on the GPU solver correctly is important to make sound purchasing decisions regarding GPUs, and to utilize bought GPUs to their full extent. In general, the bulk of the calculations done in the simulation should be done to solve the fluid flow, rather than chemistry problems for instance. Simulations should take more than a couple of minutes to run for the speedups to be significant, as the GPU solver does not speed up the start and shutdown time, t_{sas}. It should also be taken account that meshing would be done on the CPU, therefore the total time increases significantly if the case needs to be re-meshed between every run. The solver matrix size needs to be sufficiently

large to take advantage of multiple GPUs, at minimum around 2 million cells per GPU. On the other hand, the entire solver matrix needs to fit in the VRAM of the GPU. This creates a size window that the solver matrix needs to fit in. Finally, the case setup should be compared against the supported features of the GPU solver, found in Appendix A.

Figure 5.1 shows a decision tree model for how to approach GPUs for CFD simulations as a series of business decisions over three adoption phases. After identifying simulation cases that can benefit from being run on the GPU solver. The next step is to estimate the GPU simulation capacity required for the next 3 years. If the simulation capacity required exceeds 4000 h on 2x H100, which is the breakeven between 2x H100 on a local cluster and the Rescale cluster as shown in Figure 4.16, it makes economic sense to buy 2x H100 cards. The performance of and demand for GPU simulations should then be continuously evaluated on a yearly basis. This allows for further GPU purchases if the demand is high, while minimizing sunk costs if the demand decreases. While implementing GPU simulations in the early adoption phase may seem to be a lot of work for the reduced cost, time, and power consumption on a limited number of simulation cases, it positions the company better for a future larger investment in GPUs. Allowing for accuracy testing and exploration of higher-fidelity simulations at an earlier stage compared to the competition, and being prepared to scale up faster when the solver is deemed mature enough. This goes hand in hand with the continuous updates of the GPU solver from Ansys making the solver more mature, as well as the fast development of GPU performance.

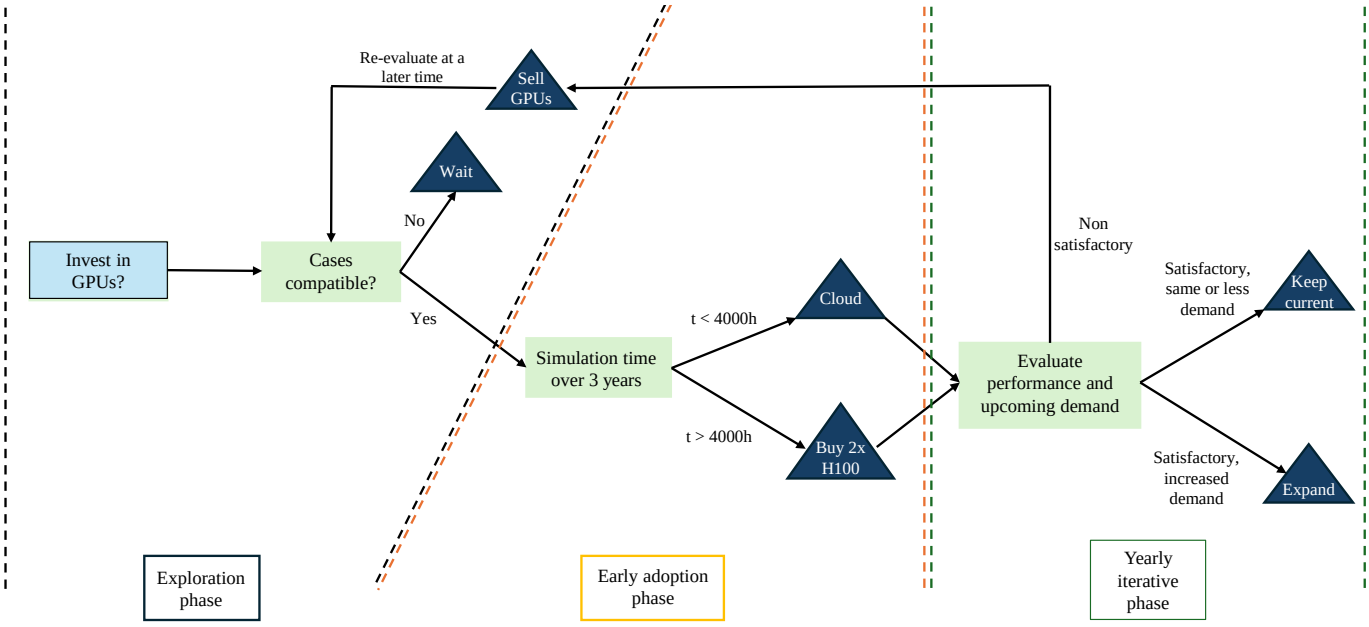

Figure 5.1: Strategic implementation decision tree.

The speedups from running CFD simulations on GPUs can be leveraged in different ways to add maximum value to the engineering process. They can, for example, be used to allow an engineer to test more design iterations of a component in the design phase, or to validate how multiple components work together in a shorter time

frame. They can also be used to increase simulation fidelity, as intensive simulations that would normally take an unreasonably long time to complete on CPUs can be completed in a reasonable time frame on GPUs. An example is turbulence modeling, where Large Eddy Simulations (LES) are more computationally expensive, but can be used to model turbulence with higher accuracy than RANS simulations in some cases.

5.1 Conclusion

This study shows that the Fluent GPU solver outperforms the CPU solver with regard to time, cost, power consumption, and convergence. Furthermore, it provides accurate results for the tested cases. It does, however, lack some features that are commonly used in aerospace applications, and the GPU solver can not replace the CPU solver completely in its current state.

The key findings are that 2x H100 cards are equivalent to 450-750 cpu cores in simulation capacity per iteration, that the GPU solver converges all residuals to 1e-04 in 27-73% fewer iterations, the energy consumption per iteration is reduced by 88-94%, the cost for running simulations on the Rescale cloud is reduced by 83-91% and the total cost of ownership to upgrade a local cluster with a GPU system instead of a CPU system of equivalent simulation capacity is reduced by 48-67%.
The Fluent native GPU solver is shown to be mature enough that it is possible to move into the early adoption phase of CFD on GPUs, either through cloud simulations or through a smaller-scale GPU purchase accounting for a minority of the total local cluster simulation capacity. In conclusion, the GPU solver should be used in all cases that it supports.

5.2 Further research

Further research on this topic could focus on expanding the range of cases in which the GPU solver has been benchmarked on and incorporate additional physics models and solver settings. LES simulations, for instance, could benefit from running on the GPU solver. It could also be worthwhile to dig further into how well the GPU and CPU solver matches each other in terms of simulation accuracy, and also how accurately cases originally modeled in CFX can be modeled with the Fluent GPU solver. The Fluent GPU solver could also be compared against other commercial GPU solvers. It could also be evaluated how the impact of the GPU solver speedups affect the engineering process and study how they can be leveraged to add maximum value to a business.

Bibliography

[1] J. Hess and A. Smith, "Calculation of potential flow about arbitrary bodies," *Progress in aerospace sciences*, vol. 8, pp. 1–138, 1967.

[2] W. Slagter. (2025) Ansys speeds up volvo ex90 aerodynamics simulations with nvidia gpu-accelerated cfd. ANSYS Inc. [Online]. Available: https://www.ansys.com/blog/ansys-speeds-up-volvo-ex90-aerodynamics-simulations

[3] N. Alarcon. (2019) Volkswagen accelerates aerodynamics concept design with nvidia v100 gpus on aws. NVIDIA Corporation. [Online]. Available: https://developer.nvidia.com/blog/volkswagen-accelerates-aerodynamics-concept-design-with-nvidia-v100-gpus-on-aws/

[4] C. Porter and N. Krishnamoorthy. (2022) The computational fluid dynamics revolution driven by gpu acceleration. NVIDIA Corporation. [Online]. Available: https://developer.nvidia.com/blog/computational-fluid-dynamics-revolution-driven-by-gpu-acceleration/

[5] EDR & Medeso. [Online]. Available: https://edrmedeso.com/

[6] Edr & medeso ab. Allabolag.se. [Online]. Available: https://www.allabolag.se/foretag/edr-medeso-ab/v%C3%A4ster%C3%A5s/datorer-kringutrustningar/2K3GXCDI5YCUH

[7] (2025) Gkn sweden, om oss. GKN Aerospace Sweden. [Online]. Available: https://www.gknaerospace.com/se//

[8] (2022) Unleashing the full power of gpus for ansys fluent, part 1. ANSYS Inc. [Online]. Available: https://www.ansys.com/blog/unleashing-the-full-power-of-gpus-for-ansys-fluent/

[9] Z. Cooper-Baldock, B. Vara Almirall, and K. Inthavong, "Speed, power and cost implications for gpu acceleration of computational fluid dynamics on hpc systems," *Supercomputing Asia*, 2024.

[10] "Ansys fluent native multi-gpu solver: Cfd validation studies in version 23r2," ANSYS Inc., Tech. Rep., 2023.

[11] "Speed and accuracy: First-of-its-kind broad-spectrum cfd solver built natively on gpus," ANSYS Inc., Tech. Rep., 2022.

[12] (2024) What is reynolds number? SimScale. [Online]. Available: https://www.simscale.com/docs/simwiki/numerics-background/what-is-the-reynolds-number/

[13] (2021) Mach number. NASA. [Online]. Available: https://www.grc.nasa.gov/www/k-12/airplane/mach.html

[14] H. K. Versteeg and W. Malalasekera, *An Introduction to Computational Fluid Dynamics*. Pearson Education Limited, 2007.

[15] H. Jasak, "Error analysis and estimation for the finite volume method with applications to fluid flows," Ph.D. dissertation, Imperial College London, 1996.
[16] *Ansys Fluent Theory Guide*, Ansys Inc., 2025.
[17] T.-R. Teschner. (2024) How to write a cfd library: The sparse matrix class. cfd.university. [Online]. Available: https://cfd.university/learn/how-to-compile-write-and-use-cfd-libraries-in-c/how-to-write-a-cfd-library-the-sparse-matrix-class/
[18] F. R. Menter, "Improved two-equation k-omega turbulence models for aerodynamic flows," *NASA Technical Memorandum*, 1992.
[19] W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," 1997. [Online]. Available: https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF
[20] Add or subtract floating point numbers (ieee 754). numeral-systems. Https://numeral-systems.com/ieee-754-add/.
[21] R. Farber, "Cuda application design and development," 2012. [Online]. Available: https://www.sciencedirect.com/book/9780123884268/cuda-application-design-and-development/
[22] I. Peng, "Introduction: Gpu architecture," KTH, 2024.
[23] "Nvidia h100 tensor core gpu architecture," NVIDIA Corporation, 2023.
[24] *Ansys Fluent User's Guide*, Ansys Inc., 2025.
[25] M. Witkowski, A. Oleksiak, T. Piontek, and Węglarz, "Practical power consumption estimation for real life hpc applications," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 208–217, 2013.
[26] J. Ludema and M. Nossokoff. (2024) Energy efficiency is driving many hpc users to the cloud. Hyperion Research. [Online]. Available: https://d1.awsstatic.com/products/ec2/hpc/AWS_NVIDIA_Hyperion_Research_Energy_Efficiency_Cloud_Whitepaper.pdf
[27] (2025) Primary energy consumption by region. Statista. [Online]. Available: https://www.statista.com/statistics/263457/primary-energy-consumption-by-region/
[28] T. Stobierski. (2019) How to do a cost-benefit analysis & how to do it. [Online]. Available: https://online.hbs.edu/blog/post/cost-benefit-analysis
[29] S. Pugh, "Concept selection: a method that works," *Proceedings of International Conference on Engineering Design*, pp. 497–506, 1981.
[30] (2024) Total cost of ownership (tco) – cheat sheet. Supply Chain Today. [Online]. Available: https://www.supplychaintoday.com/total-cost-of-ownership-tco-cheat-sheet/
[31] W. Slagter. (2024) Understanding the total cost of ownership in hpc and ai systems. ANSYS. [Online]. Available: https://www.ansys.com/blog/understanding-total-cost-ownership-hpc-ai-systems
[32] N. Rasmussen, "Calculating total cooling requirements for data centers," 2017. [Online]. Available: https://download.schneider-electric.com/files?p_Doc_Ref=SPD_NRAN-5TE6HE_EN&p_enDocType=White+Paper/
[33] V. Vikhorev, V. Chernoray, O. Thulin, S. Deshpande, and J. Larsson, "Detailed experimental study of the flow in a turbine rear structure at engine-realistic flow conditions," *Journal of Turbomachinery*, vol. 143, no. 9, 2021.

[34] S. Karl, T. Bykerk, and M. Laureti, "Design of a truncated ideal nozzle for a re-usable first stage launcher," *3rd International Conference on High-Speed Vehicle Science and Technology (HiSST)*, pp. 1–4, 2024.

[35] Exxact Corporation. [Online]. Available: https://configurator.exxactcorp.com/configure/TS2-145302459-ANS

[36] L. Smith. (2023) Ultimate amd genoa-x workstation review. StorageReview. [Online]. Available: https://www.storagereview.com/review/ultimate-amd-genoa-x-workstation-review

Appendix A

Supported physics models

• **Solver**	**CPU**	**GPU**
Pressure-Based	♦	♦
Density-Based	♦	◊
Absolute Velocity Formulation	♦	♦
Relative Velocity Formulation	♦	◊
Time-Steady	♦	♦
Time-Transient	♦	♦

• **Density Properties**	**CPU**	**GPU**
Piecewise-Linear	♦	♦
Piecewise-Polynominal	♦	♦
Polynominal	♦	♦
Expression	♦	♦
Compressible-Liquid	♦	♦
Ideal-Gas	♦	♦
Incompressible-Ideal-Gas	♦	♦
Boussinesq	♦	♦
Constant	♦	♦
RGP-Table	♦	◊
Real-Gas-Soave-Redlich-Kwong	♦	◊
Real-Gas-Peng-Robinson	♦	◊
Real-Gas-Aungier-Redlich-Kwong	♦	◊
Real-Gas-Redlich-Kwong	♦	◊
Real-Gas-Nist	♦	◊

• **Physics model**	**CPU**	**GPU**
Multiphase	♦	♦
Volume of fluid	♦	♦
Mixture	♦	◊
Wet Steam	♦	◊
Eulerian Inhomogeneous Model	♦	◊
Energy	♦	♦

Viscous	⧫	⧫
Inviscid	⧫	◊
Laminar	⧫	⧫
Spalart-Allmaras	⧫	◊
k-epsilon	⧫	⧫
k-omega	⧫	⧫
Transition k-kl-omega	⧫	◊
Transition SST	⧫	◊
Reynolds Stress	⧫	◊
Scale-Adaptive Simulation	⧫	◊
Detatched Eddy Simulation	⧫	◊
Large Eddy simulation	⧫	⧫
Radiation	⧫	⧫
Rosseland	⧫	◊
P1	⧫	◊
Discrete Transfer	⧫	◊
Surface to Surface	⧫	⧫
Discrete Ordinates	⧫	⧫
Monte Carlo	⧫	◊
Heat exchanger	⧫	⧫
Dual Cell Model	⧫	⧫
Ungrouped Macro Model	⧫	⧫
Macro Model Group	⧫	⧫
Species	⧫	⧫
Species Transport	⧫	⧫
Diffusion Energy Source	⧫	⧫
Full Multicomponent Diffusion	⧫	◊
Thermal Diffusion	⧫	◊
Non-premixed Combustion	⧫	◊
Premixed Combustion	⧫	◊
Partially Premixed Combustion	◊	⧫
Composition PDF Transport	⧫	◊
Discrete phase	⧫	⧫
Saffman Lift Force	⧫	◊
Virtual Mass Force	⧫	◊
Pressure Gradient Force	⧫	◊
Erotion/Accretion	⧫	◊
DEM Collision	⧫	◊
Stochastic Collision	⧫	◊
Breakup	⧫	◊
Solidification & Melting	⧫	◊
Virtual Blade Model	⧫	◊
Acoustics	⧫	⧫
Wave Equation	⧫	◊

Ffowcs Williams & Hawkings (FW-H)	♦	♦
Structure	♦	◊
Eulerian Wall Film	♦	◊
Potential/Electrochemistry	♦	◊
Battery Model	♦	◊
Ablation	♦	◊
• **Pressure-Velocity Coupling schemes**	**CPU**	**GPU**
Simple	♦	♦
SimpleC	♦	♦
PISO	♦	◊
Coupled (Steady state)	♦	♦
Coupled (Transient)	♦	◊
• **Dimension**	**CPU**	**GPU**
2D	♦	◊
3D	♦	♦

Table A.1: Physics models and solution schemes available in the CPU and GPU solver in Ansys Fluent 2025R1

Appendix B

Sparse matrix structure

A-matrix for general $A\phi = B$ with 4 cells.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \text{ With vector } \phi \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{bmatrix} \text{ And vector B } \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

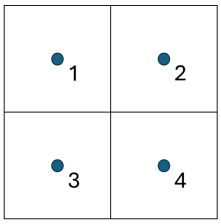

Figure B.1: 2x2 equidistant mesh with 4 cells

For a 2x2 equidistant mesh with 4 cells the A matrix contains the following non-zeros:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & 0 & A_{24} \\ A_{31} & 0 & A_{33} & A_{34} \\ 0 & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

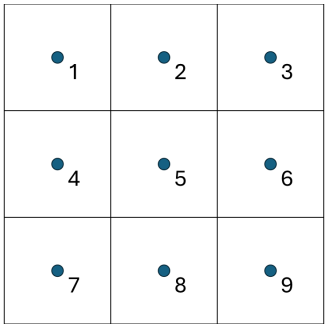

Figure B.2: 3x3 equidistant mesh with 9 cells

For a 3x3 equidistant mesh with 9 cells the A matrix contains the following coefficients:

$$\begin{bmatrix} A_{11} & A_{12} & 0 & A_{14} & 0 & 0 & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & A_{25} & 0 & 0 & 0 & 0 \\ 0 & A_{32} & A_{33} & 0 & 0 & A_{36} & 0 & 0 & 0 \\ A_{41} & 0 & 0 & A_{44} & A_{45} & 0 & A_{47} & 0 & 0 \\ 0 & A_{52} & 0 & A_{54} & A_{55} & A_{56} & 0 & A_{58} & 0 \\ 0 & 0 & A_{63} & 0 & A_{65} & A_{66} & 0 & 0 & A_{69} \\ 0 & 0 & 0 & A_{74} & 0 & 0 & A_{77} & A_{78} & 0 \\ 0 & 0 & 0 & 0 & A_{85} & 0 & A_{87} & A_{88} & A_{89} \\ 0 & 0 & 0 & 0 & 0 & A_{96} & 0 & A_{98} & A_{99} \end{bmatrix}$$

DEPARTMENT OF Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY