GPU-Accelerated Finite Volume Particle Simulation of Free Jet Deviation by Multi-jet Rotating Pelton Runner

Thèse N° 7256

Présentée le 18 octobre 2019

à la Faculté des sciences et techniques de l'ingénieur
Laboratoire de machines hydrauliques
Programme doctoral en energie

pour l'obtention du grade de Docteur ès Sciences

par

Siamak ALIMIRZAZADEH

Acceptée sur proposition du jury

Prof. K. A. J. Mulleners, présidente du jury
Prof. F. Avellan, directeur de thèse
Prof. C. Tournier, rapporteur
Dr J. Decaix, rapporteur
Prof. M. Picasso, rapporteur

2019

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

“You can be the strongest man in the world, and you're still going to go through problems. And that's the one thing with me. I don't ever want my medals to define who I am. What I'm doing now to have a chance to save a life which is way bigger than ever winning a gold medal.”

– Michael Phelps
The most decorated Olympian of all time

TO Michael Phelps ...

Acknowledgment

To my Ph.D. advisor, Prof. François Avellan: I would like to express my sincere gratitude for all your continuous support, trust, and patience during my doctoral study. Thank you for providing me with the excellent opportunity to learn and grasp high-quality research methodologies at an impressive world-leading laboratory – LMH. My special thanks also go to my jury members Prof. Tournier, Prof. Picasso, Dr. Decaix, and jury president Prof. Mulleners, for the time they spent to read this thesis as well as their constructive suggestions and enriching discussions.

To my life companion, my soul mate, my beautiful lady, and my LOVE, Shahrzad: I am grateful to your honesty, loyalty, patience and all the emotional support. Thank you for letting me grow. Your role in my life is ineffable. Forgive me for all I couldn't do and give me more time. The future is awesome !

To my father, my upholder: I appreciate your continuous support in all the hard moments of my life. You were always there to support whenever a new enigma was challenging me. With great joy in my heart, I thank you!

I would like to express my special thanks to Dr. Audrey Maertens for all her extensive and helpful feedback as well as sharing her deep and immense knowledge in fluid mechanics and numerical modeling. Her in-depth comments had a significant impact on the quality of this research. That was indeed an excellent opportunity for me to learn a lot from her. She was awesome.

My sicnere thanks also go to Dr. Ebrahim Jahanbakhsh, the pioneer developer of the SPHEROS and leader of GPU-SPHEROS project. His experience in programming and numerical modeling, indeed, played an essential role in the implementation of such a massively parallel solver. I thank Dr. Christian Vessaz for sharing all his experience and extensive knowledge in the Pelton turbine field.

I am grateful to Dr. Kiyohito Tani and Mr. Takashi Kumashiro from Hitachi-Mitsubishi Hydro Corporation, the research and development part for scientific collaboration and their trust. I loved Japanese culture during my trip to Japan. I also like their reliable products. Tokyo is indeed the most awesome city on this planet that I have ever visited.

I also would like to express my sincere thanks to Dr. Mark Harris and Dr. Peter Messmer from NVIDIA for their technical support. On behalf of GPU-SPHEROS development team, I also thank NVIDIA for their GPU grant program.

I thank Luuk Brummans from Delft university for his great job as an internship student here at LMH. I enjoyed supervising his work.

I would like to show my thank to Isabelle Stoudmann Schmutz for her incredibly gracious posture. She was always kind and continuously supportive! I also would like to appreciate all the time I spent with Sebastián (Leguizamón) and Joao (Gomes Pereira) with having stimulating discussions. Our debates were fascinating and our interaction was great, not always smooth. Seb was beaten in our last chess game and never played the next one !!

I thank Andres and Elena for helping me to wirte German and Italian Abstratcs of my thesis. It was a great support. I also would like to thank Dr. Philippe Cerutti for his support as an IT engineer.

With special mention to Ali Amini, David Buzzi, Pascal Clausen, Arthur Favrel, Federico Martinez, Alexis Papagiannopoulos, Simon Pasche, Masashi Sakamoto, Outi Supponen, and Keita Yamamoto. I enjoyed all the moments and chats I had with you.

This research has been performed within the Swiss Competence Center on Energy Research, Supply of Electricity (SCCER - SoE), GPU-SPHEROS project Grant No. 17568.1 PFEN IW, with the support of the Swiss Commission for Technology and Innovation Kommission für Technologie und Innovation (CTI/KTI). The research was also supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID s777.

Thank you for all your encouragement !

Siamak

15 August 2019

Lausanne

Abstract

Numerical simulation of Pelton turbine hydrodynamics is helpful to identify the energy loss mechanisms in the runner and minimize their effect. However, it is a challenging task that involves handling the unsteady free surface flow and moving boundaries requiring dynamic mesh approach, as well as run-time local grid refinements at the interphase. Unlike the mesh-based methods, the Lagrangian particle-based methods are robust in handling free surface problems with moving boundaries such as Pelton turbine flow.

Within the framework of the present research, the 3-D Finite Volume Particle Method (FVPM) has been developed and accelerated on Graphics Processing Unit (GPU). FVPM is locally conservative and consistent, employing an Arbitrary Lagrangian-Eulerian (ALE) approach for particle motion to achieve a reasonably uniform particle distribution. The method is based on spherical-support top-hat kernels in which the particle interaction vectors are computed and used to weigh the conservative flux exchange. To capture the turbulence, the standard and realizable k-ε as well as k-ω Shear Stress Transport (SST) turbulence models have been implemented and integrated into ALE-based FVPM. The wall function approach has been used for near-wall turbulence computations. The solver is called GPU-SPHEROS and has been implemented from scratch in the CUDA C++ parallel computing platform.

All the parallel algorithms and data structures have been designed specifically for the GPU many-core architecture. The roofline analysis method has been utilized to assess the performance of the CUDA kernels and define the appropriate optimization strategies. In particular, the neighbor search algorithm, accounting for almost a third of the overall computation time, features an efficient Space-Filling Curve (SFC) as well as an optimized octree construction and traverse procedure. The memory-bound interaction vector computation, accounting for almost two-thirds of the overall computation time, features fixed-size memory pre-allocation and an efficient data ordering to reduce memory transactions and avoid the cost of dynamic memory operations. A speedup by a factor of almost six times has been achieved for a single NVIDIA® Tesla™ P100 16GB GPU with GP100 Pascal architecture vs. a dual-setup Broadwell Intel® Xeon® E5-2690 v4 CPU node with 28 total physical cores.

Once GPU-SPHEROS validated, it is used for jet interference investigation in a six-jet Pelton turbine as an industrial-size practical application. The numerical simulations have been performed at eight operating points ranging from $N / N_{BEP} = 89\%$ to $N / N_{BEP} = 131\%$, where N is the runner rotational speed and BEP is the Best Efficiency

Point. It is shown by the numerical results that a significant torque and efficiency drop occurs at high speed factors due to jet interference, whereas large load fluctuations caused by jet disturbance can occur at about any $N \neq N_{BEP}$. Compared to the available experimental data provided by Hitachi-Mitsubishi Hydro Corporation, the torque and efficiency trends, as well as the range of the specific speed in which the jets interfere, are well-predicted, which provides confidence in the use of the GPU-SPHEROS for the design optimization of Pelton turbines. All the multi-jet Pelton turbine computations have been performed on Piz Daint, a GPU-powered supercomputer with 5'704 GPU nodes, developed and operated by Swiss National Supercomputing Centre - CSCS.

Keywords: Pelton turbine, bucket, jet interference, torque, efficiency, GPU-SPHEROS, parallelization, GPU, CUDA, CUDA kernel, memory access, memory-bound, compute-bound, coalesced memory, flops, roofline performance model, performance optimization

Résumé

La simulation numérique des écoulements dans les turbines Pelton permet d'identifier les mécanismes responsables des pertes énergétiques dans la roue et de les minimiser. Cependant, l'écoulement étant caractérisé par une surface libre et des interfaces en mouvement, sa simulation nécessite l'utilisation d'un maillage dynamique, avec des raffinements aux interfaces en temps réel. Alternativement, des méthodes particulaires peuvent être utilisées pour éviter des difficultés liées au maillage. En raison de leur formulation lagrangienne ces méthodes sont en effet particulièrement robustes pour traiter les problèmes de surface libre avec des interfaces en mouvement tels qu'ils se peuvent se présenter dans les turbines Pelton.

Dans le cadre de la recherche présentée ici, la méthode des Volumes Finis Particulaires (FVPM) 3-D a été développée et accélérée sur Graphics Processing Unit (GPU) dans la plate-forme de calcul parallèle CUDA. FVPM est localement conservative et consistante, utilisant une approche Arbitrairement Lagrangienne ou Eulérienne (ALE) pour le mouvement des particules afin d'obtenir une distribution de particules raisonnablement uniforme. La méthode est basée sur l'utilisation de noyaux à support sphérique et des vecteurs d'interaction sont utilisés pour pondérer de l'échange les flux entre particules voisines. Pour saisir les caractéristiques de l'écoulement moyen, les modèles de turbulence k-ε standard et réalisable ainsi que k-ω SST ont été implémentés et intégrés dans le code, GPU-SPHEROS.

Tous les algorithmes parallèles et les structures de données ont été conçus spécifiquement pour l'architecture multi-cœur des GPU. Un modèle de performance de « roofline » a été utilisé pour évaluer la performance des noyaux CUDA et définir des stratégies d'optimisation appropriées. En particulier, l'algorithme de recherche des voisins, qui représente près d'un tiers du temps de calcul total, est basé sur l'utilisation d'une courbe de remplissage de l'espace (SFC), méthode reconnue pour son rendement, ainsi que d'une procédure de construction optimisée d'octree. Le calcul des vecteurs d'interaction, limité par la taille de la mémoire et qui représente près des deux tiers du temps de calcul total, est doté d'une pré-allocation de mémoire de taille fixe et d'un ordonnancement des données permettant de réduire les transactions et le coût des opérations de mémoire dynamiques. Le temps de calcul a ainsi été divisé par près de six sur un seul GPU NVIDIA® Tesla™ P100 de 16 Go avec architecture GP100 Pascal par rapport à un nœud équipé de deux CPU Broadwell Intel® Xeon® E5-2690 v4 avec 28 cœurs physiques en tout, sans hyper-threading.

Une fois le solveur validé, il a été utilisé pour une application industrielle, à savoir l'étude des interférences de jets dans une turbine Pelton multi-jets. Les simulations numériques ont été effectuées pour huit points de fonctionnement allant de $N / N_{BEP} = 89\%$ à $N / N_{BEP} = 131\%$, où N est la vitesse de rotation de la roue en min^{-1} et BEP est le point de meilleure rendement. Les résultats numériques montrent que le couple et le rendement baissent drastiquement à haute vitesse de rotation, en raison de l'interférence des jets, alors que la perturbation des jets, à toute vitesse différente de N_{BEP}, donne lieu à une grande fluctuation de la charge. Comparées aux mesures expérimentales effectuées par Hitachi-Mitsubishi Hydro Corporation, les tendances observées pour le couple et le rendement, ainsi que la plage de vitesses dans laquelle les jets interfèrent, concordent avec les mesures expérimentales. Cette validation confirme que GPU-SPHEROS peut être un outil approprié pour optimiser la conception des turbines Pelton. Tous les calculs ont été effectués sur Piz Daint, le supercalculateur équipé de 5'704 nœuds GPU, exploité par le Centre national suisse de supercalcul - CSCS.

Mots-clés: Turbine Pelton, auget, interférence de jet, couple, rendement, GPU-SPHEROS, parallélisation, GPU, CUDA, noyau CUDA, accès mémoire, limité par la mémoire, limité par le calcul, mémoire « coalesced », flops, modèle roofline, optimisation des performances

Abstrakt

Die numerische simulation der hydrodynamik in Pelton turbinen dient dazu, die mechanismen, die zu energieverlusten im laufrad führen, zu identifizieren und deren effekte zu minimieren. Aufgrund der instationären strömung mit einer freien oberfläche und beweglichen grenzen ist dies jedoch eine anspruchsvolle aufgabe, die eine dynamische definition des berechnungsrasters und eine lokale Laufzeit-Maschenverfeinerung an den schnittstellen voraussetzen. Im gegensatz zu gitterbasierten methoden erlauben partikelbasierte Lagrange ansätze eine robuste Handhabung der freien oberflächen mit beweglichen grenzen, wie diese in Pelton strömungen zu finden sind.

Im rahmen dieses forschungsprojektes wurde die dreidimensionale partikelbasierte Finite Volumen Methode (3-D Finite Volume Particle Method oder FVPM) weiterentwickelt und auf Grafikkarten (Graphics Processing Units oder GPU) beschleunigt. FVPM ist lokal konservativ und konsistent und verwendet einen arbitrary Lagrangian-Eulerian Ansatz (ALE) für partikelbewegungen, um eine angemessene Partikelverteilung zu erreichen. Die Methode basiert auf sphärisch top-hat Kernel, in welchen die partikelinteraktionsvektoren berechnet und zur gewichtung der konservativen Flüsse verwendet werden. Um die turbulenz zu erfassen wurden standard k-ε und k-ω Shear Stress Transport (SST) Modelle in die ALE-basierte FVPM integriert. Für die bestimmung von wandnahen turbulenzen wurde eine wandfunktion angewandt. Der Rechner wird GPU-SPHEROS genannt und von grund auf in der CUDA C++ parallelen rechnerplattform implementiert.

Alle parallelen algorithmen und datenstrukturen wurden spezifisch für die GPU Vielrechnerarchitektur entworfen. Die roofline analyse methode wurde für die leistungsbeurteilung der CUDA kernel und für die definition der geeigneten optimierungsstrategien verwendet. Vor allem der nachbarn suchen algorithmus, welcher fast ein drittel der gesamten rechenzeit einnimmt, beinhaltet eine effizienten Space-Filling Curve (SFC) und eine optimierte octree konstruktion und traverse prozedur. Zur berechnung der speichergebundenen interaktionsvektoren, welche fast zwei drittel der gesamten rechenzeit beansprucht, arbeitet die methode mit einer festen speicherplatzvorvergabe und einem effizienten datenordungsprinzip, um speichertransaktionen zu reduzieren und kosten für dynamische speicheroperationen zu verhindern. Eine fast sechsfache beschleunigung wurde erreicht für eine einzelne NVIDIA® Tesla™ P100 16GB GPU mit GP100 Pascal architektur, gegenüber einer dual-setup Broadwell Intel® Xeon® E5-2690 v4 CPU node mit insgesamt 28 physischen rechenkernen.

Nach der validierung von GPU-SPHEROS wurde der rechner für die untersuchung in einer einer 6-strahligen Pelton turbine im rahmen einer praktischen industriellen anwendung verwendet. Die numerischen simulationen wurden an acht betriebspunkten von $N\ /\ N_{BEP} = 0.89$ bis $N\ /\ N_{BEP} = 1.39$ durchgeführt, wobei N die drehgeschwindigkeit des schaufelrades und BEP der punkt des besten wirkungsgrades (Best Efficiency Point - BEP) ist. Es konnte gezeigt werden, dass bei hohen geschwindigkeitskoeffizienten aufgrund der wechselwirkungen zwischen den wasserstrahlen eine beträchtliche drehmoment und wirkungsgradabnahme auftritt, während grosse lastvariationen durch eine beeinträchtigung der strahlen bei beliebigen N_{BEP}, werten beobachtet werden können. Aufgrund von verfügbaren messdaten von Hitachi-Mitsubishi Hydro Corporation werden die drehmoment und wirkungsgradtrends sowie die bandbreite der spezifischen geschwindigkeiten, bei welchen die strahlen interferieren, gut vorausgesagt. Dies schafft vertrauen in die anwendung von GPU-SPHEROS für die optimierung von Pelton turbinendesigns. Alle berechnungen von mehrstrahligen Pelton turbinen wurden auf Piz-Daint durchgeführt, einem GPU-gepowertem supercomputer mit 5'704 GPU knoten, entwickelt und betrieben durch das Swiss National Supercomputing Centre – CSCS.

Schlüsselwörter: Pelton turbinen, Eimer, Jet-Interferenz, Drehmoment, Effizienz, GPU-SPHEROS, Parallelisierung, GPU, CUDA, CUDA kernel, Speicherzugriff, Speicher gebunden, rechner gebunden, verschmolzenes Gedächtnis, flops, roofline Leistungsmodell, Leistungsoptimierung

Astratto

La simulazione numerica dell'idrodinamica delle turbine Pelton è utile per identificare i meccanismi che causano le perdite energetiche nella girante e per la minimizzazione dei conseguenti effetti. Tuttavia per poter correttamente simulare il flusso di acqua, che è caratterizzato da una superficie a pelo libero instabile e da un'ampia deformazione dei bordi, è necessario utilizzare una mesh dinamica e una rifinizione locale della mesh nei bordi a ogni instante temporale.

In alternativa, metodi particolati possono essere utilizzati per sormontare i problemi legati alla mesh. Grazie alla loro natura Lagrangiana, questi metodi sono particolarmente robusti per risolvere i problemi riguardanti superfici a pelo libero con ampie deformazioni dei bordi come nel caso del flusso in una turbina Pelton.

Nel presente studio, il metodo dei volumi finiti particolati (FVPM) è stato sviluppato e velocizzato sull' unità di elaborazione grafica (GPU) nella piattaforma di calcolo parallelo CUDA. FVPM è localmente conservativo e consistente, utilizza un approccio arbitrariamente Lagrangiano o Euleriano (ALE) per il movimento delle particelle con lo scopo d'ottenerne una distribuzione ragionevolmente uniforme.

Il metodo si basa sull'utilizzo di kernels a supporto sferico nei quali i vettori d'interazione delle particelle sono calcolati e utilizzati per ponderare gli scambi di flusso conservativi tra particelle vicine. Per determinare le caratteristiche del flusso medio, i modelli di turbolenza k-ε standard e k-ω SST sono stati implementati e integrati nel codice numerico, che prende il nome di GPU-SPHEROS.

Tutti gli algoritmi paralleli e le strutture dei dati sono stati progettati nello specifico per l'architettura multi-core della GPU. Un modello roofline è stato utilizzato per valutare la performance dei kernels CUDA e per attuare delle strategie d'ottimizzazione appropriate. In particolare, l'algoritmo che cerca le particelle vicine, che costituisce all'incirca un terzo del tempo totale di calcolo, si basa sull'utilizzo di una curva di riempimento dello spazio (SFC), metodo noto per la sua efficacia, e di una procedura per la costruzione ottimizzata dell'octree. Il calcolo dei vettori d'interazione, limitato dalla memoria e che rappresenta all'incirca i due terzi del tempo totale di calcolo, vanta una pre- allocazione della memoria di taglia fissa e un'organizzazione dei dati che permette di ridurre le transazioni di memoria e il costo delle operazioni dinamiche della memoria. Il tempo di calcolo è stato così ridotto di circa sei volte in un solo GPU NVIDIA® Tesla™ P100 di 16 Go con un'architettura GP100 Pascal rispetto a un nodo dotato di due CPU Broadwell Intel® Xeon® E5-2690 v4 con 28 nuclei fisici e senza hyper-threading.

Una volta validato, il solver è stato utilizzato per lo studio delle interferenze dei getti d'acqua in una turbina Pelton poligetto, per mostrarne l'efficacia in un' applicazione pratica su scala industriale. Le simulazioni numeriche sono state effettuate per otto punti di funzionamento, da $N\,/\,N_{BEP} = 0.89$ a $N\,/\,N_{\mathrm{BEP}} = 1.31$, dove N è la velocità di rotazione della girante in min^{-1} e N_{BEP} è la velocità nel punto di massimo rendimento. I risultati numerici mostrano che la coppia e il rendimento diminuiscono drasticamente quando la velocità di rotazione è elevata a causa dell'interferenza dei getti, mentre per tutte le velocità di rotazione diverse da N_{BEP} si può osservare un'importante oscillazione del carico a causa della perturbazione dei getti. Comparando questi risultati con le misure sperimentali effettuate da Hitachi-Mitsubishi Hydro Corporation, la tendenza osservata per la coppia e il rendimento, così come l'intervallo di velocità nel quale i getti possono interferire, sono in buon accordo con le misure sperimentali. Questa validazione conferma che GPU-SPHEROS può essere uno strumento adatto per ottimizzare la progettazione delle turbine Pelton. Tutti i calcoli sono stati effettuati su Piz Daint, un supercomputer dotato di 5704 nodi GPU, sviluppato e utilizzato dal Centro Svizzero di Calcolo Scientifico (CSCS).

Parole chiave: Turbina Pelton, pala, interferenza del getto, coppia, rendimento, GPU-SPHEROS, parallelizzazione, GPU, CUDA, Kernel CUDA; accesso alla memoria, limite della memoria, limite di calcolo, memoria "coalescente", flops, modello roofline, ottimizzazione delle performances.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ALE	Arbitrary Lagrangian-Eulerian
BC	Boundary Condition
BEP	Best Efficiency Point
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
CPU	Central Processing Unit
CSCS	Centro Svizzero di Calcolo Scientifico (Swiss National Supercomputer Centre)
DNS	Direct Numerical Simulation
DP	Double Precision
EPFL	École Polytechnique Fédérale de Lausanne
Flop	Floating-Point Operation
Flops	Floating-Point Operation per second
FPM	Finite Particle Method
FVM	Finite Volume Method
FVPM	Finite Volume Particle Method
GPU	Graphics Processing Unit
GPGPU	General-Purpose computing on Graphics Processing Units
HT	Hyper-Threading
IEC	International Electrotechnical Commission
LES	Large Eddy Simulation
OI	Operational Intensity
RANS	Reynolds–Averaged Navier–Stokes
SFC	Space–Filling Curve
SGS	Sub-Grid Scale
SIMT	Single-Instruction Multiple-Thread
SM	Streaming Multi-Processor
SPH	Smoothed Particle Hydrodynamics
SST	Shear Stress Transport
VOF	Volume of Fluid

Latin Letters

a	sound speed	$(\mathrm{m \cdot s^{-1}})$
B_2	bucket width	(m)

$\mathbf{B}$	boundary interaction vector	(m^2)
C	absolute flow velocity	$(\mathrm{m}\cdot\mathrm{s}^{-1})$
Cu	tangential absolute flow velocity component	$(\mathrm{m}\cdot\mathrm{s}^{-1})$
$\mathbf{C}$	flow velocity vector	$(\mathrm{m}\cdot\mathrm{s}^{-1})$
$\mathbf{C}'$	velocity fluctuations	$(\mathrm{m}\cdot\mathrm{s}^{-1})$
$\mathbf{C}_{\mathrm{par}}$	velocity component parallel to the wall	$(\mathrm{m}\cdot\mathrm{s}^{-1})$
C_2	jet velocity	$(\mathrm{m}\cdot\mathrm{s}^{-1})$
c_v	nozzle efficiency	$(-)$
D_1	runner pitch diameter	(m)
D_2	jet diameter	(m)
D_k	turbulence destructionrate	$(\mathrm{kg}\cdot\mathrm{m}^{-1}\cdot\mathrm{s}^{-3})$
d	distance from the nearest wall	(m)
E	specific energy	$(\mathrm{m}^2\cdot\mathrm{s}^{-2})$
E_t	transformed specific energy	$(\mathrm{m}^2\cdot\mathrm{s}^{-2})$
$\mathbf{F}$	flux function	$(*)$
$\mathbf{f}$	force	(N)
g	gravity acceleration magnitude	$(\mathrm{m}\cdot\mathrm{s}^{-2})$
$\mathbf{g}$	gravity acceleration vector	$(\mathrm{m}\cdot\mathrm{s}^{-2})$
H	head	(m)
h	particle smoothing length	(m)
$\mathbf{I}$	identity matrix	$(-)$
I_{turb}	turbulence intensity	$(-)$
k	turbulence kinetic energy	$(\mathrm{m}^2\cdot\mathrm{s}^{-2})$
L	characteristic length	(m)
m	mass	(kg)
N	rotational speed	(min^{-1})
N_{GPU}	number of used GPUs	$(-)$
n	rotational frequency	(s^{-1})
n_P	number of particles per jet diameter	$(-)$
$\mathbf{n}$	normal vector to the free surface	$(-)$
$\mathbf{n}_w$	normal vector to the wall	$(-)$
P	supplied power	$(\mathrm{kg}\cdot\mathrm{m}^2\cdot\mathrm{s}^{-3})$
P_h	hydraulic power	$(\mathrm{kg}\cdot\mathrm{m}^2\cdot\mathrm{s}^{-3})$
P_k	turbulence production	$(\mathrm{kg}\cdot\mathrm{m}^{-1}\cdot\mathrm{s}^{-3})$
P_t	transformed power	$(\mathrm{kg}\cdot\mathrm{m}^2\cdot\mathrm{s}^{-3})$
p	pressure	(Pa)
p_c	convergence rate	$(-)$
Q	overall discharge	$(\mathrm{m}^3\cdot\mathrm{s}^{-1})$
Q_t	transferred discharge	$(\mathrm{m}^3\cdot\mathrm{s}^{-1})$
Q_0	discharge per nozzle	$(\mathrm{m}^3\cdot\mathrm{s}^{-1})$
q	memory traffic	$(\mathrm{B}\cdot\mathrm{s}^{-1})$

$\mathbf{R}$	smoothing mass flux term	$(\mathrm{kg \cdot m^{2} \cdot s^{-3}})$
S	modulus of the mean rate of strain rate tensor	$(\mathrm{s^{-1}})$
S_e	elementary surface	$(\mathrm{m^{2}})$
S_f	output saving frequency	$(\mathrm{s^{-1}})$
$\mathbf{S}$	strain rate tensor	$(\mathrm{s^{-1}})$
$\mathbf{S}_e$	elementary surface area vector	$(\mathrm{m^{2}})$
$\mathbf{s}$	deviatoric stress tensor	$(\mathrm{N \cdot m^{-2}})$
T	torque	$(\mathrm{N \cdot m})$
t	time	(s)
t_s	simulation time	(s)
Δt	time step	(s)
U	peripheral flow velocity	$(\mathrm{m \cdot s^{-1}})$
$\mathbf{U}$	conserved variable	(*)
u^+	dimensionless velocity	(–)
V	volume	$(\mathrm{m^{-3}})$
W	relative flow velocity	$(\mathrm{m \cdot s^{-1}})$
w	application work	$(\mathrm{Flop \cdot s^{-1}})$
$\dot{\mathbf{x}}$	particle velocity vector	$(\mathrm{m \cdot s^{-1}})$
$\dot{\mathbf{x}}^{\mathrm{c}}$	velocity correction vector	$(\mathrm{m \cdot s^{-1}})$
$\mathbf{x}$	position vector	(m)
$\mathbf{X}_{\mathrm{O}}^{inlet}$	center of circular injected jet as inlet boundary	(m)
y^+	dimensionless wall distance	(–)
Δy	distance from the wall	(m)
z_0	number of nozzles	(–)
X, Y, Z	Conventional Cartesian coordinate components	(m)

Greek Letters

β	relative flow angle	(m)
$\mathbf{\Gamma}$	particle interaction vector	$(\mathrm{m^{2}})$
Δ	bucket loss factor	(–)
$\mathbf{\Delta}$	weight vector for flux exchange	$(\mathrm{m^{2}})$
δ	particle spacing	(m)
$\boldsymbol{\delta}$	Kronecker delta	(–)
ε	turbulence kinetic energy dissipation rate	$(\mathrm{m^{2} \cdot s^{-3}})$
η	global efficiency	(–)
θ	rotation angle	(deg)
μ	dynamic viscosity	$(\mathrm{kg \cdot m^{-1} \cdot s^{-1}})$
μ_t	eddy viscosity	$(\mathrm{kg \cdot m^{-1} \cdot s^{-1}})$
v	kinematic viscosity	$(\mathrm{m^{2} \cdot s^{-1}})$
ρ	density	$(\mathrm{kg \cdot m^{-3}})$

ρ_0	reference density	$(\mathrm{kg \cdot m^{-3}})$
σ_e	kernel summation function	$(-)$
Ψ	vorticity tensor	$(\mathrm{s^{-1}})$
ψ	FVPM test function	$(-)$
Ω	computational domain	$(\mathrm{m^3})$
ω	eddy frequency	$(\mathrm{s^{-1}})$

Subscripts

1	runner inlet reference section
$\bar{1}$	runner outlet reference section
2	jet reference section
b	boundary
e	elementary surface
eff	effective
i	i^{th} particle
j	j^{th} particle
ij	interface of i^{th} and j^{th} particles
ref	reference section
t	transformed
$turb$	turbulence
α	Cartesian coordinate
β	Cartesian coordinate

Dimensionless Numbers

$Re = \frac{\rho C L}{\mu}$	Reynolds number
$C_P = \frac{p - p_{ref}}{\frac{1}{2}\rho C_{ref}^2}$	pressure coefficient with respect to the reference condition
$k_{Cu} = \frac{\pi}{60}\frac{N D_1}{\sqrt{2gH}}$	speed factor
$k_T = \frac{T}{\frac{1}{2}\rho D_1^3 C_2^{\,2}}$	torque factor
$n_q = N\frac{Q^{0.5}}{H^{0.75}}$	unit specific speed

1 Introduction

1.1 Motivation

The Pelton turbine is an impulse turbine invented in 1880 by an American inventor, Lester Allan Pelton [1]. As of 2018, this machine type represents 8 and 17% of the world and Europe hydroelectric capacity (see Figure 1.1), respectively. After almost one and a half century of Pelton turbine evolution, designs still keep improving with extensive research carried out on different aspects from distributor and nozzle optimization to silt erosion and cavitation which all have an impact on the runner performance. Although an experimental investigation is a reliable approach for most aspects of hydropower design, it requires a lengthy and costly trial and error procedure. With the advent and development of modern Computational Fluid Dynamics (CFD) boosted by ever increasing available computing power, the design process can now benefit from numerical approaches by handling the costly trial and error design phase with CFD and then performing reliability assessment with a limited number of experiments, only in the final design phase. However, to deal with the process, a robust and reliable solver is required, to able to provide accurate enough results in a reasonable time.

Among existing numerical approaches, the most mesh-based numerical methods such as the Finite Volume Method (FVM) or Finite Element Method (FEM) can fulfill conservation and consistency simultaneously, although they have difficulty to deal with moving boundaries and the transient free surface formed during the complex interaction between the Pelton turbine components, i.e., jet-bucket and jet-jet. The Lagrangian particle-based methods can be an alternative approach to robustly cope with these difficulties. However, most of the particle-based methods suffer from a lack of conservation or consistency.

This research is aimed at developing a high-performance conservative and consistent particle-based solver for numerical simulation of free surface problems, especially those with large boundary motions. Such a solver should be able to robustly handle the jet-jet and jet-bucket interaction as well as tracking the complex free surface around the rotating buckets, robustly. To achieve this goal, the Finite Volume Particle Method (FVPM) has been selected as a conservative and consistent particle-based method. The method benefits from the desirable features of both conventional mesh-based FVM and particle-based Smoothed Particle Hydrodynamics (SPH) although the computational cost is higher due to FVPM more sophisticated algorithm. The Graphics Processing Unit (GPU) many-core architecture with

thread-level parallel programming capability can be utilized to parallelize the computations and improve the solver performance for industrial-size applications.

Numerical solvers can be used to investigate different loss mechanisms in a real-scale Pelton turbine and in fact, in a broader scope. The developed GPU-accelerated 3-D FVPM solver is then used to investigate a particular loss mechanism in the present research; the interference between the jets in a multi-jet Pelton turbine. With simulating a full-size six-jet Pelton runner at different off-design conditions, the solver provides unique information such as flow visualizations, pressure distributions, and torque time-histories, which are all valuable in further improving Pelton designs.

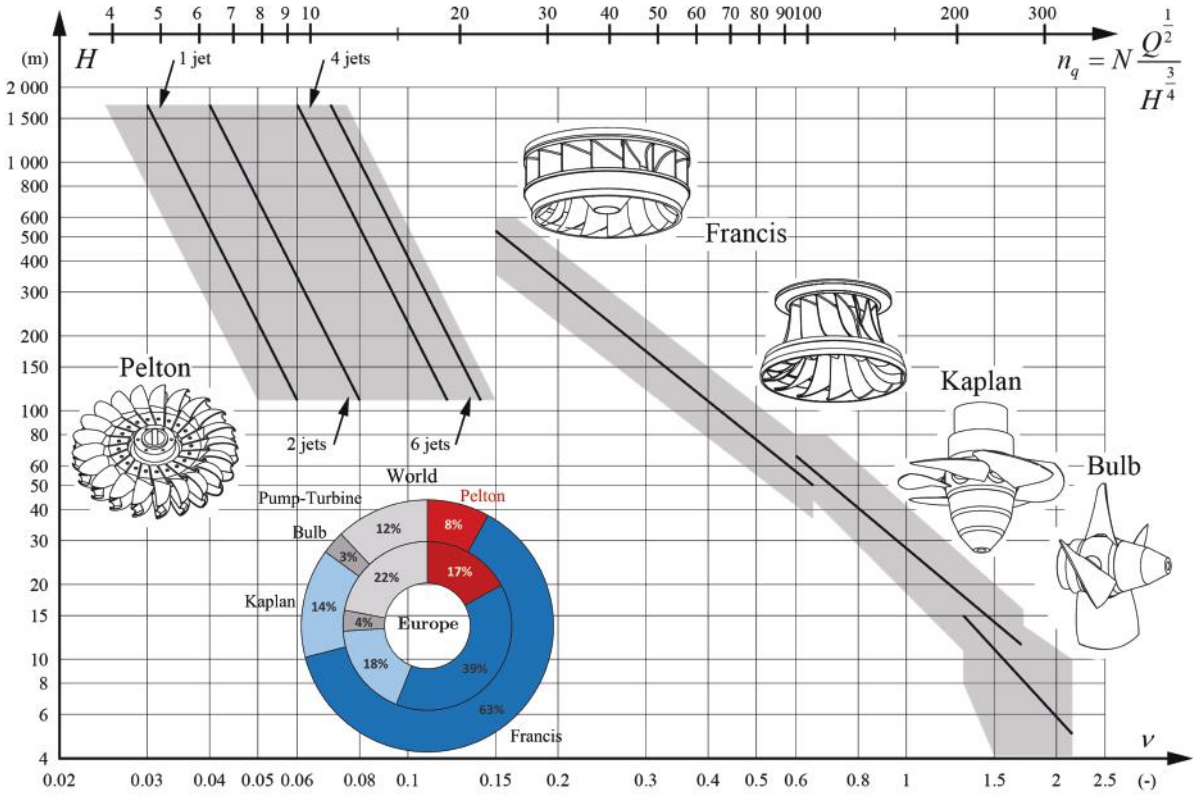


Figure 1.1. The appropriate operating range for the different runner types, where H (m), Q ($m^3 \cdot s^{-1}$) and N (min^{-1}) are the rated head, discharge and runner rotational speed, respectively.

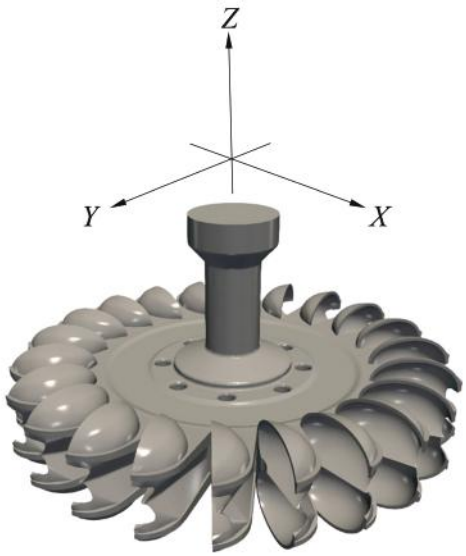


Figure 1.2. A vertical axis Pelton runner geometry with 22 buckets. This geometry will be used in chapter 4 for numerical simulations of multi-jet Pelton flow.

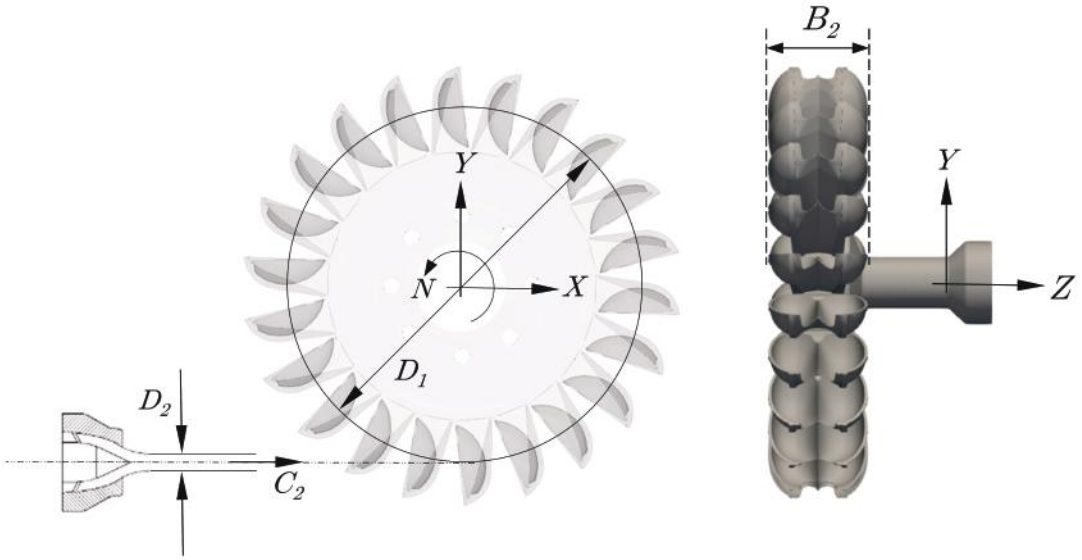


Figure 1.3. The main aspects of a single-jet Pelton runner. The runner rotates about Z–axis and the jet is injected along the X–axis direction.

1.2 Multi-jet Pelton Turbines

The Pelton turbine is one of the most efficient impulse turbines typically installed for high head hydropower plants. It generates power by transmitting the momentum of a water jet impinging on rotating buckets into the runner [1], [2]. The discharge is controlled by the nozzle spears, and the jet velocity C_2 is a function of the head H, i.e., $C_2 = c_v\sqrt{2gH}$, where c_v is the nozzle loss coefficient. The most appropriate types of runner, including Pelton runner, are reported in Figure 1.1 for a given head operating range as a function of the specific speed. As

of 2018, Pelton turbines represent 8% share of the world 1'100 GW hydroelectric capacity. The fluid flow inside the Pelton bucket is a transient free surface flow, and the pressure at the inlet and outlet of the bucket is atmospheric pressure [2]. A vertical-axis Pelton runner, as well as its principal dimensions, are shown in Figure 1.2 and 1.3, respectively.

The number of jets used in a Pelton turbine generally ranges from one to six. For a multi-jet Pelton turbine, the proper number of jets depends on the values of both the head and turbine specific speed. Given a hydroelectric site with a rated head H and discharge Q, the main design parameters are the turbine rotational speed N and the number of jets z_0. The jet velocity C_2 and therefore the optimum runner tangential speed $U = \pi n D_1$ is defined by the rated head H in which D_1 and n being the runner pitch diameter and rotational frequency, respectively. The value of the runner pitch diameter D_1 is therefore fixed by the first design parameter N. Similarly, the jet diameter D_2 and the bucket width B_2 are selected by the rated discharge Q, together with the second design parameter, z_0. The size of the runner is then a consequence of the design parameters N and z_0. This relationship is cast in the so-called runner unit specific speed defined as $n_q = N\sqrt{Q / z_0} H^{-0.75}$, which reveals the inevitable trade-off between the runner performance and the turbine manufacturing cost. Given that n_q is proportional to the bucket width per unit diameter B_2 / D_1, high specific speed generating units are more compact, implying less manufacturing and construction costs. However, this lower cost comes with a higher risk of jet interference, i.e., the interaction between two water jets on the same bucket, due to an increase of the discharge and the number of jets. Jet interference yields a torque drop and therefore, efficiency loss, which becomes significant for $z_0 \geq 5$. Moreover, Pelton runners with a large number of jets have more flexibility to control the power over a broader range, but the maximum number of jets is, however, limited by jet interference [3].

1.3 State of the art

1.3.1 Numerical simulation of Pelton turbine hydrodynamics

Numerical simulation of Pelton turbine flow is a challenging task featuring transient free surface flow and moving boundaries. Mesh-based methods have been broadly used to investigate Pelton turbine hydrodynamics. Zoppe et al. [4] employed the Volume of Fluid (VOF) free surface modeling approach to computing the pressure field on the Pelton bucket wall. Even though the numerical results for the bucket wall pressures were in a good agreement with the experimental results, VOF underestimated the leakage through the cutout compared to the experimental visualization. Kvicinsky et al. [5], [6] and Perrig et al. [7] used ANSYS CFX finite volume solver with the homogenous VOF for free surface and the k-ε turbulence model for the RANS solver to compute the resulting force experienced the stationary bucket. The numerical results showed a good agreement with the experimental results at the bucket middle zone while showing less accuracy at the splitter zone. However, the numerical simulation in the case of a stationary bucket is a simplified setup in which the boundary motions are ignored. Židonis & Aggidis [8] performed a single-jet numerical simulation of a

rotating Pelton runner with the VOF model to optimize the number of buckets and improve the hydraulic efficiency. Santolin et al. [9] and Jost et al. [10] simulated both nozzle and runner with homogeneous and inhomogeneous VOF model to estimate the impact of the real jet shape and velocity profile on the runner performance. Benzon et al. [11], [12] used both ANSYS Fluent and CFX to optimize the injector design. They reported a reduction of the injector-induced loss of 0.6% for the optimized geometry. Rossetti et al. [13] used CFX to investigate the cavitation mechanism in a Pelton turbine with Rayleigh–Plesset cavitation model for water-water vapor interphase transfer as well as a simplified computing domain featuring only three buckets to reduce the number of mesh elements. The water vapor produced during the water jet cut-in procedure has been reported.

However, mesh-based numerical simulation methods with Eulerian formulations face difficulty to cope with such a complex transient free-surface problem with moving boundaries, and a dynamic mesh approach, as well as run-time local grid refinements at the interphase, is required to handle these sophisticated flow features. Also, the VOF method is numerically diffusive and can lead to smearing of the free surface [14], [15]. The mesh-free (or particle-based) methods can overcome these difficulties by taking advantage of their Lagrangian approach. Koukouvinis et al. [14] used the standard Smoothed Particle Hydrodynamics (SPH) method to simulate fluid flow in a Pelton turbine. Thanks to its mesh-free nature, the approach was able to capture the flow features without diffusion at the air-water interface. However, the bucket geometry was represented only by its interior surface, meaning that the jet impact on the bucket backside was neglected. Their results were validated with ANSYS Fluent only. Marongiu et al. [16] employed a hybrid SPH-ALE method to compute the Pelton runner torque. The results highlighted a satisfactory agreement with the ANSYS CFX mesh-based solver, although the computed wall pressure was noisier than the mesh-based results. Anagnostopoulos and Papantonis [17] developed a Fast Lagrangian Solver (FLS) with a reduced computational cost for the design and optimization of Pelton turbines. However, FLS provides only an estimation of the integrated pressure based on the inlet and outlet particle velocity vectors; neither the whole pressure field nor the exact water sheets are considered.

Ye-Xiang et al. [22] performed a Pelton flow analysis based on the animated cartoon frame method to derive the fundamental equations of the dynamic performance of a single-jet Pelton turbine. The results revealed an efficiency deterioration at high specific speeds due to the interference between adjacent jets. Židonis & Aggidis [8] also found an efficiency drop in model tests for high flow rates due to jet interference. The operating range with a high risk of jet interference was determined with both numerical and model test analyses. Kubota [3] performed Pelton model tests with two adjacent jets to investigate the jet interference in a six-jet Pelton turbine case using a camera and experimental torque measurements. The experiments revealed a sudden efficiency deterioration at higher specific speeds wherein the jets tend to interfere significantly.

In 2015, Vessaz et al. [18] used SPHEROS, a 3-D Finite Volume Particle (FVPM) solver to compute the torque on a single-jet rotating Pelton runner. To reduce the computational cost, the numerical simulation was performed with only five adjacent buckets, and the intermediate buckets torque were used for the overall torque computations. Since the intermediate buckets provided the same torque time-history with the same trend due to geometry geometric periodicity, the time-shifted torque copy of one bucket intermediate bucket torque time-history accounting for the bucket periodicity can be superimposed and integrated, to find overall torque. The computed pressure and torque were validated against experimental data with reasonable accuracy. In the present research, the 3-D FVPM with spherical-support top-hat kernels is used to investigate jet interference in a multi-jet Pelton turbine.

1.3.2 Particle-based methods

The particle-based methods have been developed to provide stable and accurate numerical solutions for PDEs with various kind of boundary conditions (BCs) and arbitrary particle distribution without any node connectivity [34]. Contrary to the Eulerian mesh-based methods, particle-based methods with Lagrangian formulation are robust in handling free surface problems with moving boundaries. The particle-based methods exist in both weak and strong formulation. Unlike the strong formulation, the weak form of discretization provides both consistency and conservation regardless of variation in particles size [30] even though the integrations of the test function is required which applies higher computational costs. However, this integral is usually approximated due to its complexity. Smoothed Particle Hydrodynamic (SPH), Finite Particle Method (FPM), Diffuse Approximation Method (DAM), SPH-FPM and Finite Volume Particle Method (FVPM), all are particle-based methods developed in strong or weak form. The particle-based numerical simulation methods have been used in broad research fields from astrophysics to fluid and solid mechanics [23]–[30]. The reader is referred to Appendix A for the weak formulation of hyperbolic conservation laws.

FVPM was first introduced in 2000 by Hietel et al. [31] in 2-D. Later, in 2009, Nestor et al. [32] extended the method to viscous incompressible flow with using a consistency-corrected SPH approximation to evaluate velocity gradients. They applied the MUSCL scheme as well as second-order temporal discretization to improve accuracy. In 2011, Quinlan et al. [33] developed a fast approach for exact evaluation of particle area in 2-D using circular-support top-hat kernels. A simple schematic of FVPM with overlapping circular-supported particles is shown in Figure 1.4. The method was then extended to 3-D by Jahanbakhsh et al. [30] in 2015 featuring exact computation of particle volume and area for the cubic-support top-hat kernel. They later upgraded the cubic-support version to spherical-support, which provided more accurate and stable results thanks to its non-directionality and smooth interaction between the particles, although it incurred an extra computational cost due to its more sophisticated algorithm [11].

FVPM benefits from the desired features of both particle-based SPH and mesh-based FVM [30]. Unlike the standard SPH, FVPM is locally conservative and zero-order consistent regardless of variations in particles size. Indeed, the method can be interpreted as generalized conventional mesh-based FVM [19] in which the control volumes are replaced by overlapping compact supports. A pair of vectors $\mathbf{\Gamma}_{ij}$ and $\mathbf{\Gamma}_{ji}$ are then computed to weight the conservative flux exchange between each pair of neighbor particles $\mathbf{F}_{ij}$. These vectors are called interaction vectors, and their difference $\mathbf{\Delta}_{ij} = \mathbf{\Gamma}_{ij} - \mathbf{\Gamma}_{ji}$ is analogous to the area vectors in FVM [30]. FVPM also features an Arbitrary Lagrangian-Eulerian (ALE) approach where the computing nodes can arbitrarily move to achieve a reasonably uniform particle distribution. A brief comparison between SPH, FVM, and FVPM is provided in Table 1.1. FVPM has been extensively utilized to simulate the free surface flow in Pelton turbines from 2014 [36],[37],[38]. In the present research, a high-performance 3-D FVPM solver with spherical-support top-hat kernels is developed for free surface simulations. The solver is used to investigate the influence of the jet interferences in a multi-jet Pelton turbine, as a real application. Since the exact computation of FVPM particle interaction vectors is an expensive task that should be performed at every time step, the method has been accelerated on GPU. The accelerated solver is called GPU-SPHEROS.

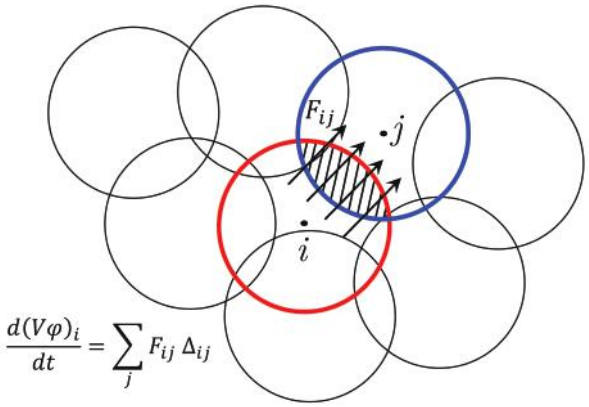


Figure 1.4. 2-D FVPM with circular-supported top-hat kernels; a simple schematic of overlapping particles exchanging flux.

Table 1.1. Comparison of FVM, SPH and FVPM features

	FVM	SPH	FVPM
Description method	Eulerian	Lagrangian	ALE
Formulation	weak	strong	weak
Conservation	yes	yes	yes
Consistency	yes	no	yes
Computational cost	low	medium	high
Free surface tracking	challenging	robust	robust
Handling moving boundaries	complex	simple	simple

1.3.3 Parallelization for GPU

1.3.3.1 GPU vs. CPU

GPUs and CPUs have significantly different architectures that make each one suited for specific tasks. CPUs are optimized for handling a single (complex) task rapidly while GPUs are fit for data-parallel simple tasks and hiding the latency with massive parallelism. A modern CPU with multi-core architecture can support up to hundreds of concurrent threads by having tens of cores while a modern GPU with many-core architecture features thousands of cores able to support tens of thousands of concurrent threads. A CPU is designed to run a massive and complex task as fast as possible by using pipelining, caching, and branch prediction. A GPU, on the other hand, is not fit for dense and complex processing on an individual or a few streams of data since the transistors are more dedicated to data processing rather than data caching and flow control. Although a CPU core is substantially faster than a GPU core itself, it cannot efficiently handle many streams of instructions simultaneously while GPU cores, with a slower clock rate and fewer features, are well suited for handling many independent simple tasks in parallel. This can explain why sometimes a well-optimized GPU-accelerated application running on thousands of GPU cores is only two or three times faster than its well-optimized CPU version running on a multi-core CPU setup having only tens of cores. Indeed, using the capabilities of a GPU requires a significant degree of parallelism. A serial code typically runs faster on a CPU than on a GPU. As a result, GPUs are never used alone but only as an extension of a CPU-based machine [40], [39].

1.3.3.2 GPU Parallel Computing Model

GPUs work based on the Single-Instruction Multiple-Threads (SIMT) execution model, introduced by NVIDIA® [40], in which a single instruction is run simultaneously by multiple threads enabling the developer to write thread-level parallel codes [40]. GPUs are built around many multi-threaded Streaming Multi-processors (SMs) designed to run hundreds of concurrent threads, which are queued up for work in groups of 32, called warps. All warps are further grouped into thread-blocks and can communicate within their block. Multiple thread-blocks run concurrently on an SM as far as the SM has sufficient resources and new thread-blocks are launched once the SM is vacated. All threads in a warp execute the same instruction in which each thread performs the operation on its private data. Since the threads in a warp execute the same instruction at a time, full efficiency is not obtained if all the threads of a warp do not follow the same path. This is called branch divergence and can turn a parallel execution into serial. Each SM has its own registers and shared memory, which are partitioned among warps and thread-blocks, respectively. A limited on-chip L1 cache is dedicated to each SM while L2 cache is shared between all the SMs [40], [40] and [42].

A good performance on the GPU is only achievable if sufficient parallelism is exposed, the algorithms are well suited to GPU hardware, memory has coalesced access pattern, execution within warps is coherent, data transfer between host and device is minimized and/or

overlapped with computations and GPU resources usage is balanced carefully [43]. The coalesced memory access concept is explained in the next chapter, section 2.3.1. The technical data and schematic of NVIDIA® Tesla™ P100 GPU architecture layout are given/shown in Table 1.2 and Figure 1.5 [43], respectively.

1.3.3.3 Memory Hierarchy

A schematic of the NVIDIA® Tesla™ P100 memory hierarchy is shown in Figure 1.6. A CUDA kernel, i.e., a function running on GPU in parallel, reads or writes the data into the device memory (DRAM) through the logical addressing spaces and different data caching levels. Global memory is the main memory of the GPU with a lifetime of allocating program. The data stored in global memory are visible to all threads over the whole runtime. Although global memory is large in size, it resides off-chip and features a long latency. The other GPU memory type is the local memory, which is per-thread and used for the operations not fitted into registers. Local memory is not a physical memory but an abstraction of global memory. Local memory resides off-chip and is as expensive to access as global memory. The global and local memory data are cached into L1 and L2 caches with roughly ~28 and ~300 clock-cycles of hit latency, respectively. Apart from global and local memory, a fast 64 kB per SM on-chip memory is available on each SM, which is called shared memory and features almost 100 times shorter latency than the global memory. Shared memory provides fast data access for all the threads within a thread-block. Unlike registers which are managed by the compiler, the shared memory should be explicitly declared and managed by the developer. The other memory type in GPU is the texture cache which is optimized for interpolation of multidimensional arrays connected to this read-only cache. Tesla P100 features 24 kB of L1 per SM and 4'096 kB of L2 for the full GPU. Altogether, registers and shared memory are the fastest GPU memory types on a GPU while global memory is the slowest one, although it is large (16GB on Tesla™ P100 SXM-2) [44][45].

1.3.3.4 GPU-accelerated particle methods

General Purpose computing on GPU (GPGPU) became practical and fashionable after about 2001 [46]. Since then, GPUs have been used in different areas of computational physics, e.g., molecular dynamics, Lattice Boltzmann method, Monte Carlo, finite element method and finite volume method to accelerate non-graphic computations [47]–[53]. The capability of GPUs to handle particle-based methods is demonstrated by [54]–[57]. GPU-SPH [54] and DualSPHysics [55] are both examples of broadly-used open-source particle-based solvers released during the last decade.

Table 1.2. Technical specifications of NVIDIA® Tesla™ P100 GPU (Pascal GP100) [44].

GPU	NVIDIA® Tesla™ P100 16GB
Compute capability	6.0
Architecture	GP100
Double Precision (DP) peak performance	5'300 GFlop s^{-1}
Peak memory bandwidth	732 GB s^{-1}
Double Precision (DP) cores	1'792
Base clock rate	1.33 GHz
Number of Streaming Multi-processors (SMs)	56
Max threads per SM	2048
Max warps per SM	64
Max thread-blocks per SM	32
Max 32-bit registers per SM	65'536
Max registers per thread	255
Shared memory per SM	64 kB
L1 cache per SM	24 kB
L2 cache	4096 kB
Launch date	Q3'2016

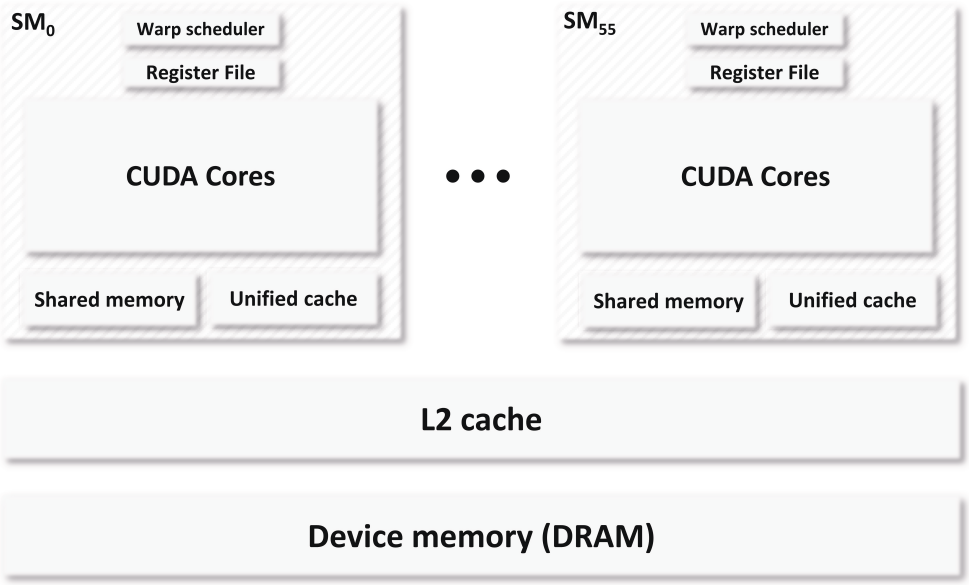


Figure 1.5. The schematic of the NVIDIA® Tesla™ P100 GPU hardware architecture [44]. The Tesla™ P100 GP100 features 1'792 double precision (DP) cores and 56 SMs (i.e., 32 DP cores per SM). Each SM can handle up to 2048 parallel threads.

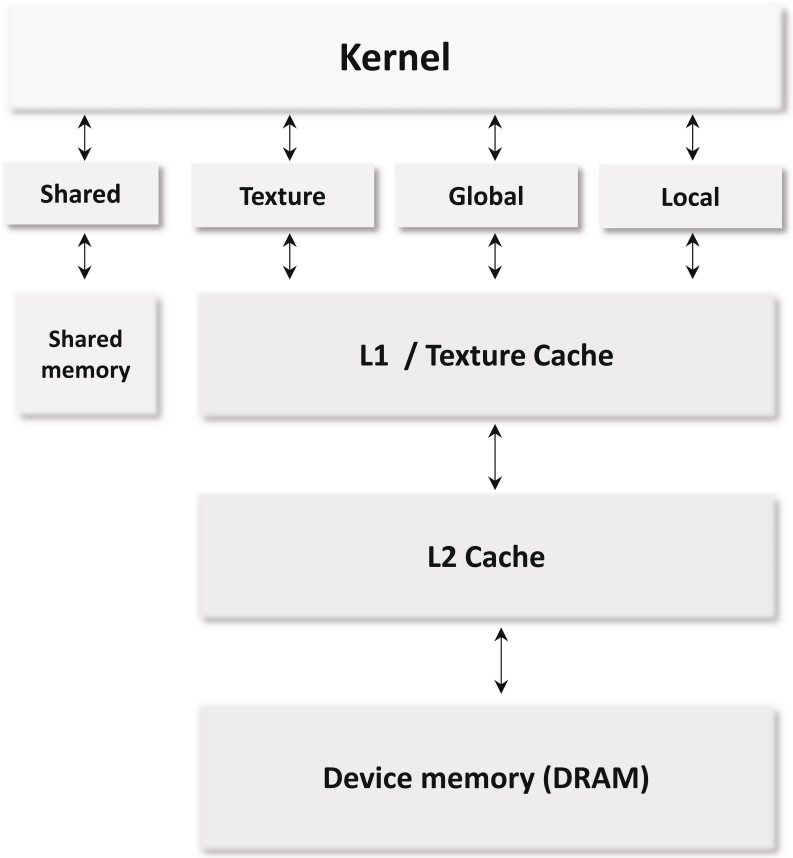


Figure 1.6. The Tesla™ P100 memory model [45]. Unlike the DRAM, which is the slowest memory type, the shared memory has the lowest latency with almost 100 times shorter than global memory.

1.4 Research objective

The present doctoral research is aimed at developing a high-performance conservative and consistent particle-based solver for industrial-size turbulent free-surface problems with or without large moving of boundaries. In the context of this research, a GPU-accelerated 3-D FVPM fluid solver has been developed to handle industrial-size free surface numerical simulations such as multi-jet Pelton runner flow. In this regard, new embarrassingly parallel algorithms have been designed to best use the potential of GPU hardware. The currently accelerated solver, GPU-SPHEROS, has been developed in the CUDA parallel computing platform. All the parallel algorithms and data structures have been designed specifically for GPU many-core architecture, and a roofline performance model has been utilized to visualize and determine the performance limiters to define the appropriate optimization strategies. In particular, the neighbor search algorithm, accounting for almost a third of the overall running time, features an efficient Space-Filling Curve (SFC) as well as an optimized octree construction procedure. The memory-bound interaction vector computation, accounting for almost two-thirds of the overall computing time, features fixed-size memory pre-allocation and

an efficient data ordering to reduce memory transactions and cost of dynamic memory operations, i.e., allocation and deallocation.

To capture the mean flow characteristics, standard and realizable k-ε as well as k-ω Shear Stress Transport (SST) have been implemented and integrated into ALE-based FVPM as two broadly-used two equation RANS models. The wall function approach has been utilized for the near-wall turbulence computations, and turbulence production limiters have been applied to prevent unrealistic turbulence build-up. To compute the nearest wall distance field required in SST, the diffusion-only equation with a source term of unity is solved within an iterative Poisson solver. The implemented turbulence models are then utilized for numerical simulations of Pelton turbine flow.

GPU-SPHEROS has been validated against the ANSYS CFX mesh-based commercial software as well as experimental data for *i*) lid-driven cavity, *ii*) fully developed turbulent flow in a circular pipe, *iii*) turbulent flow in a circular open channel, *iv*) impinging jet on a flat plate, and *v*) water jet deviation by rotating Pelton buckets. Once the solver validated, it has been used to investigate the interaction between the adjacent jets inside and outside the buckets, i.e., the jet interference and jet disturbance, respectively, in a six-jet Pelton runner as an industrial-size application. According to Kubota [3], both aforementioned phenomena are worth to be considered in the design process of a Pelton machine since they can significantly contribute to energy loss.

All the numerical simulations of multi-jet Pelton runner flow have been performed on Piz Daint, a GPU-powered supercomputer developed and operated by Swiss National Supercomputer Centre – CSCS.

1.5 Outline

The present thesis is a compilation of two independent published and submitted research articles in peer-reviewed journals.

This document contains five chapters, including the algorithms, implementation, optimization, and application. In *chapter 2*, the discretized form of the governing equations are presented, and the simulation flowchart, the parallel algorithms and the implementation methods have been described. Once all the algorithms implemented, a roofline performance model is then utilized in *chapter 3* for performance analysis. The performance limiters are determined, and appropriate optimization techniques are applied based on the limiters. The performance of the optimized code is compared to the original one within the roofline model. The optimized and validated solver is then used in *chapter 4* for a six-jet Pelton flow simulation as an industrial-size problem. The simulations are performed at eight operating points ranging from $N / N_{BEP} = 89\%$ to $N / N_{BEP} = 131\%$, and the jet-bucket, as well as jet-jet interaction, is investigated based on the torque time-history and free surface visualization. For reliability assessment, the results are compared to the experimental torque and efficiency measured in

Hitachi-Mitsubishi Hydro Corporation – HMHydro test rig. Finally, a conclusion and perspectives are given at the end, in *chapter 5*.

2 GPU-SPHEROS

In the present chapter, GPU-SPHEROS, as a GPU-accelerated 3-D FVPM solver, is introduced by presenting the discretized form of the governing equations, solver structure, as well as parallel algorithms. The solver has been developed in the CUDA parallel computing platform. The performance analysis and solver validation will be presented in the next chapter. The reader is referred to nomenclature for the definition of the letters and symbols used in the equations, figures, and tables in this document.

Part of this chapter is a reproduction and modification of part of a published peer-reviewed research article [95].

2.1 Governing equations

2.1.1 Conservative form

In fluid mechanics, any fluid flow field is characterized by the balance in mass, momentum, and energy, and conservation laws can be solved to find conserved physical quantities. The mass and momentum conservation equations for weakly compressible fluid flow are given as,

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{C} \tag{2.1}$$

and,

$$\rho \frac{D\mathbf{C}}{Dt} = \nabla \cdot \left(\mathbf{s} - p\mathbf{I}\right) + \rho \mathbf{g} \tag{2.2}$$

The deviatoric stress $\mathbf{s}$ is computed based on the strain rate,

$$\mathbf{s} = 2\mu_{eff}\left(\mathbf{S} - \frac{1}{3}\mathrm{tr}(\mathbf{S})\mathbf{I}\right) \tag{2.3}$$

where tr is the trace of a tensor and $\mu_{eff} = \mu + \mu_t$ in which μ_t is the turbulence eddy viscosity. $\mathbf{S}$ is strain rate tensor which is computed based on the velocity gradients,

$$\mathbf{S} = \frac{1}{2}\left(\nabla \mathbf{C} + \nabla \mathbf{C}^{\mathrm{T}}\right) \tag{2.4}$$

The pressure for weakly compressible flow is computed based on Tait's equation of state [59],

$$p = \frac{\rho_0 a}{\gamma}\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] \tag{2.5}$$

where γ is a constant coefficient set to 7.0 for water and the speed of sound a is assumed as 10 times greater than maximum fluid velocity to reduce the computational costs [60]. The conservation equations can be written as the following PDE form,

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = 0 \tag{2.6}$$

In general, Eq. (2.6) is known as the conservation law or balance equation if there is a source term. $\mathbf{U}$ and $\mathbf{F}$ are the conserved physical quantity and numerical flux function, respectively, defined as,

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho \mathbf{C} \end{pmatrix} \tag{2.7}$$

and,

$$\mathbf{F} = \begin{pmatrix} \rho \mathbf{C} \\ \rho \mathbf{C} \otimes \mathbf{C} - \mathbf{s} + p\mathbf{I} \end{pmatrix} \tag{2.8}$$

2.1.2 The Finite Volume Particle Method

GPU-SPHEROS has been developed based on the Finite Volume Particle Method (FVPM) which is a generalized form of classical mesh-based Finite Volume Method (FVM) [19]. In FVM, the domain is discretized into elements as finite volumes, and the area vectors for surfaces of each element are computed and used to weight the conservative flux exchanges between the control volumes. In FVPM, the control volumes are defined based on particle supports which have overlap, and the flux is exchanged through the overlapping areas. For each pair of i^{th} and j^{th} neighbor particles, two interaction vectors, $\mathbf{\Gamma}_{ij}$ and $\mathbf{\Gamma}_{ji}$, are computed, in which their difference $\mathbf{\Delta}_{ij} = \mathbf{\Gamma}_{ij} - \mathbf{\Gamma}_{ji}$ is equivalent to area vectors in FVM. The discretized momentum, mass and volume conservation equations are derived as [20],

$$\frac{d}{dt}\left(\rho_i V_i \mathbf{C}_i\right) = \sum_j \left[\left(\rho \mathbf{C} \otimes \dot{\mathbf{x}} - \rho \mathbf{C} \otimes \mathbf{C}\right)_{ij} - p_{ij}\mathbf{I} + \mathbf{s}_{ij}\right] \cdot \mathbf{\Delta}_{ij} - p_b \mathbf{B}_i \tag{2.9}$$

$$\frac{d}{dt}\left(\rho_i V_i\right) = \sum_j \left[\left(\rho \dot{\mathbf{x}} - \rho \mathbf{C}\right)_{ij} - \mathbf{R}_{ij}\right] \cdot \mathbf{\Delta}_{ij} \tag{2.10}$$

$$\frac{d}{dt} V_i = \sum_j \dot{\mathbf{x}}_{ij} \cdot \mathbf{\Delta}_{ij} + \dot{\mathbf{x}}_i \cdot \mathbf{B}_i \tag{2.11}$$

with,

$$\dot{\mathbf{x}}_{ij} = \left(\dot{\mathbf{x}}_j \cdot \mathbf{\Gamma}_{ij} - \dot{\mathbf{x}}_i \cdot \mathbf{\Gamma}_{ji}\right) \frac{\mathbf{\Delta}_{ij}}{\mathbf{\Delta}_{ij} \cdot \mathbf{\Delta}_{ij}} \tag{2.12}$$

$$\mathbf{B}_i = -\sum_j \mathbf{\Delta}_{ij} \tag{2.13}$$

The indices "*i*" and "*ij*" refer to the center of i^{th} particle and interface of i^{th} and j^{th} neighbor particles, respectively. The index "b" then refers to the boundary term. The boundary interaction vector $\mathbf{B}_i$ is used to weight the flux exchange through the boundary. This vector is only non-zero for free surface particles and becomes zero elsewhere. Since the velocity and pressure are computed at the same computational node, the term $\mathbf{R}_{ij}$ is added to the right-hand side of (2.9) to smooth the mass flux and damp the checkerboard pressure oscillations [61],

$$\mathbf{R}_{ij} = \left(\frac{1}{2} \left(\nabla p_i + \nabla p_j \right) - \tilde{\nabla} p_{ij} \right) \Delta t \tag{2.14}$$

The pressure gradient at the center of the particle ∇p_i is computed by the volume integral formulation [20],

$$\nabla p_i = \frac{1}{V_i} \sum_j \left(\frac{p_i + p_j}{2} \right) \mathbf{\Delta}_{ij} \tag{2.15}$$

while the gradients at the interface of i^{th} and j^{th} particles, i.e., $\tilde{\nabla} p_{ij}$ are computed with weighted least squares approach given as,

$$\begin{bmatrix} \tilde{p} \\ \tilde{\nabla} p \end{bmatrix} = \begin{bmatrix} \sum_j \mathbf{G}_j & \sum_j (\mathbf{x} - \mathbf{x}_j)^T \mathbf{G}_j \\ -\sum_j (\mathbf{x} - \mathbf{x}_j)\ \mathbf{G}_j & \sum_j (\mathbf{x} - \mathbf{x}_j)(\mathbf{x} - \mathbf{x}_j)^T \mathbf{G}_j \end{bmatrix}^{-1} \begin{bmatrix} \sum_j \mathbf{G}_j\ p \\ \sum_j (\mathbf{x} - \mathbf{x}_j)\ \mathbf{G}_j\ p \end{bmatrix} \tag{2.16}$$

where $\mathbf{G}$ is the Gaussian function defined as [20],

$$\mathbf{G}_j = \exp\left[-4 \left(\frac{\left| \mathbf{x} - \mathbf{x}_j \right|}{h_j} \right) \right] \tag{2.17}$$

The particle interaction vector is computed based on the integration of Shepard test function ψ over the domain [30],

$$\mathbf{\Gamma}_{ji} = \int_\Omega \frac{\psi_i \nabla W_j}{\sigma} dV \tag{2.18}$$

The Shepard test function ψ_i is defined as,

$$\psi_i = \frac{W_i(\mathbf{x})}{\sigma} \tag{2.19}$$

with,

$$W_i(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \Omega_i \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

as spherical-support top-hat kernel function and σ

$$\sigma = \sum_j W_j(\mathbf{x}) \tag{2.21}$$

as kernel summation. With this kernel choice, the integral over the volume of the domain Ω is simplified to an integral over the support boundary $\partial\Omega$. According to Quinlan et al. [33], the interaction vector computation is simplified as,

$$\mathbf{\Gamma}_{ji} = \sum_{e \in \left(\Omega_j \cap \partial\Omega_i\right)} \mathbf{S}_e \left(\frac{1}{\sigma_e + 1} - \frac{1}{\sigma_e} \right) \tag{2.22}$$

For any i^{th} particle with spherical supporting border $\partial\Omega_i$, the surface is partitioned into sub-surfaces created by intersecting neighbor particles. These sub-surfaces are called elementary surfaces e and are covered by σ_e neighbor particles. The process of computing the elementary surfaces is called surface partitioning and can be challenging due to the complex shape of elementary surfaces. For a set of elementary surfaces, all the corresponding area vectors $\mathbf{S}_e$ should be computed. An illustrative example of partitioned spherical support intersected by six neighbor particles is shown in Figure 2.2. The reader is referred to [30] for more details on the exact computation of FVPM interaction vectors with spherical-supported particles.

2.1.3 Particle motion and boundary conditions

In FVPM, the particle motion is defined based on the Arbitrary Lagrangian-Eulerian (ALE) approach in which the particle velocity $\dot{\mathbf{x}}$ is adjusted to fluid velocity $\mathbf{C}$ modified by a correction term $\dot{\mathbf{x}}_i^c$ to attain a reasonably uniform particles distribution. The particle velocity $\dot{\mathbf{x}}$ is computed by [30],

$$\dot{\mathbf{x}}_i = \begin{cases} \mathbf{C}_i + \dot{\mathbf{x}}_i^c - \left(\dot{\mathbf{x}}_i^c \cdot \mathbf{n}_i\right)\mathbf{n}_i & \text{for free surface, i.e. } \left|\mathbf{B}_i\right| \neq 0 \\ \mathbf{C}_i + \dot{\mathbf{x}}_i^c & \text{otherwise} \end{cases} \tag{2.23}$$

where $\mathbf{n}$ is the outward pointing unit vector at the free surface,

$$\mathbf{n}_i = \frac{\mathbf{B}_i}{\left|\mathbf{B}_i\right|} \tag{2.24}$$

and $\dot{\mathbf{x}}_i^c$ is the velocity correction vector defined as,

$$\dot{\mathbf{x}}_i^c = \lambda C^{char} \sum_j \left(\left(\frac{h_i}{\ell \left|\mathbf{x}_j - \mathbf{x}_i\right|} \right)^3 - 1 \right) \frac{\mathbf{x}_j - \mathbf{x}_i}{\left|\mathbf{x}_j - \mathbf{x}_i\right|} \tag{2.25}$$

In (2.25), λ is an adjusting coefficient set to a value between zero to 0.25. C^{char} is the characteristic velocity of the domain, and the parameter ℓ is computed based on,

$$\ell = \frac{h}{\delta} \tag{2.26}$$

which is typically adapted to a value ranging between 0.75 and 0.85.

Three different types of Boundary Conditions (BCs) have been implemented in GPU-SPHEROS: *i*) no-slip wall, *ii*) inlet boundary and, *iii*) free surface boundary. For the

no-slip wall boundary, a layer of spherical fluid particles is artificially overlaid on the geometry surface to build an impermeable geometrical boundary shape. These particles move with the wall velocity C_{wall}. For the inlet BC, fluid particles are injected into the domain with a known velocity and turbulence intensity I_{turb}. The velocity of the inlet particles are adjusted to fluid velocity $\mathbf{C}$ computed based on the discharge Q and the turbulence intensity is set based on the physic. Typically, $I_{turb} = 1\%$, 5%, and 10% correspond to low, medium and high turbulence intensity, respectively. Once a set of inlet particles entered into the computational domain, the inlet is fed by new particles injected in the same way as the previous set. This procedure is continued until the injection is stopped. For the free surface particles, the boundary interaction vector $\mathbf{B}_i$ becomes non-zero, and the pressure term $p_b\mathbf{B}_i$ is added to the fluxes [20].

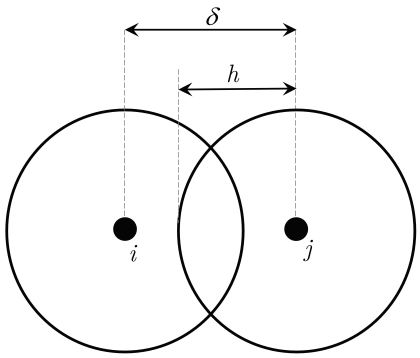


Figure 2.1. Neighbor particle distance and overlap. ℓ is h / δ which generally ranges between 0.75 and 0.85.

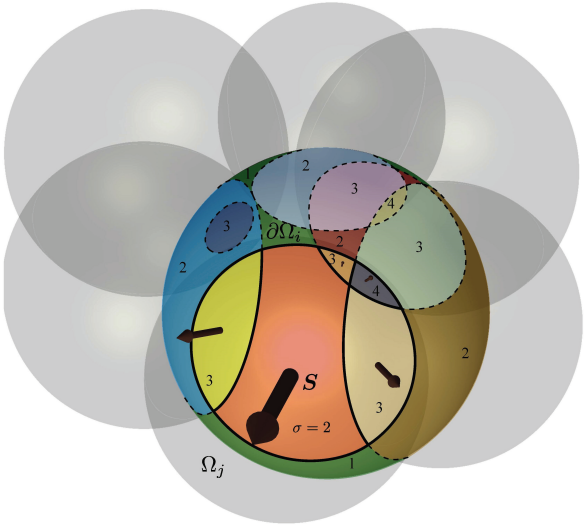


Figure 2.2. The intersection of the spherical surface of the i^{th} particle, $\partial\Omega_i$ with its neighboring particles. The elementary surfaces are shown in different colors with corresponding σ_e values [30].

2.1.4 Temporal scheme

In GPU-SPHEROS, the second-order explicit Runge-Kutta temporal scheme has been used for time integration. The field variable is updated in two steps, the predictor and the corrector. First, the intermediate fluxes are computed within the predictor step,

$$\mathbf{U}^{\left(t+\frac{\Delta t}{2}\right)} = \mathbf{U}^{(t)} - \nabla \cdot \mathbf{F}\left(\mathbf{U}^{(t)}\right)\frac{\Delta t}{2} \tag{2.27}$$

and then, the corrector step is performed to compute the final fluxes based on the computed intermediate flux

$$\mathbf{U}^{(t+\Delta t)} = \mathbf{U}^{(t)} - \nabla \cdot \mathbf{F}\left(\mathbf{U}^{\left(t+\frac{\Delta t}{2}\right)}\right)\Delta t \tag{2.28}$$

Since the solver is explicit, the Courant–Friedrichs–Lewy condition (CFL ≤ 1.0) must be satisfied for numerical stability and the time step size is determined based on:

$$\Delta t \le \mathrm{CFL} \times \min\left(\frac{\delta_i}{a_i + \left|\mathbf{C}_\mathrm{i}\right|}\right) \tag{2.29}$$

2.1.5 RANS–FVPM integration

2.1.5.1 Standard and realizable k-ε model

Within the framework of the present research, standard and realizable k-ε as well as k-ω SST, have been implemented and integrated into ALE-based FVPM as broadly-used two-equation models to capture the mean flow characteristics. The aforementioned RANS models have been extensively used and validated for both internal and free surface flows [62], [63]. For standard k-ε, the discretized transport equations for turbulence kinetic energy k and turbulence kinetic energy dissipation rate ε are derived as,

$$\frac{d}{dt}\left(m_i k_i\right) = \sum_j \left(\left((\mu + \frac{\mu_t}{\sigma_k})\nabla k\right)_{ij} - \left(\rho k\left(\mathbf{C} - \dot{\mathbf{x}}\right)\right)_{ij}\right) \cdot \mathbf{\Delta}_{\mathrm{ij}} + \left(P_k - D_k\right)_i V_i \tag{2.30}$$

$$\frac{d}{dt}\left(m_i \varepsilon_i\right) = \sum_j \left(\left((\mu + \frac{\mu_t}{\sigma_\varepsilon})\nabla \varepsilon\right)_{ij} - \left(\rho \varepsilon\left(\mathbf{C} - \dot{\mathbf{x}}\right)\right)_{ij}\right) \cdot \mathbf{\Delta}_{\mathrm{ij}} + \frac{\varepsilon_i}{k_i}\left(C_{1\varepsilon}P_k - C_{2\varepsilon}D_k\right)_i V_i \tag{2.31}$$

where P_k and D_k are the turbulence kinetic energy production and destruction, respectively, computed as,

$$P_k = 2\mu_t \mathbf{S}_{\alpha\beta}\mathbf{S}_{\alpha\beta} \qquad (\alpha, \beta = 1, 2, 3) \tag{2.32}$$

and,

$$D_k = \rho\varepsilon \tag{2.33}$$

The eddy (or turbulence) viscosity μ_t is computed as a function of ρ, k and ε,

$$\mu_t = C_\mu \frac{\rho k^2}{\varepsilon} \tag{2.34}$$

In the standard model, C_μ is a constant equal to 0.09. For wall boundaries, the zero-flux boundary condition has been applied for turbulence kinetic energy k, and a scalable wall function approach is used to adjust turbulence variables near the wall. The production of k near the wall is given by,

$$P_k^{wall} = \frac{\tau_{wall}^2}{\kappa \rho \Delta y_i C_\mu^{\frac{1}{4}} k_{wall}^{\frac{1}{2}}} \tag{2.35}$$

The turbulence dissipation equation is not solved for the wall-adjacent particles but instead is computed by Eq. (2.36),

$$\varepsilon_{wall} = \frac{C_\mu^{\frac{3}{4}} k_{wall}^{\frac{1}{2}}}{\kappa \Delta y_i} \tag{2.36}$$

where κ is von Kármán constant (which is $\kappa \approx 0.41$) and Δy_i is the normal distance of the center of i^{th} particle to the wall. It should be noted that the subscript "*wall*" refers to the adjacent particle to the wall, not the wall particle itself. The wall shear stress τ_{wall} is computed based on the scalable wall function approach,

$$\tau_{wall} = \frac{\rho C_{P_i} C_\mu^{\frac{1}{4}} k^{\frac{1}{2}}}{u^+} \tag{2.37}$$

where,

$$u^+ = \frac{1}{\kappa} \log(E y^+) \tag{2.38}$$

Eq. (2.38) is known as the logarithmic law, with $E = 9.793$, and a given formula for y^+,

$$y_i^+ = \max\left(\frac{\rho \Delta y_i C_\mu^{\frac{1}{4}} k^{\frac{1}{2}}}{\mu}, 11.06\right) \tag{2.39}$$

The y^+ against u^+ is shown in Figure 2.3 for pipe flow using the experimental data of Wei and Willmarth [90]. The logarithmic-law velocity profile is only accurate for $y^+ > 30$ within the logarithmic region without separated flow, but not for the buffer layer, where $5 < y^+ < 30$, or viscous sublayer where $y^+ < 5$. The scalable wall function approach is used to produce consistent results for arbitrary particle refinements avoiding errors originating from applying the log-law to the laminar and buffer regions of the boundary layer by shifting the near-wall particle to $y^+ = 11.06$. The value of $y^+ = 11.06$ in Eq. (2.39) is derived based on the intersection of linear and logarithmic u^+ profiles [64]. Eq. (2.40) gives an initial estimation of y^+ as a function of Reynolds number and near-wall particle size, derived based on the boundary layer

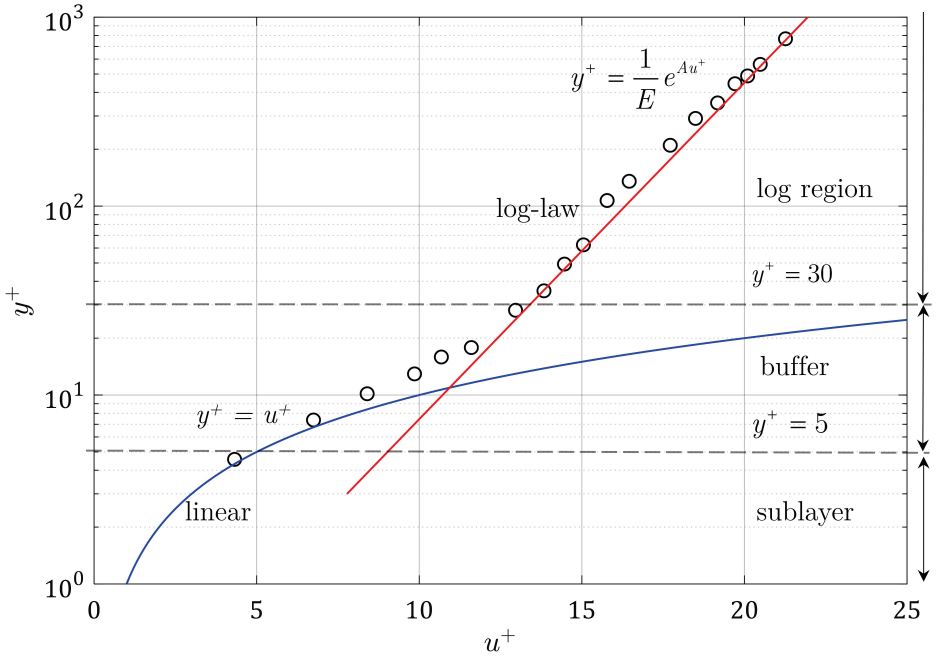


Figure 2.3. Mean velocity profile in a fully developed turbulent pipe flow [62]. The logarithmic-law is accurate within the logarithmic region for $y^+ > 30$, not the buffer layer, where $5 < y^+ < 30$, and the viscous sublayer, with $y^+ \leq 5$. The circles represent the experimental data for pipe flow provided by Wei and Willmarth 1989 [90].

theory for flow over a flat-plate [65] which can be used for initial particle spacing for a target y^+ value.

$$y^+ = f(\frac{\Delta y}{L}, Re_L) = \left[\sqrt{0.0135} \cdot \frac{\Delta y}{L}\right] Re_L^{\frac{13}{14}} \tag{2.40}$$

For the free surface boundaries, the gradients of k and ε normal to the free surface are set to zero, i.e., $\partial k/\partial \mathbf{n} = \partial \varepsilon/\partial \mathbf{n} = 0$, and for the inlet boundary, the values of k and ε are identified and adjusted based on the turbulence intensity and length scale.

The values for the model constants $C_{1\varepsilon} = 1.44$, $C_{2\varepsilon} = 1.92$, $\sigma_k = 1.00$, $\sigma_\varepsilon = 1.30$ have been obtained by comprehensive data fitting based on a broad range of turbulent flows [66].

2.1.5.2 Turbulence limiter

The unrealistic turbulence energy overproduction around the stagnation zone can create excessively large eddy viscosity values, which in turns significantly affect the flow predictions. To tackle the problem, one strain-rate tensor in (2.32) is replaced by the vorticity tensor Ψ. The turbulence production term then reads,

$$P_k^{limited} = 2\mu_t \mathbf{S}_{\alpha\beta} \Psi_{\alpha\beta} \tag{2.41}$$

in which the vorticity tensor is computed as,

$$\Psi = \frac{1}{2}\left(\nabla \mathbf{C} - \nabla \mathbf{C}^{\mathrm{T}}\right) \tag{2.42}$$

This limiter is known as Kato-Launder modification [91]. In shear flows and wakes, the modification gives the same results as the unmodified version. In the stagnation region, the vorticity tends to its minimum values, and therefore, the turbulence production is bounded. To prevent excessive unrealistic turbulence build-up, this strain limiter has been implemented and applied to the k-ε model.

2.1.5.3 Realizability

Realizability is the minimum requirement to prevent non-physical turbulence results. The realizable model, proposed by Shih et al. [67], have shown improvement over the standard k-ε model for the cases with vortices, rotational flow, and strong streamline curvature. The realizable k-ε model differs from the standard model in two important ways: firstly, the eddy viscosity equation has a non-constant C_μ [67][68] and secondly, the dissipation equation is modified based on the dynamic equation of the mean-square vorticity fluctuations. In the realizable model, the transport equation for turbulence kinetic energy k remains the same as in the standard model, but the dissipation equation is modified as,

$$\frac{d}{dt}(\rho\varepsilon) + \nabla \cdot (\rho\varepsilon \mathbf{C}) = \nabla \cdot \left((\mu + \frac{\mu_t}{\sigma_\varepsilon}) \nabla \varepsilon \right) + \rho\varepsilon \left(C_1^{rzb} S - C_2^{rzb} \frac{\varepsilon}{k + \sqrt{v\varepsilon}} \right) \tag{2.43}$$

with,

$$S = \sqrt{2 \mathbf{S}_{\alpha\beta} \mathbf{S}_{\alpha\beta}} \tag{2.44}$$

and,

$$C_1^{rzb} = \max \left[0.43, \frac{S}{S + 5\frac{\varepsilon}{k}} \right] \tag{2.45}$$

C_2^{rzb} is adjusted to 1.9 and C_μ is a function of strain rate tensor $\mathbf{S}_{\alpha\beta}$ as well as turbulence variables k and ε. The reader is referred to [67] for the full formulation of realizable C_μ.

Altogether, k-ε is a well-established model with a good performance for industry relevant applications validated for a wide range of flows but it provides poor results for flow with extra large strain rates and/or anisotropic normal stresses. For these cases, the Reynolds Stress Model (RSM) can address the problem, but it has a slower convergence rate as well as higher computational costs compared to two-equation models [66].

The normal stress positivity and Cauchy-Schwarz inequality are the main physical requirements to satisfy the realizability. The reader is referred to Appendix B for more details on realizability and Cauchy-Schwarz condition.

2.1.5.4 k-ω Shear Stress Transport

The Shear Stress Transport model, SST, combines the advantage of low-Reynolds k-ω and high Reynolds k-ε models by using blending functions for a smooth transition between the underlying models. The model is well known for providing an accurate prediction of flow separation under adverse pressure gradients [69]. The discretized transport equations for turbulence kinetic energy k and eddy frequency ω are derived as

$$\frac{d}{dt}\left(m_i k_i\right) = \sum_j \left[\left((\mu + \frac{\mu_t}{\sigma_{k3}}) \nabla k \right)_{ij} - \left(\rho k \left(\mathbf{C} - \dot{\mathbf{x}} \right) \right)_{ij} \right] \cdot \boldsymbol{\Delta}_{\mathrm{ij}} + \left(P_k - \beta' \rho k \omega \right)_i V_i \tag{2.46}$$

$$\frac{d}{dt}\left(m_i \omega_i\right) = \sum_j \left[\left((\mu + \frac{\mu_t}{\sigma_{\omega 3}}) \nabla \omega \right)_{ij} - \left(\rho \omega \left(\mathbf{C} - \dot{\mathbf{x}} \right) \right)_{ij} \right] \cdot \boldsymbol{\Delta}_{\mathrm{ij}} + \left(\left(1 - F_1\right) \frac{2\rho}{\sigma_{\omega 2} \omega} \nabla k \nabla \omega + \alpha_3 \frac{\omega}{k} P_k - \beta_3 \rho \omega^2 \right)_i V_i \tag{2.47}$$

The coefficients are a linear blending of inner ϕ_1 and outer ϕ_2 constants derived by

$$\phi_3 = F_1 \phi_1 + \left(1 - F_1\right) \phi_2 \tag{2.48}$$

The blending function F_1 tends to one inside the boundary layer and switches over to zero away from the wall. The turbulence eddy viscosity for k-ω SST is given by [69],

$$\mu_t = \frac{\rho a_1 k}{\max\left(a_1 \omega, SF_2\right)} \tag{2.49}$$

with $a_1 = 0.31$. The eddy viscosity limiter in (2.49) is to modify the shear stress transport behavior and avoid overprediction of eddy viscosity. Since the underlying assumptions are not correct for free shear flow, the limiter is restricted to near the wall by F_2, which is a blending function similar to F_1. For SST, to avoid unrealistic excessive turbulence build-up around the stagnation zone, the turbulence production term P_k in (2.46) and (2.47) is replaced by $P_k^{\max}$ which is limited by a maximum value [69],

$$P_k^{\max} = \min\left(P_k, 10 \beta' \rho k \omega\right) \tag{2.50}$$

with $\beta' = 0.09$. An automatic y^+--insensitive wall function approach is used for near-wall turbulence computations (see Appendix B). As the grid is refined, the method automatically switches from wall function to low-Reynolds formulation by blending the wall value for omega between the viscous sublayer and logarithmic region. The reader is referred to Appendix B for blending functions formulation and automatic wall treatment as well as model constants.

2.1.5.5 Nearest wall distance

To avoid the costly computation of exact nearest wall distance field, the diffusion-only equation with uniform source term $\nabla^2 \zeta = -1$ is numerically solved for a scalar field ζ using an iterative Poisson solver [70]. The Dirichlet ($\zeta = 0$) and Neumann (zero-flux) boundary

conditions are applied to the wall and other boundaries such as free surface, respectively. The discretized form of the $\nabla^2\zeta = -1$ is derived as [70],

$$\zeta_i^{n+1} = \zeta_i^n + \left(\sum_j \nabla\zeta_{ij} \cdot \mathbf{\Delta}_{\mathrm{ij}} + V_i \right) \frac{1}{\sqrt{\mathbf{\Delta}_{\mathrm{ij}} \cdot \mathbf{\Delta}_{\mathrm{ij}}}} \times \min\left(\left| \mathbf{x}_{\mathrm{i}} - \mathbf{x}_{\mathrm{j}} \right| \right) \tag{2.51}$$

and the nearest wall distance for i^{th} particle is computed based on the following expression [70],

$$d_i = -\left|\nabla\zeta_i\right| + \sqrt{\left|\nabla\zeta_i\right|^2 + 2\zeta_i} \tag{2.52}$$

Since ζ is non-negative, it is guaranteed that the computed wall distance will be always non-negative. As an example, the computed wall distance field for a rectangular domain with surrounding walls and free surface boundaries is shown in Figure 2.4. Also, the computed distance for the particles located at $y = 0.5$ compared to the exact distance is shown in Figure 2.5. Although the estimated distance is approximate, with almost 30% of maximum error at $x = y = 0.5$, the computations are not as expensive as the exact version.

2.2 The solver structure

2.2.1 The overall algorithm

A CUDA C++ program includes both CPU (host) and GPU (device) code. When a CUDA source code is compiled, the host and device parts are separated by the compiler. A typical host compiler (such as g++) is invoked for compiling the host code, and the device code is compiled by NVIDIA® CUDA Compiler, NVCC. Indeed, a simple CUDA program running procedure is split into the four following overall steps,

- first initializing the data on the host memory,
- then copy the initialized data to the device memory,
- run the kernel on GPU, and finally,
- copy the output data back to the host memory.

To avoid any data hazard, a correct synchronization between CPU and GPU is required. This however managed by the developer [71].

GPU-SPHEROS has been developed based on GPU well-suited parallel algorithms and data structures. The overall algorithm, summarized in Algorithm 2.1, includes three main parts: a) particle neighbor search, b) computing the particle interaction vectors, and c) computing fluxes and forces as well as integration in time. The CUDA kernels are shown with “do in parallel” keyword. All the parts run entirely on GPU to avoid costly host-device communication and the data are only copied back onto the host memory for saving purposes (e.g., every 100 or 1000 time steps). The Thrust and CUSP parallel algorithm libraries have also been used for programming productivity.

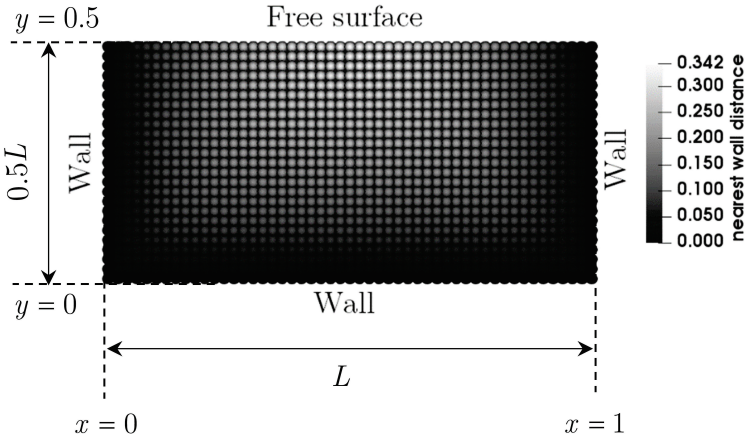


Figure 2.4. Computed nearest wall distance for a rectangular domain with three rigid walls and a free surface boundary. The particles are fixed, not moving.

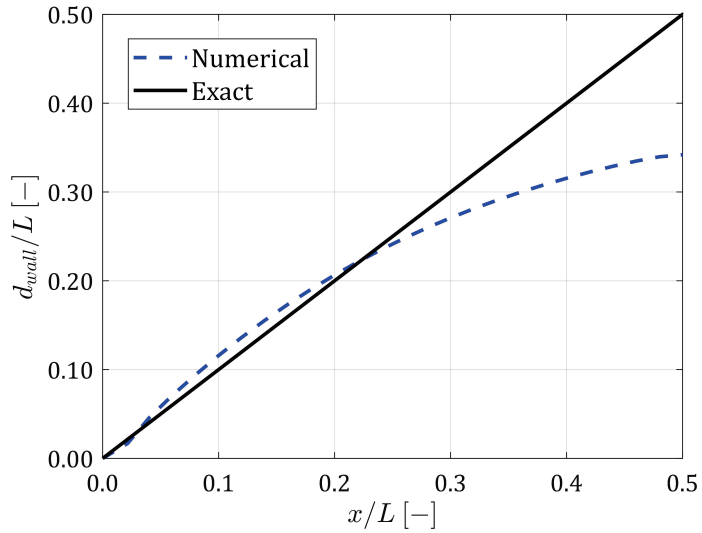


Figure 2.5. The computed nearest wall distance with the Poisson solver for free surface particles located at $y = 0.5$.

2.2.2 Simulation flowchart

To execute the code, first, the data are initialized on the host memory and then copied to the device. An octree-based neighbor search is then performed to find all the neighbor particles j for each i^{th} particle (part a). Once the neighbors are identified, the interaction vectors $\mathbf{\Gamma}_{ij}$ and $\mathbf{\Gamma}_{ji}$ (part b) are computed for each pair of i^{th}, and j^{th} neighbor particles and their difference $\mathbf{\Delta}_{ij}$ is used to compute the flux exchange between the neighbor particles (part c). The variables are then updated, and the same process is performed for the next time step. The simulation flowchart is shown in Figure 2.6. For both CPU and GPU versions, computing interaction vectors features the highest computational cost, with over 60% of the total running time. As it will be discussed later in chapter 3, ensuring that this part is efficient is key for speeding up GPU-SPHEROS.

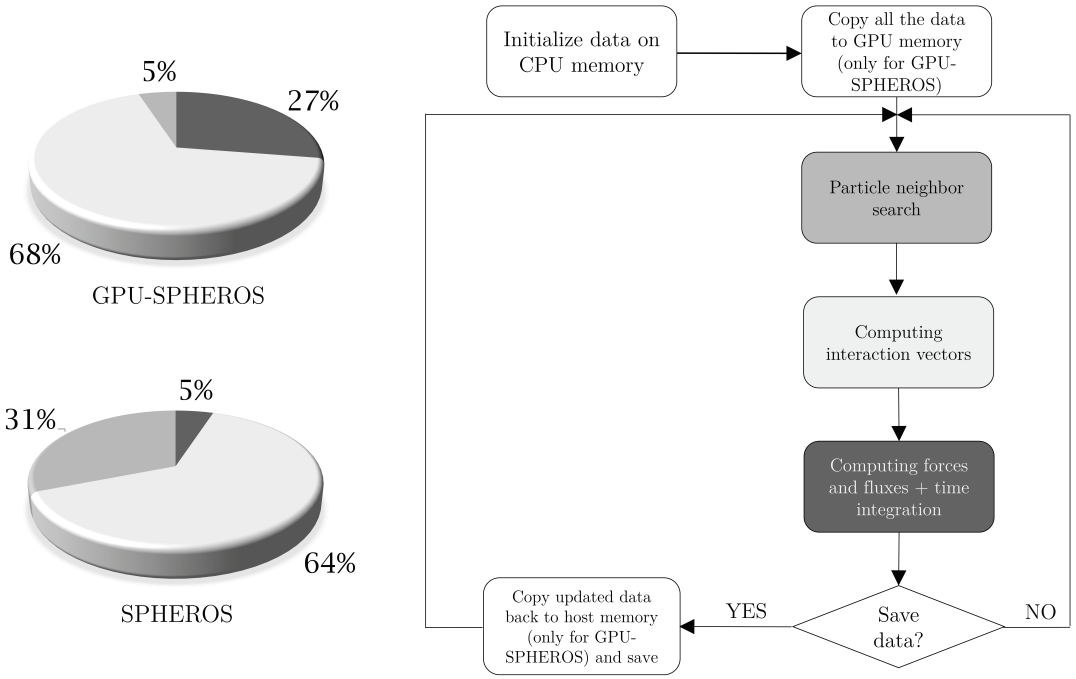


Figure 2.6. The flowchart of numerical simulation with SPHEROS or GPU-SPHEROS. For both codes, the pie chart represents the ratio of running time for each part of the code (after optimization), represented by their respective color, to the overall time.

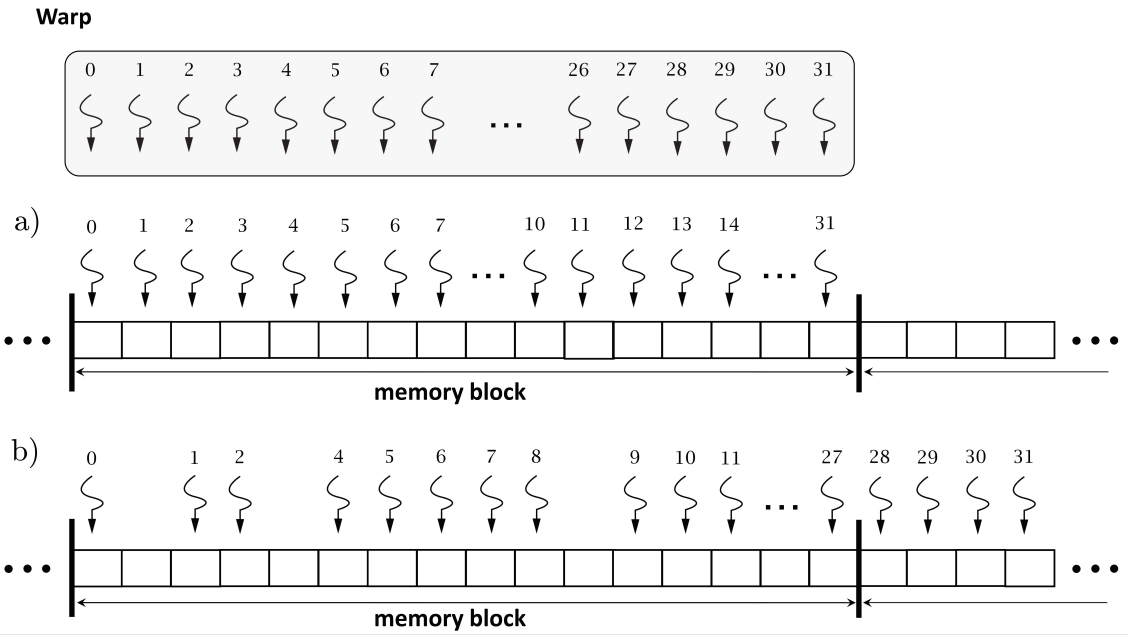


Figure 2.7. a) A warp has coalesced access to data elements with the minimum required memory transactions within a single memory block, and b) The data access pattern by a warp is not fully coalesced, and more memory transactions are required for requested loads or stores.

Algorithm 2.1. GPU-SPHEROS overall algorithm has three main parts, including a) particle neighbor search, b) computing interaction vectors and, c) computing forces and fluxes as well as integration in time.

```
01:  for each time step t do
02:      for all particles i do in parallel
03:          Find all the neighbor particles j (using an octree-based algorithm)
04:      end
05:      for all particles i do in parallel
06:          for each neighbor j do
07:              Compute interaction vectors with spherical-support kernel
08:          end
09:      end
10:      for all particles i do in parallel
11:          for each neighbor j do
12:              Compute momentum, mass and volume flux
13:              Compute turbulence kinetic energy and dissipation flux
14:          end
15:      end
16:      for all particles i do in parallel (using second-order Runge-Kutta scheme)
17:          Update volume, mass, momentum
18:          Compute density and velocity
19:          Compute pressure from the Tait equation of state
20:          Update particle position and velocity
21:          Update turbulence variables
22:      end
23:      t ← t + dt
24:  end
```

2.3 3-D FVPM implementation for GPU

2.3.1 General implementation consideration

The running time of a kernel is limited either by the GPU computational power or by the memory bandwidth. A kernel is called memory-bound (or bandwidth-bound) if its performance is limited by GPU bandwidth due to a large number of memory accesses per data element. For such kernels, the primary performance optimization strategy consists in storing the data in the fast but smaller GPU memory resources such as shared or constant memory to reduce memory latency and maximize hardware bandwidth usage [43]. Memory access can be further optimized by using a coalesced pattern. Shared memory can also be used to avoid uncoalesced memory accesses by loading and storing data from global memory in a coalesced pattern. The memory has a coalesced access pattern if load and store addresses from a warp are in the same memory block in which the warp accesses the data elements with minimum number memory transactions (e.g., four 32-byte transactions for a 128-byte memory block). A schematic of a coalesced and scattered memory access by a warp is shown in Figure 2.7.

On the other hand, a kernel performance is limited by GPU computational throughput if there is a large number of floating point operations per data element access. The kernel is then called compute-bound, and the number of concurrent threads per SM is mainly limited by register

count [43]. The primary optimization strategy comprises reducing the register pressure by using shared memory and data caching. The register pressure is intensfied by increasing the number of registers, which are required to hold all the per-thread private data. When register pressure is too high, and there are not enough physical registers to hold all the variables, registers are spilled into caches and then into the local memory, which is off-chip and has the same latency as global memory. Reducing register pressure can prevent register spilling.

GPU-SPHEROS is developed and optimized for/on NVIDIA® Tesla™ P100 (see Table 1.1 for P100 technical specifications). P100 takes advantage of High-Bandwidth Memory (HBM) technology as well as improved unified memory which enables the programmer to access both the CPU and GPU memory with a single pointer by automatic data migration between host and device physical memories. Unlike the previous architectures, Kepler and Maxwell, in Pascal, the GPU addressing capacity has been extended to 49-bit virtual addressing in Pascal architecture, which is sufficiently large to also cover modern CPUs 48-bit virtual addressing spaces as well as the full memory of the GPU itself. The program can then access the full address spaces of both CPU and GPU, which is not limited by device memory size anymore. Unified memory technology is used in GPU-SPHEROS to simplify host-device memory management [44].

2.3.2 Octree-based neighbor search

A Space-Filling Curve or SFC is a continuous function, which maps points from a multi-dimensional space into one-dimensional. SFCs have been used by researchers for particle nearest neighbor search [74], [75]. Bédorf et al. [72] developed a GPU-accelerated octree-based code for N-body simulations using the Morton curve method. In GPU-SPHEROS, the neighbor search implementation is based on the work by [72], using Morton keys to give a 1-D representation of the original 3-D coordinate space. After all the Morton keys have been computed using bitwise interleaving of particles coordinates $P = P(x,y,z)$ (see Appendix C), the particle data are sorted in increasing Morton key order, using the Thrust radix sort algorithm, and given an ID corresponding to their location along the Morton curve. This reordering improves data access efficiency by achieving a z-ordered particle distribution in memory.

Space is then partitioned into sub-spaces called branches using an octree, constructed recursively (see Algorithm 2.2). For this purpose, several levels of bitwise masking are applied to the particles Morton keys $\mathcal{PM}_i$. The first level mask is a 64-bit unsigned integer starting with the three most significant bits on (i.e., one), and the rest off (i.e., zero) and the mask is updated every level by setting the next three significant bits to one. For each level, the particles with identical masked Morton key $\mathcal{BM}_i$ are assigned to the same branch. The number of particles in each branch is computed using the parallel stream compaction algorithms provided by the Thrust library. If the number of particles in one branch is less than the adjusted limit, $\mathcal{N}_{leaf}$, that branch is called a leaf and is not further split. The binary masking and particles grouping process is repeated sequentially for every level until all the particles have been

assigned to leaves or the maximal depth of the octree has been reached, whichever occurs first. An example of a quad-tree generated with the Morton curve method for $\mathcal{N}_{leaf} = 4$ is shown in Figure 2.8. The branches with $\mathcal{N}_{leaf} \leq 4$ are flagged as a leaf in which filled by light gray and are not masked further. The other branches in which filled by dark gray and their particles are masked in the next levels.

By construction, all the neighbors of a particle are located in its own leaf and in its leaf's neighbor leaves. This knowledge saves a lot of additional floating-point operations and memory transactions, as the distance with particles belonging to other leaves does not need to be computed. For each leaf l, containing the set of particles $\mathcal{P}_l$, its neighbor leaves $N(l)$ (including itself) are identified and stored, as illustrated in Figure 2.9. The particles data are then passed to a kernel that computes the physical distance between the particles and identifies the neighbors of all the particles in $\mathcal{P}_l$. If j^{th} and ith particles are neighbors, the ID of j is saved into the neighbor list of the i^{th} particle, $\mathcal{NGB}_i$. Since the positions of the particles are updated each time-step, the neighbor search process must be performed every time-step. A new Morton curve is computed in every search process, and the particles IDs are renewed. The search algorithm is summarized as Algorithm 2.3.

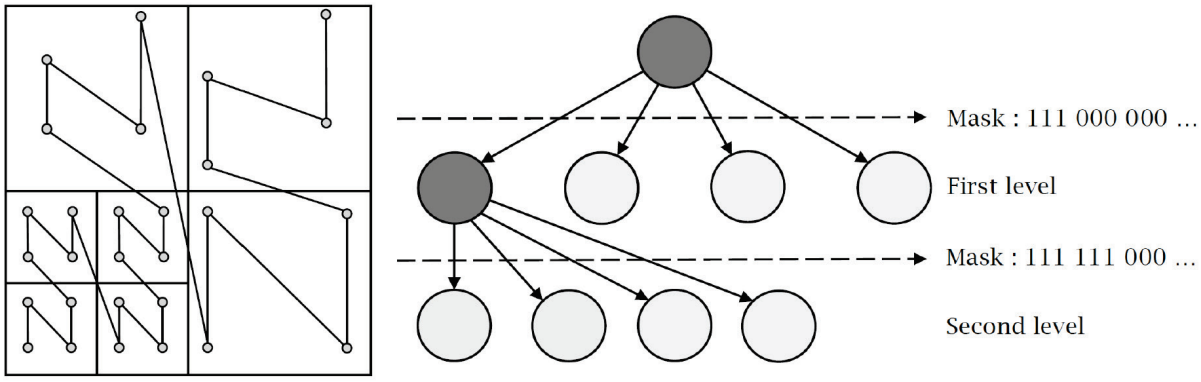


Figure 2.8. Quad-tree generated with the Morton curve method for 28 particles with $\mathcal{N}_{leaf} = 4$ (left) and schematic representation of the corresponding tree (right). For particle grouping into branches, their Morton keys are masked at each level by the corresponding level mask. The particles with the same masked keys will be grouped in the same tree branch.

Algorithm 2.2. Recursive octree construction based on the Morton curve method

```
01:  Construct octree (branch, level):
02:      for each particle i in branch do in parallel
03:          Apply the current level bitwise mask to particle Morton key PM_i
04:          Save masked keys as branch Morton key BM_i
05:      end
06:      Group particles with identical masked keys in identical branch
07:      Count the number of particles in each branch with stream compaction algorithms
08:      Tag the branches with more than N_leaf particles as nodes and the rest as leaves
09:      if all the branches are leaves or maximum tree depth is reached then
10:          break
11:      end
13:  for each node n, do in parallel
14:      Construct octree (n, next level)
15:  end
```

Algorithm 2.3. Octree-based particle neighbor search in GPU-SPHEROS

```
01:  for each particle i do in parallel
02:      Generate particle Morton key PM_i (see Appendix C)
03:  end
04:  Reorder data based on generated Morton keys PM_i (with Thrust parallel sort algorithm)
05:  Construct octree (all particles, level 1) [based on Algorithm 2.2]
06:  for each leaf l do
07:      Find neighbor leaves N(l)
08:      Identify particles P_N(l) ⊃ P_l located inside neighbor branches in parallel
09:      for each particle j in P_N(l) do in parallel
10:          for each particle i in P_l do
11:              Check the physical distance between i and j
12:              if i and j are neighbors then
13:                  Save j in NGB_i the neighbor list of particle i
14:              end
15:          end
16:      end
17:  end
```

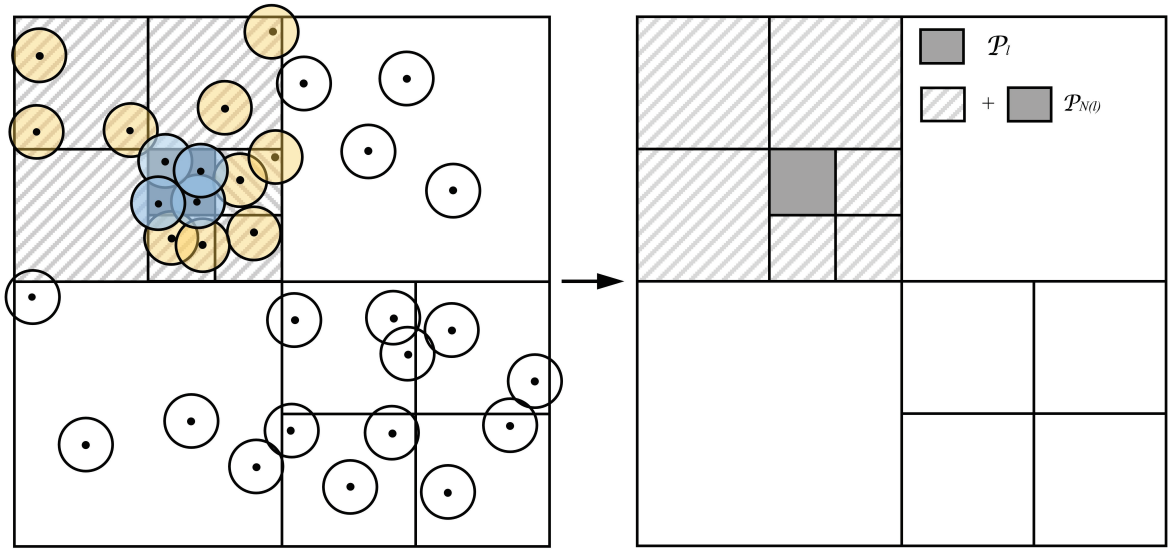


Figure 2.9. To find the neighbors of all the particles in $\mathcal{P}_l$ set (in blue), only the distances with particles in $\mathcal{P}_{N(l)}$ set need to be checked. $\mathcal{P}_{N(l)}$ set comprises the leaf itself with blue particles and its neighboring leaves with orange particles.

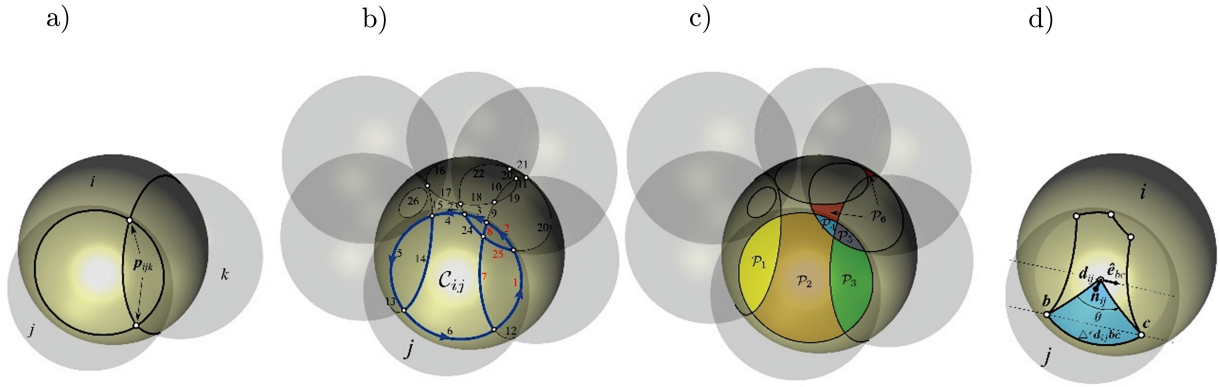


Figure 2.10. After identifying the spherical caps $\mathbf{C}_{ij}$ formed by intersecting i^{th} particle with each of its j^{th} neighbors individually, the steps for computing interaction vectors are: a) computing vertices $\mathbf{p}_{ijk}$ defined as the intersection of the particle of interest and two of its neighbors, b) constructing arcs $\mathcal{A}_{ij}$ joining the vertices, c) computing the elementary surfaces $\mathbf{S}_e$ delimited by the arcs and, d) computing the area of each elementary surface e [35].

2.3.3 Computing particle interaction vectors

In GPU-SPHEROS, the interaction vectors $\mathbf{\Gamma}_{ij}$ and $\mathbf{\Gamma}_{ji}$ are computed based on the method introduced by Jahanbakhsh et al. [35] in which each individual particle is defined by compact spherical support. Each support is intersected by its neighbor supports, and the partitioned surface has to be computed to find the elementary surfaces and area vectors required for the

Algorithm 2.4. Computing interaction vectors for 3-D FVPM with spherical-support particles on GPU [35]

```
01:  if the initial time step then
02:     for each particle i do
03:        allocate memory pool based on predefined upper bound values
04:     end
05:  end
06:  divide the particles into B_n batches
07:  for each batch B_k do
08:     for each particle i belonging to B_k do in parallel
09:        for each neighbor particle j do
10:           find the spherical cap C_ij
11:        end
12:     end
13:     for each particle i belonging to B_k do in parallel
14:        for each cap C_ij do
15:           for each cap C_ik do
16:              find the two intersecting vertices p_ijk of the surface circles
17:           end
18:        end
19:     end
20:     for each particle i belonging to batch B_k do in parallel
21:        for each cap C_ij do
22:           construct arc sets A_ij defined by all vertices p_ijk
23:        end
24:     end
25:     for each particle i belonging to B_k do in parallel
26:        for all each arc set A_ij do
27:           partition the spherical cap C_ij into the elementary surfaces e
28:        end
29:     end
30:     for each particle i belonging to B_k do in parallel
31:        for each elementary surface e do
32:           compute the area vector S_e and surface area S_e of e
33:        end
34:     end
35:     for each particle i belonging to B_k do in parallel
36:        for each neighbor particle j do
37:           compute interaction vector Γ_ij as in Eq. (2.22)
38:        end
39:     end
40:  end
```

interaction vectors computation. The overall procedure to compute the interaction vectors is illustrated in Figure 2.10. Readers are referred to [35] for the exact FVPM with spherical-support top-hat kernels, the formulation, and procedure. Several non-concurrent CUDA kernels have been implemented to perform these computations. By launching each kernel, one thread per particle is released, which is responsible for performing all the required

computations for its particle. Since the size of vertices, arcs, elementary surfaces, etc. are determined during the runtime, an estimated fixed-size upper limit memory is pre-allocated, and the particles are then grouped into smaller batches, and the batches are released sequentially to perform the parallel computations for each batch. The maximum batch size is limited by the hardware available physical memory. More details are given in the next chapter.

Algorithm 2.5. Computing fluxes, forces, and updating variables with second-order Runge-Kutta temporal scheme

① $\mathbf{f}_{\mathrm{i}} = \sum_{j}\left[\left(\rho\mathbf{C}\otimes\dot{\mathbf{x}} - \rho\mathbf{C}\otimes\mathbf{C}\right)_{\mathrm{ij}} - \mathbf{p}_{\mathrm{ij}} + \mathbf{s}_{\mathrm{ij}}\right]\cdot\boldsymbol{\Delta}_{\mathrm{ij}} - p_{\mathrm{b}}\cdot\mathbf{B}_{\mathrm{i}}$

② $\dot{m}_i = \sum_{j}\left[\left(\rho\dot{\mathbf{x}} - \rho\mathbf{C}\right)_{\mathrm{ij}} - \mathbf{R}_{\mathrm{ij}}\right]\cdot\boldsymbol{\Delta}_{\mathrm{ij}}$

③ $(\dot{mk})_i = \sum_{j}\left[\left((\mu + \frac{\mu_t}{\sigma_*})\nabla k\right) - \left(\rho k\left(\mathbf{C} - \dot{\mathbf{x}}\right)\right)_{ij}\right]\cdot\boldsymbol{\Delta}_{\mathrm{ij}} + \left(P_k - D_k\right)_i V_i$

④ $(\dot{m\varepsilon})_i = \sum_{j}\left[\left((\mu + \frac{\mu_t}{\sigma_*})\nabla \varepsilon\right) - \left(\rho\varepsilon\left(\mathbf{C} - \dot{\mathbf{x}}\right)\right)_{ij}\right]\cdot\boldsymbol{\Delta}_{\mathrm{ij}} + \frac{\varepsilon_i}{k_i}\left(C_{1\varepsilon}P_k - C_{2\varepsilon}D_k\right)_i V_i$

⑤ $\dot{V}_{\mathrm{i}} = \sum_{j}\dot{\mathbf{x}}_{\mathrm{ij}}\cdot\boldsymbol{\Delta}_{\mathrm{ij}} + \dot{\mathbf{x}}_{\mathrm{i}}\cdot\mathbf{B}_{\mathrm{i}}$

```
for all particles i do in parallel
    for each neighbor particle j do
        compute momentum and forces using ①
        compute mass flux including the smoothing mass flux term R_ij using ②
        compute turbulence fluxes using ③ and ④
        compute volume flux using ⑤
    end
end
for all particles i do in parallel (using second-order Runge-Kutta scheme)
    update mass m_i, volume V_i, and momentum m_i C_i
    update turbulence kinetic energy k_i and dissipation ε_i (or eddy frequency ω_i)
    compute particle velocity C_i and density ρ
    compute particle pressure p_i from Tait's equation of state
    compute particle velocity ẋ_i and update particle position x_i
end
```

2.3.4 Computing forces and fluxes

Once the neighbor list is identified and Δ_{ij} determined, the mass, momentum and volume fluxes are computed for all the particles by solving the discretized equations presented in section 2.1.2. The CUDA kernels release one GPU thread per particle to compute the fluxes and update the variables. For instance, the thread, which is released for the fluxes of particle i is responsible for computing all the flux exchanges between the i^{th} particle and its neighbors j. Unlike for the interaction vectors, there is no particle batching in this part of the code, since the memory requirements are limited and can be handled altogether. The algorithm for computing the fluxes and updating the variables is summarized in Algorithm 2.5.

2.4 Discussion

In this chapter, the discretized governing equations, as well as the implemented parallel algorithms, were presented. 3-D FVPM has been ported into GPU from scratch by designing the embarassignly parallel algorithms fitted to GPU many-core architecture. The method is conservative, zero-order consistent, and first-order accurate. Thanks to FVPM ALE-based formulation, the particle motion is handled with an arbitrary velocity in which the particles can be either fixed in space or move with an arbitrary velocity. This results in more flexibility in handling transient free surface problems with moving boundaries. A particle velocity correction term is used to achieve a reasonably uniform particle distribution which can have a significant impact on the errors. The particle interaction vectors $\mathbf{\Gamma}_{ij}$ and $\mathbf{\Gamma}_{ji}$ are computed based on spherical particles intersection and used to weight the conservative flux exchange. Unlike cubic-supported particles, the interaction between the spherical particles is free of directionality yielding a smooth interaction, although the overall algorithm is more sophisticated and expensive.

To model the turbulence effects, the transport equations for turbulence kinetic energy and dissipation rate have been discretized with ALE-based FVPM. The wall function approach is implemented and utilized for near-wall computations. Unlike k-ε turbulence model, for k-ω SST, the nearest wall distance is required to switch over between low-Re k-ω and high-Re k-ε via a blending function. This computation is performed at every time step via an iterative Poisson solver, which results in an extra computational cost. However, for high y^+ values, i.e., $y^+ > \sim 30$, SST will switch to k-ε formulation which is dominated. Standard or realizable k-ε can, therefore, be used in such a case for computational efficiency without the costly nearest wall distance computation. For problems with fixed particles, i.e., $\dot{\mathbf{x}} = 0$, the nearest wall distance and interaction vectors need to be computed only once, and the method will become significantly cheaper in terms of computational cost. However, a setup with moving particles is mainly required to handle the physics with transient free surface and/or moving boundaries.

Being embarrassingly parallel, the algorithms enable massive parallelization on GPU many-core architecture. Once the parallel algorithms are chosen/designed carefully, the data access in memory should be thoroughly managed to the best use of GPU high memory bandwidth.

To improve the data locality and memory access efficiency, the data is reordered in memory every time step based on the Morton curve approach. An octree-based particle neighbor search featuring GPU well-suited tree construction and traverse algorithms have been implemented for GPU. Once the octree constructed, the neighbor list is generated based on the saptial distance check between the particles in a branch and all its neighbor branches, including itself. The distance check is the most costly task of the overall neighbor search procedure and is performed by a user-developed kernel. The Thrust and CUSP parallel algorithm libraries have been utilized for programming productivity. The overall interaction vectors computation process is performed by synchronized consecutive kernels each one handling a subpart of the overall algorithm. The Kernels are synchronized, and all the data structures in memory are managed by CUDA unified memory. The flux is computed and weighted by interaction vectors, and the particle variables are updated with a second-order Runge-Kutta scheme. Separate individual kernels are launched to process each part of the computations such as computing gradients, computing fluxes, and forces, and updating variables and one thread per particle is released to perform its particle computations. The computations are entirely performed on GPU avoiding expensive host-device interaction except for the saving purposes.

Once all parts of the overall algorithm implemented for GPU, the code is profiled to determine the main performance bottlenecks. The neighbor search and computing interaction vectors, which altogether constitute 95% of the overall running time, remain the priority for optimization. The performance analysis is performed within a roofline performance model in the next chapter. The code is optimized based on the determined performance limiters, i.e., whether the hardware bandwidth or computational power. Once the solver optimized, the developer/user will be able to perform a series of standard test cases to validate the solver. The test cases should cover both laminar and turbulent internal and free surface flows for both fixed and moving particles.

3 Performance Optimization and Solver Validation

In the present chapter, the computational performance of the code is analyzed within a roofline performance model. The performance limiters are determined, and the code is optimized by applying proper optimization techniques. The validity of the solver is then evaluated for five test cases including both internal and free surface flow configuration, i.e., *i*) lid-driven cavity, *ii*) flow in a circular pipe, *iii*) open channel flow, *iv*) impinging jet on a flat plate, and *v*) jet deviation by rotating Pelton buckets. The validated solver is used in the next chapter for numerical simulation of a multi-jet Pelton turbine flow as an industrial-size application.

Part of this chapter is a reproduction and modification of a published peer-reviewed research article [95].

3.1 Performance assessment

3.1.1 Roofline-based performance analysis approach

In computer science, the roofline is an intuitive performance model which incorporates the computational throughput and memory bandwidth into a single log-log chart to provide an insight into the maximum achievable performance. The chart is described by Operational Intensity OI, expressed in Flop per byte, and computing throughput in Flops per second, i.e., Flops. For a particular multi-core or many-core architecture, the corresponding roofline is unique to that architecture and is derived based on the hardware maximum throughput and theoretical memory bandwidth. The roofline analysis approach can then be used to determine whether the application performance is limited by device memory bandwidth or peak performance. For a kernel (or application), OI is derived from dividing the number of floating-point operations performed by that kernel w by memory traffic q fetched by running the kernel [76], [77]:

$$\mathrm{OI} = \frac{w}{q} \tag{3.1}$$

For large OI values, the performance of the application is limited by the GPU peak throughput (so-called peak performance) shown by π, while for small OI, the bottleneck is data access,

and therefore, the performance is limited by $\beta \times$ OI in which β is the GPU maximum bandwidth.

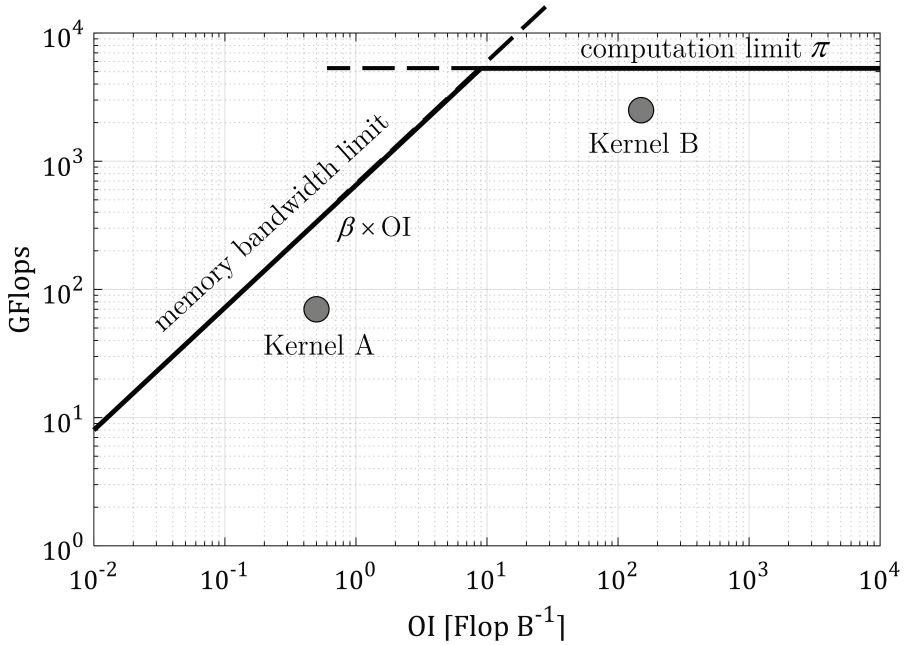


Figure 3.1. A naive roofline model example. Kernels A and B are limited by GPU memory bandwidth and theoretical performance, respectively.

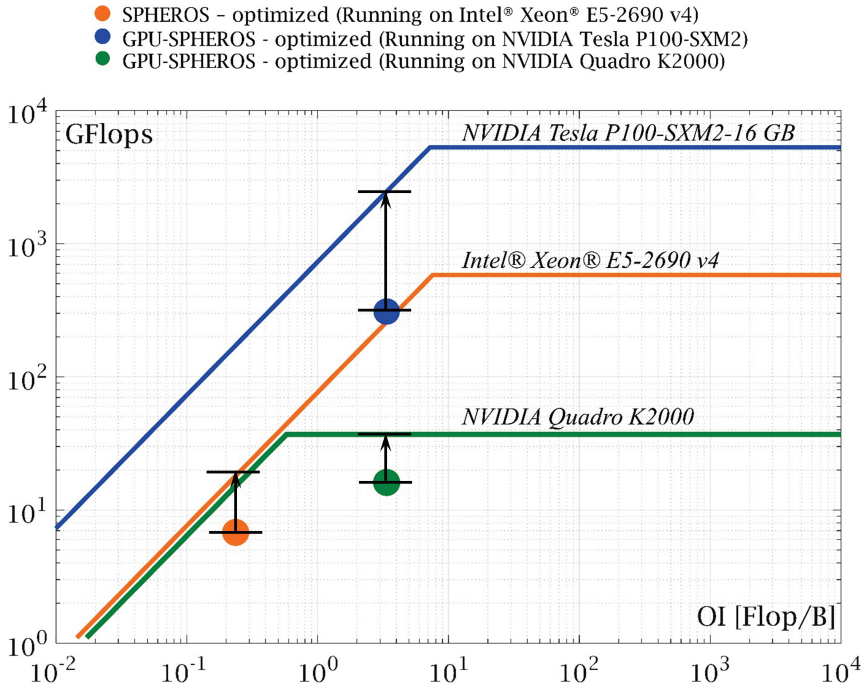


Figure 3.2. Performance of SPHEROS and GPU-SPHEROS on Intel® Xeon® E5-2690 v4 vs. NVIDIA® Tesla™ P100 SXM2 16GB vs. NVIDIA® Quadro K2000. Each marker color corresponds to its roofline.

Table 3.1. SPHEROS and GPU-SPHEROS overall performance. The given values correspond to the data shown in Figure 3.2.

Code		Processor	I [Flop B^{-1}]	Performance [GFlops]
GPU-SPHEROS	(non-optimized)	Tesla™ P100-SXM2 (5300 GFlops)	2.32	54.21
GPU-SPHEROS	(optimized)	Tesla™ P100-SXM2 (5300 GFlops)	3.36	309.67
GPU-SPHEROS	(optimized)	Quadro K2000 (37 GFlops)	3.36	16.10
SPHEROS	(optimized)	Xeon® E5-2690 v4 (582 GFlops)	0.23	6.79

A roofline plot can be used to determine the overall performance limiters for optimization decisions. An example of a naïve roofline model for Tesla™ P100 GPU is shown in Figure 3.1. The horizontal line is the maximum achievable throughput while the inclined line is derived based on the maximum bandwidth of the given GPU.

In the present research, the roofline analysis approach is used to evaluate the performance of the kernels, in which the throughput is plotted against the Operational Intensity. The result is then compared with the hardware-based theoretical bounds to determine the main performance limiter.

GPU-SPHEROS has been optimized on/for an NVIDIA® Tesla™ P100 SXM2 16GB GPU with GP100 hardware architecture. The overall performance of GPU code, GPU-SPHEROS, compared to the CPU version, SPHEROS, is shown in Figure 3.2. To demonstrate the code portability, the application was also run on NVIDIA® Quadro K2000 GPU with GK107 Kepler architecture even though Quadro K2000 is not designed for general-purpose computing. The code, however, ran successfully for the smaller problems that were not limited by device memory size. As indicated, on Tesla™ P100, the application is globally limited by the GPU memory bandwidth based on the Tesla™ P100 theoretical roofline ceilings. The CPU version has been run on Intel® Xeon® E5-2690 v4 Broadwell CPU without hyperthreading. All the corresponding values are also provided in Table 3.1.

3.1.2 Performance profiling

GPU-SPHEROS performance has been profiled with NVIDIA® Profiler tool, nvprof, and the CPU version has been profiled by Intel® Vtune™ profiler. The metrics given in Table 3.2 are used to measure the memory efficiency and Flop throughput of each kernel. The kernel is invoked many times by nvprof, and the performance measurements are averaged. The optimization routine is proceeded with finding and optimizing the performance bottleneck. Once the bottleneck optimized, the code is re-profiled, the new bottleneck is identified, and the new optimization is applied. This procedure is an iterative routine and is continued until satisfactory performance is achieved or there is not a remarkable potential for further optimization, anymore.

The overall performance of the program, together with the performance of each part of the algorithm, is shown in Figure 3.4a-d within Tesla P100 16GB naïve roofline model. It appears that the neighbor search kernel is compute-bound, whereas computing interaction vectors, as

well as forces and fluxes, are bounded by GPU bandwidth. Since the interaction vectors computations are performed by launching several non-concurrent kernels, the overall performance is measured based on time averaging the kernel performances. Based on the profiling results and identified bounds in the roofline model, specific optimization strategies are applied. As shown in Figure 3.4a, the overall performance limiter is memory bandwidth, and optimized code achieves almost 6% of the theoretical Double Precision (DP) peak performance of the Tesla P100-SXM2-16 GB. However, almost 35% of the Tesla™ P100 theoretical bandwidth is efficiently used for the parts bounded by the memory bandwidth. The achieved bandwidth is computed based on the device memory read and write transactions over the measured kernel running time. The CUDA timer is used for accurate timing.

3.2 Optimization

3.2.1 Octree-based neighbor search

The bottleneck of the neighbor search is the kernel, which computes the physical distance between the particles. This kernel takes up to 85% of overall neighbor search time and is considered as a compute-bound kernel with high OI based on Figure 3.4b. Optimizing this kernel helped to accelerate the neighbor search on GPU, dramatically.

The most effective optimization technique for this kernel consists of storing the data of particles set $\mathcal{P}_l$ in shared memory. These data are used as many times as there are set of particles in $\mathcal{P}_{N(l)}$. Therefore it is essential to ensure fast access to them. With the use of shared memory, the kernel has fast access to these data with almost one hundred times lower latency than global memory, without having the number of concurrent threads limited by the number of registers or unified cache resources. The data cached by shared memory can be accessed by all the threads inside the same thread-block when needed. By default, GP100 Pascal caches global loads in the unified cache acting as a coalescing buffer for memory accesses [44].

The "gather" algorithm of the Thrust library [79] has been used to provide a coalesced memory access by the kernel to the set of particles $\mathcal{P}_{N(l)}$. The $\mathcal{P}_{N(l)}$ particles data set are copied into a destination range according to a coalesced map. This task is efficiently handled with "thrust::gather" parallel algorithm. A temporary ID mapping the data to a contiguous memory location is attributed to each particle (see Figure 3.3). The temporary copy and ID are then passed into the kernel which will run faster than if the original data order was used. Even though "gathering" improves the performance of the kernel by improving the memory accesses performance, an extra cost is imposed by copying operation. However, the reported data in Figure 3.4b illustrate the cumulative performance gain in which this cost, which is almost 31% of the kernel performance gain after gathering, is included.

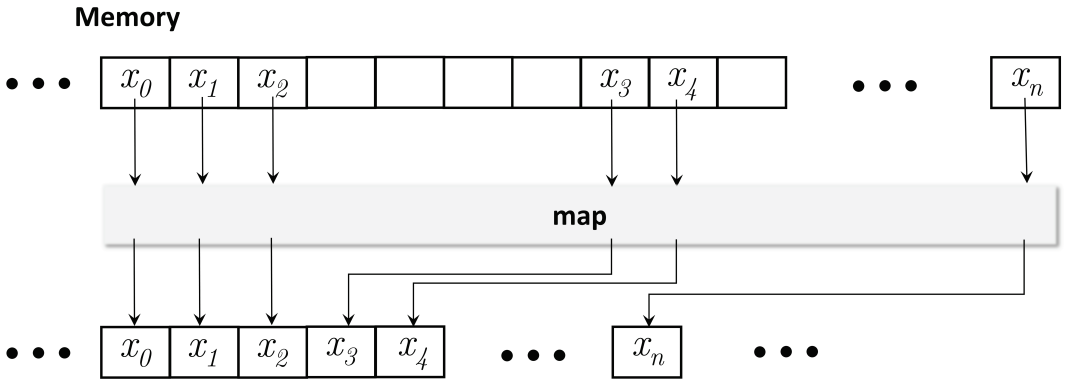


Figure 3.3. Schematic of data gathering to a contiguous temporary memory location before passing to the kernel.

Table 3.2. The used metrics for memory efficiency and Flop performance measurements on NVIDIA® Tesla™ P100 with compute capability 6.0 [78].

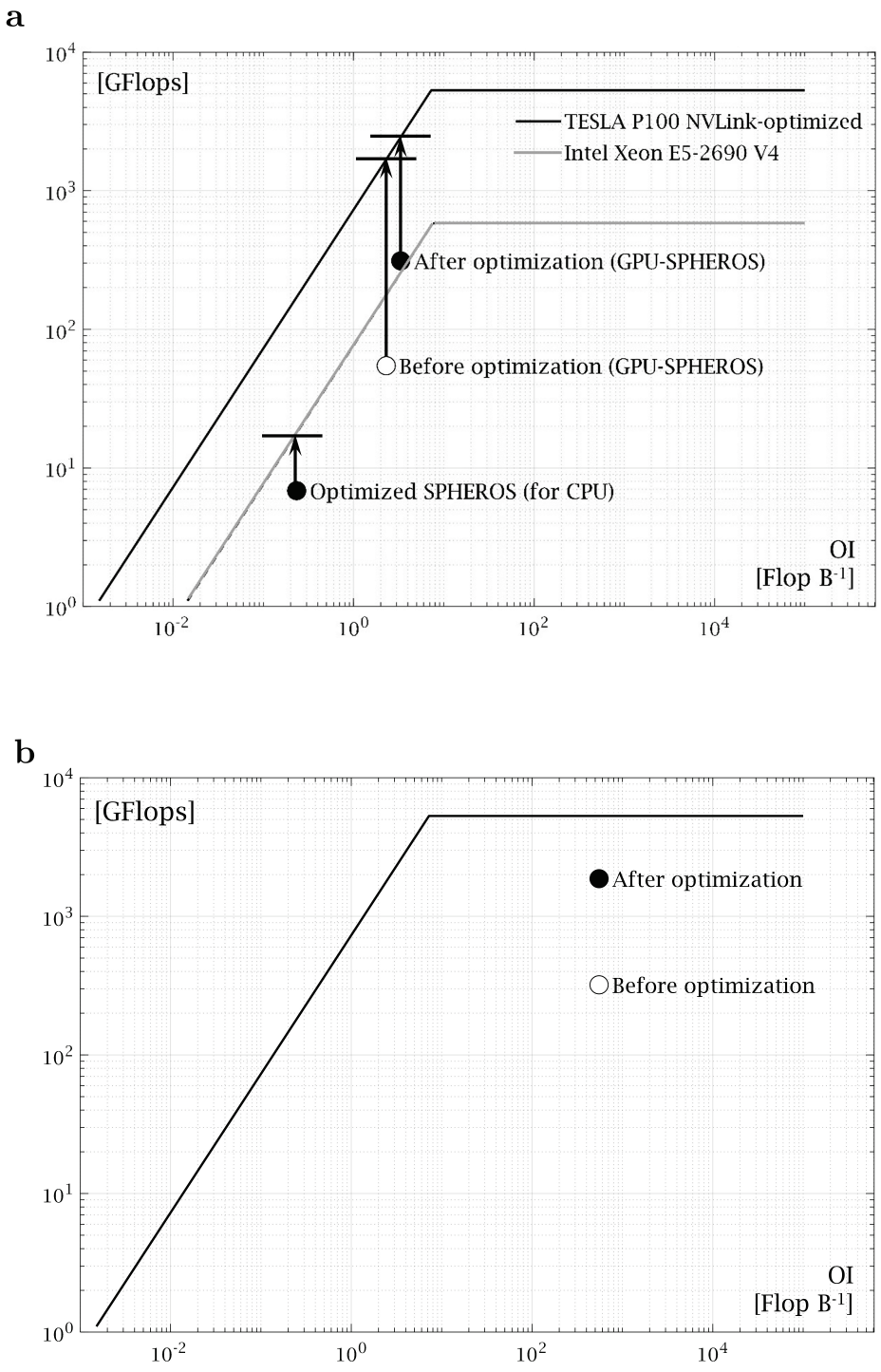
Metric	Description
`flop_dp_efficiency`	Ratio of achieved to peak double-precision floating-point operations
`achieved_occupancy`	The ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
`branch_efficiency`	The ratio of non-divergent branches to total branches expressed as a percentage
`dram_read_transactions`	Device memory read transactions
`dram_write_transactions`	Device memory write transactions
`gld_efficiency`	The ratio of requested global memory load throughput to required global memory load throughput expressed as a percentage.
`gst_efficiency`	The ratio of requested global memory store throughput to required global memory store throughput expressed as a percentage.
`sm_efficiency`	The percentage of time at least one warp is active on a specific multiprocessor
`warp_execution_efficiency`	The ratio of the average active threads per warp to the maximum number of threads per warp supported on a multiprocessor

As an illustrative example, the optimization procedure for the neighbor search kernel on NVIDIA® Tesla™ P100-SXM2 16GB has been presented in Table 3.3. Starting from v0, the original kernel without any optimization, the optimization techniques have been applied in three steps:

- v0 to v1: Reordering the data before passing into the kernel
- v1 to v2: using shared memory and,
- v2 to v3: choosing an optimized thread-block size based on the experiments

Altogether, from v0 to v3, the optimizations provided a performance gain of over 22% of NVIDIA® Tesla™ P100 16GB DP peak performance. The computational performance, as well as other metrics measured by NVIDIA® profiler "nvprof", are also shown in the table to help a better understanding of the present performance gain. The profiling data imply a significant

reduction in memory transactions as well as substitution of unified cache usage by shared memory, which has shorter latency. Like Maxwell, GP100 features an entirely dedicated 64 kB of on-chip memory per SM, always available for shared memory meaning that applications no longer need to select a preference of the L1 or shared split for optimal performance. This was not the case for former architectures, Fermi, and Kepler [44].



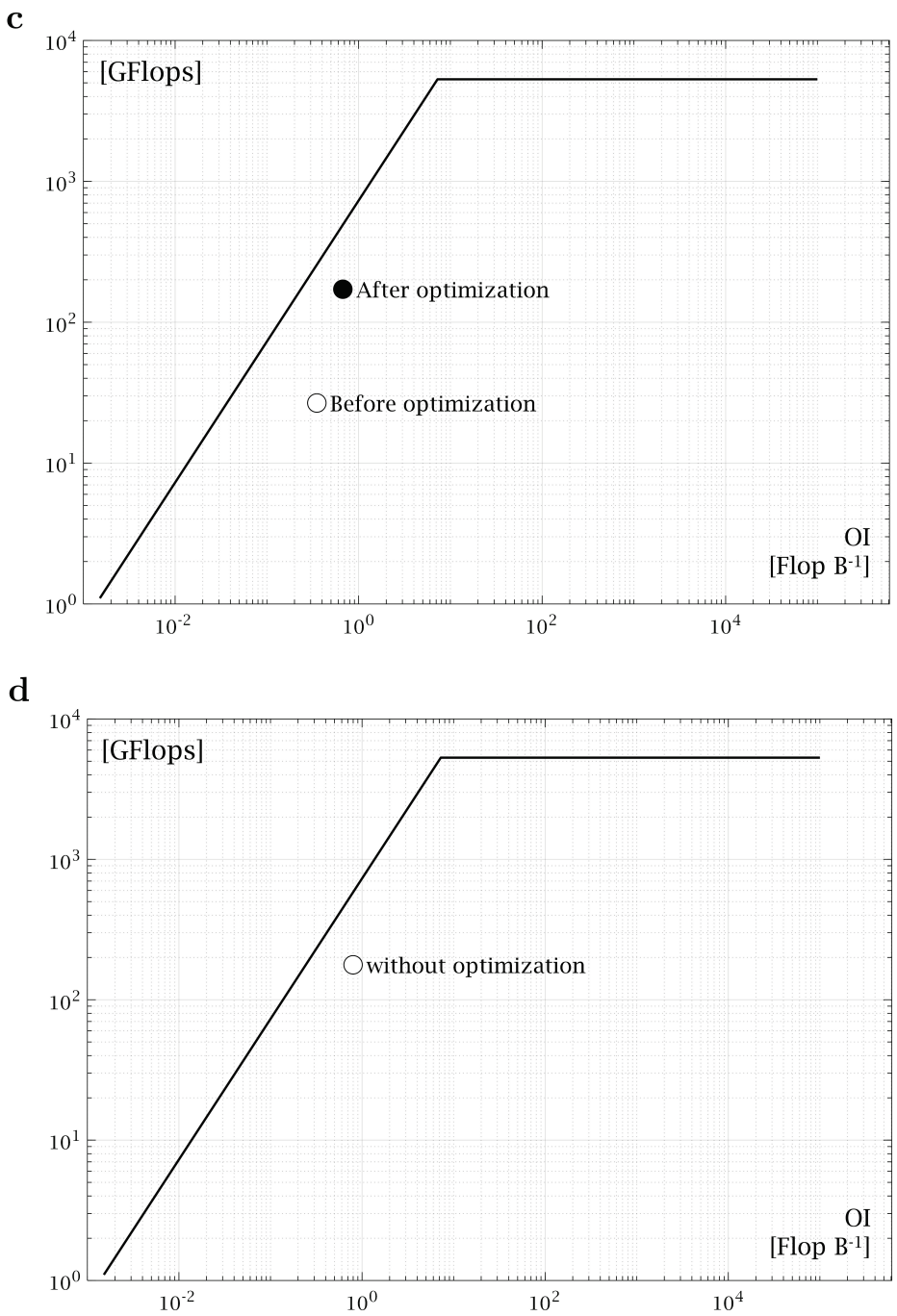


Figure 3.4. Naive roofline model of a) SPHEROS and GPU-SPHEROS (the whole application) and each part of the algorithm, b) neighbor search, c) interaction vectors, and d) fluxes and forces are shown in a theoretical roofline model for NVLink-based NVIDIA® Tesla™ P100. This part is only 5% of the overall running time and is not a performance bottleneck; therefore, it has not been optimized for the moment.

Table 3.3. An illustrative example of the optimization procedure for the neighbor search kernel on NVIDIA® Tesla™ P100-SXM2-16 GB GPU.

Metric	version			
	v0	v1	v2	v3
Flop DP Efficiency	8.5 %	9.7 %	27.0 %	30.8 %
Achieved Occupancy	35.6 %	35.2 %	34.2 %	32.0 %
Device memory read transactions	179 731	44 467	48 735	48 743
Multiprocessor Activity	83.0 %	83.4 %	83.9 %	90.5 %
Texture cache utilization	High	High	Very Low	Very Low
Branch Efficiency	100 %	100 %	100 %	100 %
Registers per thread	32	34	40	40

Altogether, a performance of more than 30% of Tesla™ P100 peak performance has been achieved for this compute-bound kernel. The performance of the particle neighbor search, before and after optimization, has been reported in Figure 3.4b in the roofline model. As shown in this figure, the operational intensity of the algorithm remains the same before and after optimization, which reveals that both kernels perform the same number of floating-point operations with the same amount of used data. However, the memory usage has been optimized, employing the aforementioned optimization techniques.

3.2.2 Computing interaction vectors

Several non-concurrent CUDA kernels have been implemented to perform different parts of interaction vectors computations (see section 2.3.3). By launching each kernel, one thread per particle is released, which is responsible for performing all the required computations for the particle.

For CUDA applications, to mitigate performance penalty due to expensive allocation and de-allocation operations, it is critical to reuse and/or sub-allocate device memory by the application wherever possible. Since the number of vertices, arcs, and elementary surfaces for interaction vectors computation depends on the local particle distribution, the size of the corresponding vectors is not fixed and is determined at run-time for each time step. To avoid inefficient dynamic memory operations, e.g., allocation, de-allocation, resizing, an upper-bound of the required memory is initially estimated and based on this estimate, a large block of fixed-size memory called memory pool, is pre-allocated for each particle. This upper-bound memory setting is based on the geometrical criteria for spherical intersecting particles and experience gained by the author in numerical simulations done by SPHEROS. Therefore, the bounds are chosen according to the developer/user experience to ensure that the simulation will not crash whilst minimizing the unused memory. For instance, in FVPM-based numerical simulations, each particle has typically between 20 and 40 neighbors and for most particles; this number is close to 27. Indeed, the number of neighbors for a particle goes higher than 40, very rarely. Therefore, the upper-bound value for neighbors of each particle can be set to $ngb_{max} = 50$,

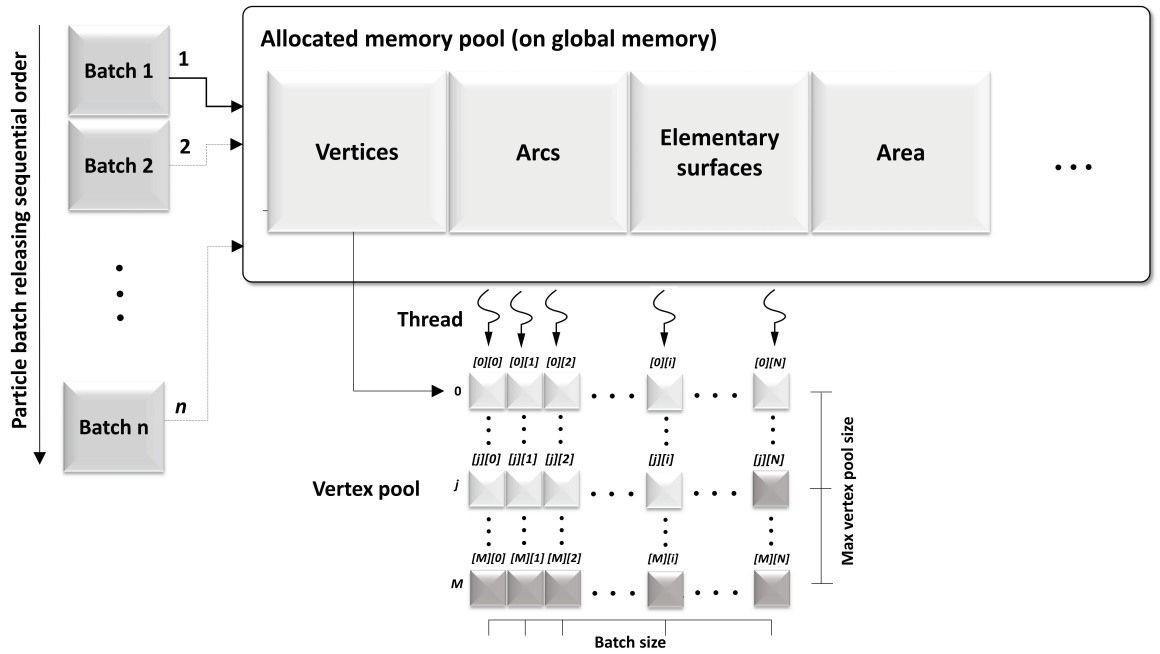


Figure 3.5. A sufficiently large memory block, here called memory pool, is pre-allocated for each vector before launching the kernels. Although a considerable part of global memory is occupied, with an unused part (shown in dark gray), the costly dynamic memory operations (allocation and de-allocation) are efficiently avoided.

which is indeed, a safe enough value. The upper-bound values for vertices, arcs, elementary surfaces, etc., are also adjusted similarly. The reader is referred to [35] for more details on intersecting spherical particles.

The kernels then perform the computations for a batch of particles in parallel, with subsequent batches released sequentially until all the interaction vectors have been computed. A schematic of the batching process and memory pool is shown in Figure 3.5. The batch size can significantly affect the performance: for small batches, the GPU is not filled, but the batch size is limited by the GPU global memory size. The performance of the application as a function of batch size is shown in Figure 3.6.

As shown within the roofline (see Figure 3.4c), computing the interaction vectors is bandwidth-bound. In the original non-optimized implementation of this algorithm, all the required computations are performed by a single kernel. Due to the full range of operations involved, the first optimization step consists in dividing this kernel into several smaller ones invoked sequentially. Much fewer memory and floating point operations are then performed per kernel, and the GPU resources are used more efficiently with more parallelism. For this bandwidth-bound part, providing an efficient memory access pattern can significantly improve the performance. As shown in Figure 3.4c, the vertical distance between the non-optimized kernel performance and the roofline limits indicates that the original implementation suffers from an immense memory transaction latency. and the first treatment to reduce the latency is to improve the access pattern of the data stored in the memory pool. The data inside the

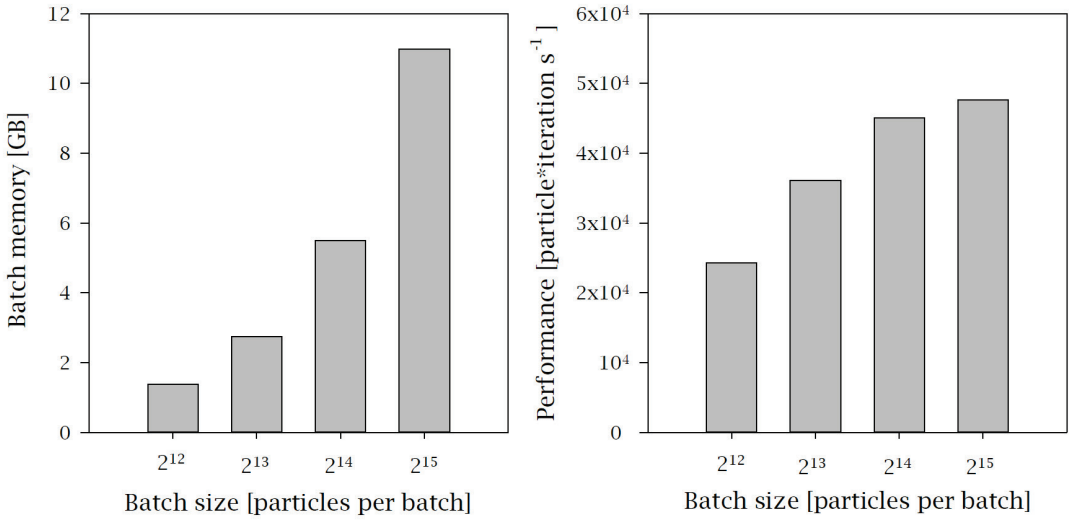


Figure 3.6. The memory occupied by a batch based on the corresponding batch size (left). Efficient batching when computing the interaction vectors has a dramatic effect on the overall performance of the solver (right).

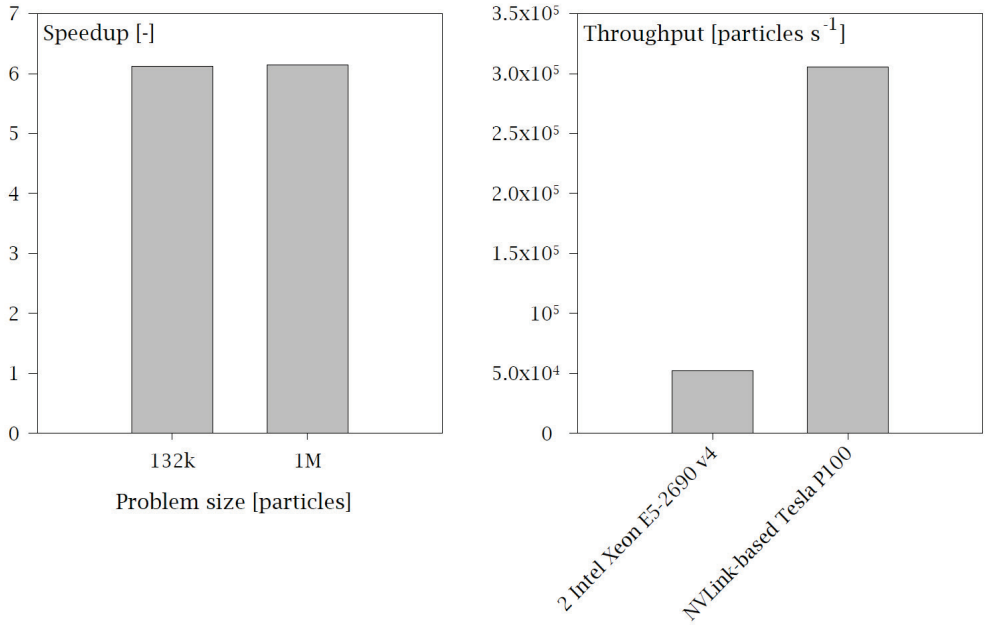


Figure 3.7. Achieved speedup (left) and solver throughput (right) on a single NVIDIA® Tesla™ P100-SXM2 16GB GPU vs. a dual-setup Broadwell CPU node with two Intel® Xeon® E5-2690 v4 and 28 total physical cores.

memory pool are stored in 1-D, 2-D, and 3-D arrays format. The original memory pool implementation features a natural selection of index order for multi-dimensional arrays. For instance, the ID of vertices located on each cap is stored in a 3-D integer array called «`capVerIds[pId][cId][vId]`» in which, «`pId`», «`cId`» and «`vId`» denote the particle, cap and vertex indices, respectively and since each thread is released for a single particle, the accessing pattern to «`capVerIds`» by the kernels will be strided. However, by inverting the indices order, i.e., «`capVerIds[vId][cId][pId]`», the consecutive threads will access the words in a coalesced pattern. The warp threads can then share their transactions, and the

performance is significantly improved (see Table 3.3c) due to a significant reduction in memory operations. This modification has been considered as a general strategy for all the arrays appearing in the interaction vectors computations such as vertices, arcs, elementary surfaces, area vectors, etc. In general, as any 2-D or 3-D array is translated into a 1-D array of data in memory layout, the order of access should be programmed carefully to avoid any performance drop due to a significant rise in memory transactions.

Further performance improvement has been achieved by simplifying the algorithm for computing the elementary surfaces e. In this algorithm, a large number of set intersections and differences are employed to construct the arc sets representing the elementary surfaces [35]. In the original implementation, the intermediate variables appearing during the process are stored in the memory pool to be re-used afterward. However, in the improved version, the loops inside the algorithm are merged in such a way that intermediate variables are not stored in the memory pool anymore which again results in a remarkable reduction in memory transactions. Since in the optimized version, the number of memory transactions is lower than in the original algorithm, the operational intensity is increased. This explains the horizontal shift in the roofline model in Figure 3.4c.

Minor optimizations, including inlining small device functions to eliminate the overhead associated with the function call and tuning the thread-block size also helped to improve the performance. Further optimization, such as the techniques for memory-bound kernels mentioned in section 2.3.1, can also be used to optimize individual kernels, but this has not been done at this point.

3.2.3 Flux computation and time integration performance

As shown in Figure 3.4d, the performance of the kernels computing the forces and fluxes and forces and integrating in time is close to the bandwidth limit, similar to optimized kernels for computing interaction vectors. Therefore, no optimization has been carried out for this part. Furthermore, this part represents only 5% of the overall running time and is therefore not a performance bottleneck. The algorithm is given in Algorithm 2.5.

Table 3.4. Intel® Xeon® E5-2690 v4 vs. NVIDIA® Tesla™ P100-SXM2-16 GB vs. NVIDIA® Quadro K2000; the specifications. The retail selling prices were surveyed on May 20, 2018.

Characteristics	Quadro K2000	Intel® Xeon® E5-2690 v4	Tesla™ P100 16GB
Architecture	GK107 Kepler	Broadwell	GP100 Pascal
FP64 peak performance	37 GFlops	582.4 GFlops @2.6 GHz	5 300 GFlops
Max. memory bandwidth	64.0 GB s^{-1}	76.8 GB s^{-1}	732.0 GB s^{-1}
Number of cores	386	14	3 584
Launch date	Q1'2013	Q1'2016	Q3'2016
Price ratio to E5-2690 v4 price	0.1	1.0	3.1

3.2.4 Overall speedup

To measure the speedup, the performance of SPHEROS on a dual CPU node with two Intel® Xeon® E5-2690 v4 Broadwell CPUs @2.6 GHz, featuring 14 cores and 28 threads per CPU is compared to the performance of GPU-SPHEROS running on NVIDIA® Tesla™ P100 SXM2 16GB GPU. The specifications on both used CPU and GPU are summarized in Table 3.4, and the speedup and software throughput is shown in Figure 3.7. As can be seen, the speedup is about six times, and almost independent of the problem size. For comparison, the Double Precision (DP) peak performance and maximum memory bandwidth of the GPU are almost 5 times higher than those of the CPU node. While it is true that the CPU software has not been as carefully optimized as the GPU one, this result suggests that GPU-SPHEROS takes great advantage of the hardware potential. The CPU Hyper-Threading (HT) was deactivated for all the experiments.

3.3 Solver validation

3.3.1 Selected test cases

After optimization, the reliability of the solver for internal and free surface flow simulations is validated by performing five different test cases: *i*) lid-driven cavity, *ii*) turbulent flow inside a pipe, *iii*) turbulent open channel flow, *iv*) an impinging jet on a flat plate and, *v*) water jet deviation by a rotating Pelton turbine which are presented in this section.

Lid-driven cavity : Once GPU-SPHEROS implemented, the validity of the solver is assessed for laminar internal flow by simulating a lid-driven cavity. The computed horizontal velocity profile along the cube vertical center-line is validated against the Ghia et al. [80] and Wong and Baker's [81] data for both 2-D and 3-D cases. The position of the particles are fixed in the space during the simulation, the computational domain is uniformly gridded, and the mass and momentum fluxes are weighted by interaction vectors and exchanged between the neighbor particles. The simulation is performed at $Re = 400$, in which the turbulence effects are negligible. Since the position of the particles is fixed, the neighbor list and the interaction vectors are computed only once at the beginning of the simulation and method will be significantly faster. Despite the simple boundary condition and geometry, lid-driven cavity flow simulation can be challenging due to vortical flow formation and discontinuity.

Circular pipe and open channel flow : Circular pipe and open channel are chosen as verification test cases for turbulence implementation and integration into GPU-SPHEROS. Since the walls are the main source of turbulence build-up, the developed turbulent boundary layer along the pipe or open channel with a fixed turbulence intensity I_{turb} at the inlet can be a good test case for verification of the model implementations. The pipe length is 10 times greater than the pipe diameter to allow the turbulent boundary layer to develop [65], and the pipe shape is circular in which the Boussinesq hypothesis for isotropy remains valid. Once the implementation of the models validated for the developed pipe internal flow, the free surface

effect can be further verified within an open channel flow numerical simulation. The tests are performed at $Re = 10'000$, which is computed based on the pipe diameter, D_{pipe}. For all the test cases, the velocity profile and turbulence kinetic energy at the end of the pipe are verified against the ANSYS CFX commercial solver results.

Impinging jet on a flat plate: Impinging jet on a flat plate is chosen as a free surface flow test case featuring hydrodynamics close to the Pelton turbine flow. Since the runner torque in a Pelton turbine is mostly generated by the pressure forces, evaluating and validating the predicted pressure accuracy is a crucial factor performed by this test case. The water jet is perpendicular to the flat plate, and the plate is considered as an impermeable no-slip wall boundary. The pressure coefficient C_p is computed for both uniform and non-uniform jet velocity profile along the center-line of the plate and the effect of the non-uniformity of the velocity profile on the pressure peak magnitude is also covered by this test. The computed pressure, as well as jet free surface elevation, are validated against the ANSYS CFX results and available experimental data. Turbulence limiters are also applied to prevent excessive turbulence build-up around the stagnant region.

Single jet Pelton turbine hydrodynamic: Once the pressure and free surface shape and elevation validated within the former test case, the solver is prepared for numerical simulation of Pelton turbine flow. The main purpose of this test case is to evaluate the solver applicability and accuracy for the rotating Pelton flow simulation. A single water jet is injected into the computational domain, and the rotating buckets are considered as no-slip wall boundaries. The generated torque time-history is derived based on the computed pressure and shear forces. To investigate the grid independence, the simulation is performed with three different spatial resolution. The time-averaged overall torque is validated against available experimental data. Similar to the impinging jet test case, the simulation is performed with both uniform and non-uniform inlet velocity and turbulence intensity provided by [82]. Once the computed torque validated against the experimental data, the solver can then be used for realistic and larger simulations such as multi-jet rotating Pelton runner flow in off-design conditions.

3.3.2 Lid-driven cavity

The lid-driven cavity is a standard test case for viscous fluid flow. The flow is surrounded by walls in a cube. The bottom and side walls are stationary while the top wall is moving along the horizontal axis with a reference velocity (see Figure 3.8). The viscous shear force applied by the moving wall leads to a complex vortical flow formation [20]. The benchmark solution for 2-D and 3-D cavity, flows are provided by Ghia et al. [80] and Wong & Becker [81], respectively. Since free surface or boundary deformation is not involved, the simulation is performed with fixed particles without any particle motion. Since the flow regime is laminar with $Re = 400$, no turbulence model has been used. The numerical and physical parameters for lid-driven cavity simulation are given in Table 3.5, and the horizontal velocity profile along the vertical axis centreline is compared to benchmark data in Figure 3.9.

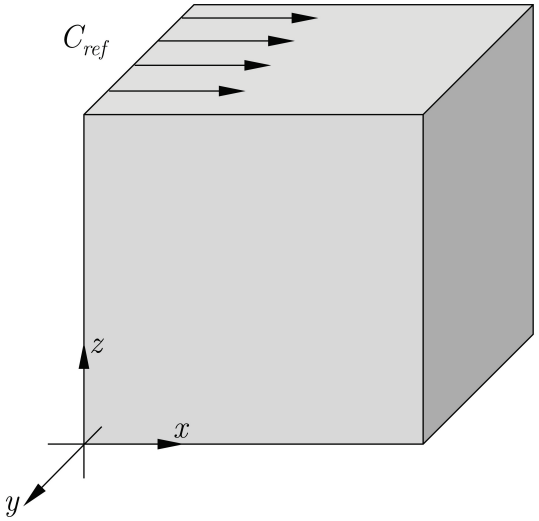


Figure 3.8. Schematic outline of the 3-D lid-driven cavity. The flow is surrounded in a cube by wall boundaries. The bottom and side walls are stationary, while the top wall is moving with a constant reference velocity C_{ref} along the horizontal axis. A vortical flow is therefore formed due to a shear force applied by the moving wall boundary.

Table 3.5. The numerical and physical parameters for lid-driven cavity

parameter		value
Courant number	CFL	0.80
Particle overlap ratio	ℓ	0.85
Velocity correction coefficient	λ	0.125
Wall velocity	C_{ref}	1.0 m·s^{-1}
Fluid density	ρ_{ref}	1.0 kg·m^{-3}
Dynamic viscosity	μ	0.0025
Reynolds number	Re	400

Turbulent flow inside a pipe

Two test cases: *i*) flow inside a circular pipe and *ii*) open channel flow has been carried out by GPU-SPHEROS to verify the implementations of k-ε and k-ω SST. All the validations are performed at $Re = 10^4$ (computed based on pipe diameter), and the results are compared to the FVM-based ANSYS CFX and Fluent results. The schematic of both test cases is shown in Figure 3.10. Since the implemented eddy viscosity models have been widely validated for different test cases during the past decades [66], the aim of these test cases is only to verify the implementation of models and integration into ALE-based FVPM by comparing the GPU-SPHEROS results to ANSYS CFX as a reliable commercial solver. Therefore, no experimental data have been involved in the turbulence validation process.

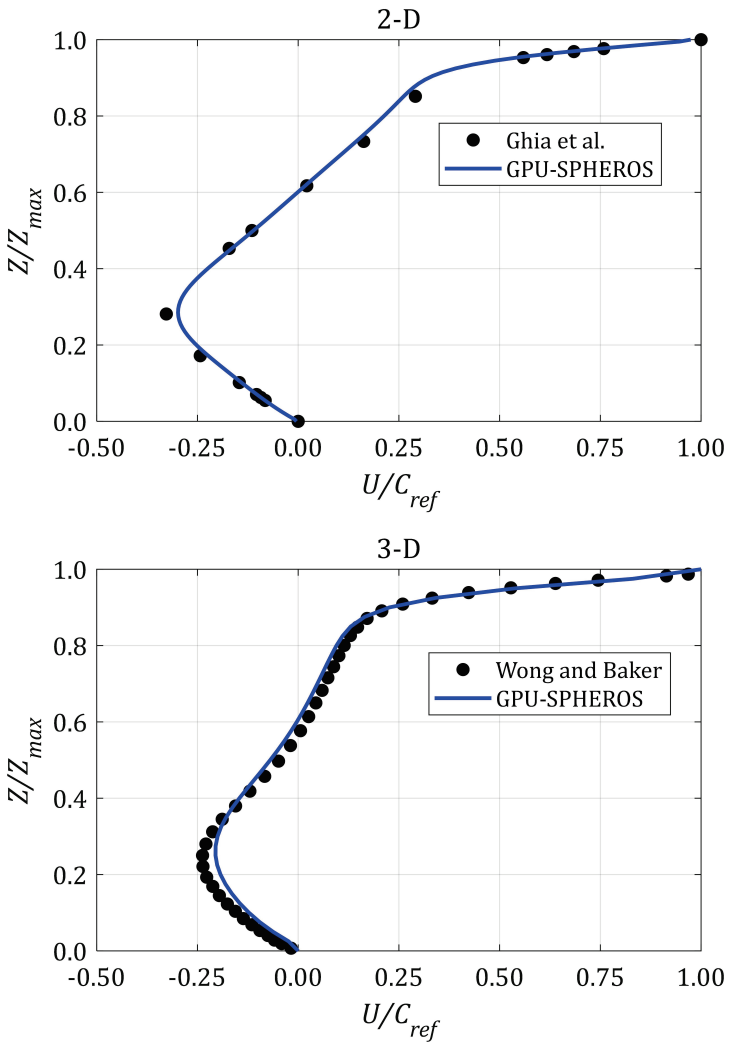


Figure 3.9. The horizontal velocity U profile along the vertical axis Z centreline for 2-D and 3-D lid-driven cavity with fixed particles with the 40x40x40 grid at $Re = 400$. The benchmark data are provided by Ghia et al. [80] and Wong & Baker [81].

Table 3.6. The numerical and physical parameters for pipe and open channel flow.

parameter		value
Courant number	CFL	0.75
Particle overlap ratio	ℓ	0.85
Velocity correction coefficient	λ	0.125
Inlet velocity	C_{ref}	1.0 m·s^{-1}
Inlet turbulence intensity	I_{turbe}	5%
Reference density	ρ_{ref}	1.0 kg·m^{-3}
Dynamic viscosity	μ	10^{-4} Pa·s
Reynolds number	Re	10^{4}

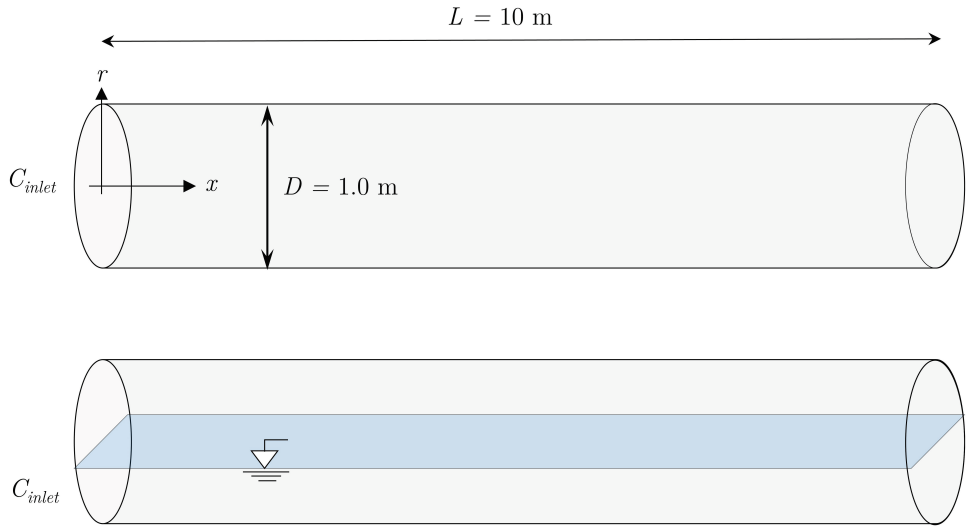


Figure 3.10. Circular pipe (top) and open channel flow (bottom) verification test cases. Since there is no particle to cover the gas phase in GPU-SPHEROS, only the liquid particles are injected into the domain, and the impact of the gas phase is neglected.

In this test case, the turbulent flow in a circular pipe has been simulated with k-ε and k-ω SST turbulence models. The pipe length is 10 times greater than pipe diameter, which letting the flow to develop ($L = 10D = 10$ m). The inlet velocity C_{inlet} is 1.0 m·s^{-1} (see Figure 3.10 top) and the pipe treats as a no-slip impermeable boundary. The numerical and physical parameters for pipe and open channel flow simulation are given in Table 3.6.

The turbulence variables are sampled at the end of the pipe, wherein the turbulent boundary layer is fully developed [92]. Using ALE-based FVPM with both fixed and moving particles, the predicted horizontal velocity U and turbulence kinetic energy k along the vertical centerline are in good agreement with Eulerian FVM-based ANSYS CFX and Fluent results (see Figure 3.11 to Figure 3.13). The spatial resolution is $n_p = 30$ in which n_p is defined as,

$$n_p = \frac{D}{\delta} \tag{3.2}$$

Given that the solver has been verified for fixed particles case in the standard model, the realizable model is only verified for moving particles case.

For the test case with fixed-particle, the distribution of the particles is uniform during the simulation, and the volume integral gradient approach, which is used to compute the velocity gradients at the center of the particles, provides second-order accurate results. These gradients are then used to compute the turbulence production P_k. However, on the other hand, in the case in which the particles are moving, the computed gradients are not 2nd-order accurate, but first-order. The production term is then affected by this accuracy difference compared to the fixed-particle case. Moreover, unlike the fixed-particle case, when the particles move, the volume is changed due to the variant particle distribution. This induces an error compared to

the fixed-particle case. However, since the mesh elements remain stationary during the simulation, the fixed-particle setup is closer to ANSYS CFX setup.

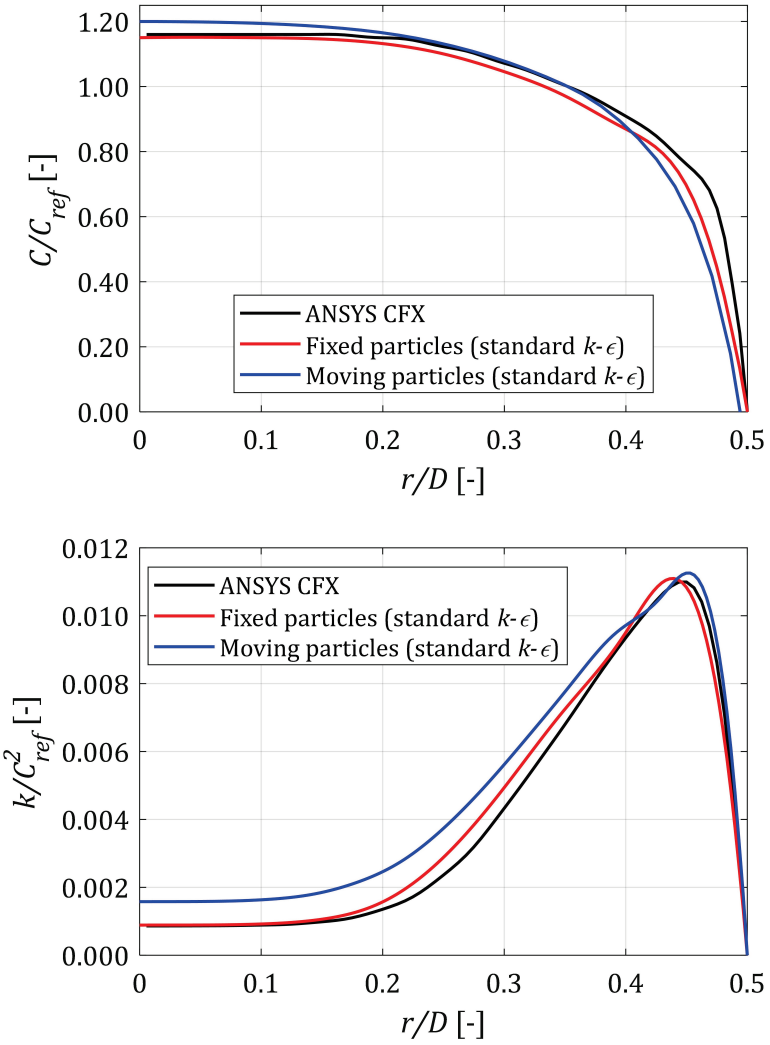


Figure 3.11. Velocity profile (top) and turbulence kinetic energy k (bottom) of developed flow inside a circular pipe with standard k-ε at $x = L$; FVPM with fixed and moving particles vs. FVM.

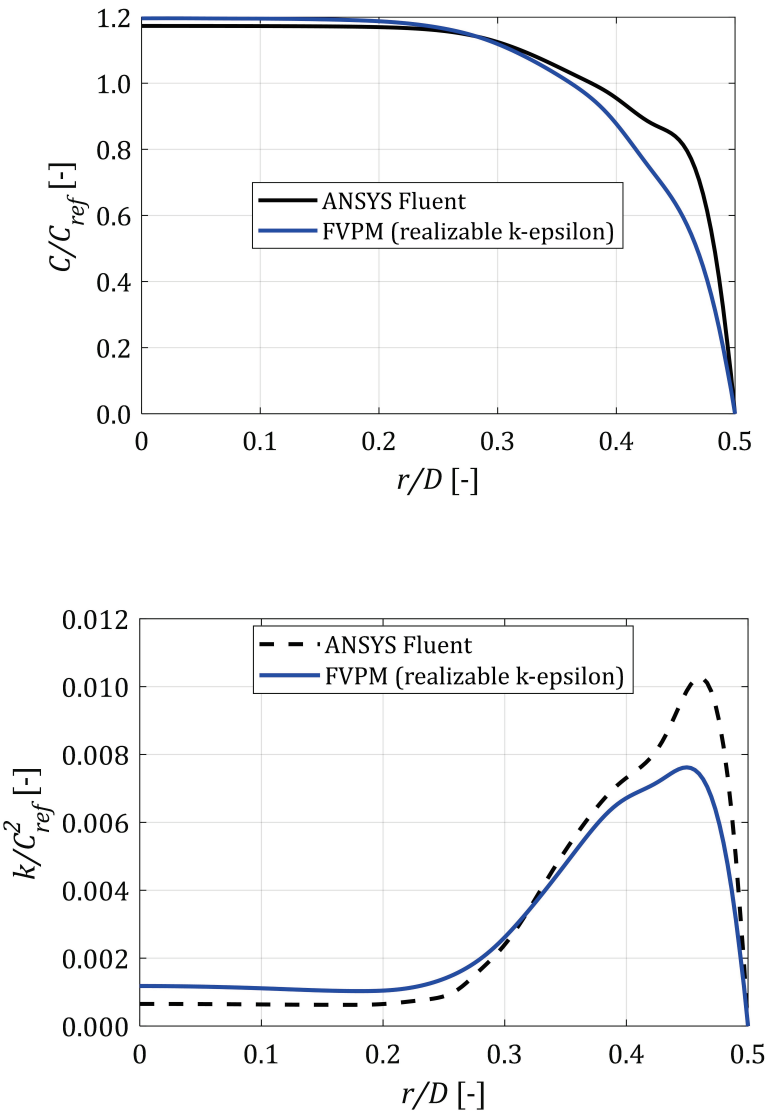


Figure 3.12. Velocity profile (top) and turbulence kinetic energy k (bottom) of developed flow inside a circular pipe with realizable k-ε at $x = L$; FVPM with moving particles compared to FVM.

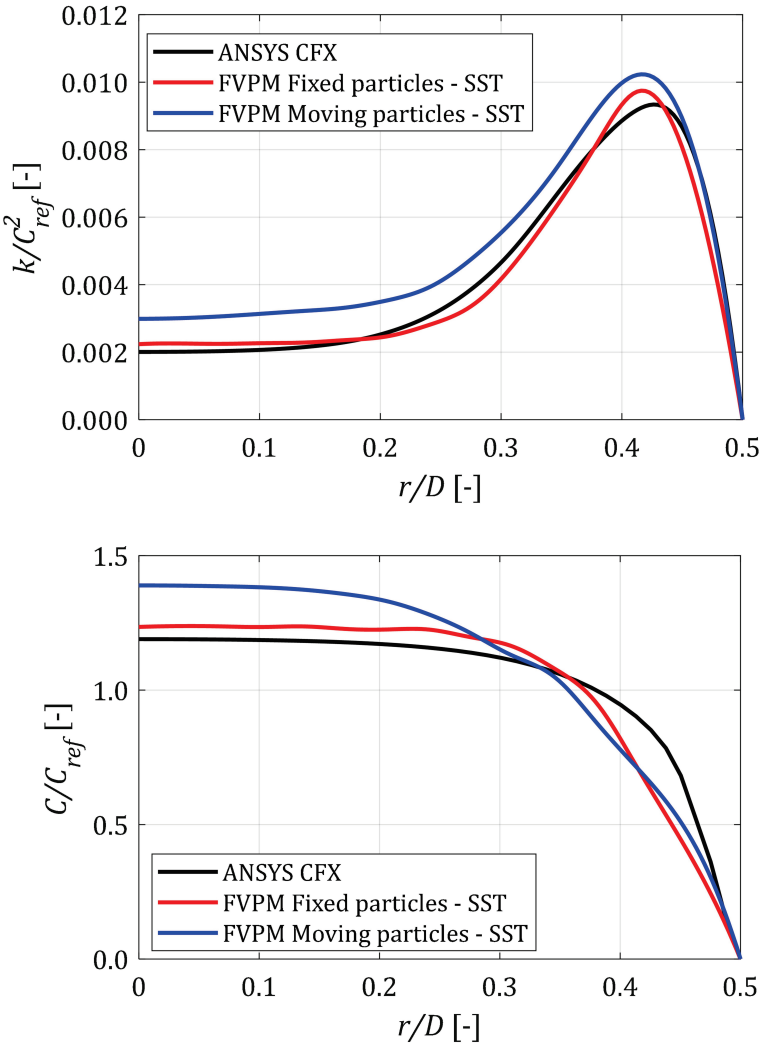


Figure 3.13. Velocity profile (top) and turbulence kinetic energy k (bottom) of developed flow inside a circular pipe with SST at $x = L$; FVPM with fixed and moving particles compared to FVM.

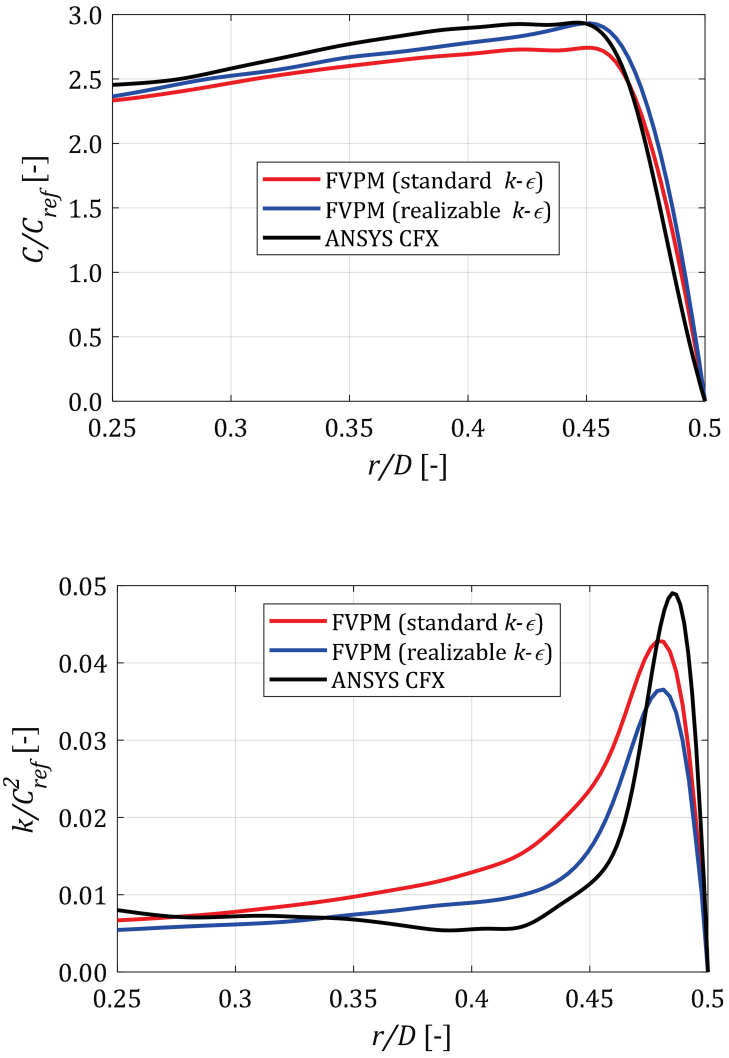


Figure 3.14. Velocity profile (top) and turbulence kinetic energy k (bottom) of open channel flow with standard and realizable k-ε models at $x = L$; FVPM with moving particles vs. FVM with fixed-size grid. The data have been extracted from the wall (i.e., $r/D_{channel} = 0.5$) to the free surface ($r/D_{channel} \approx 0.25$).

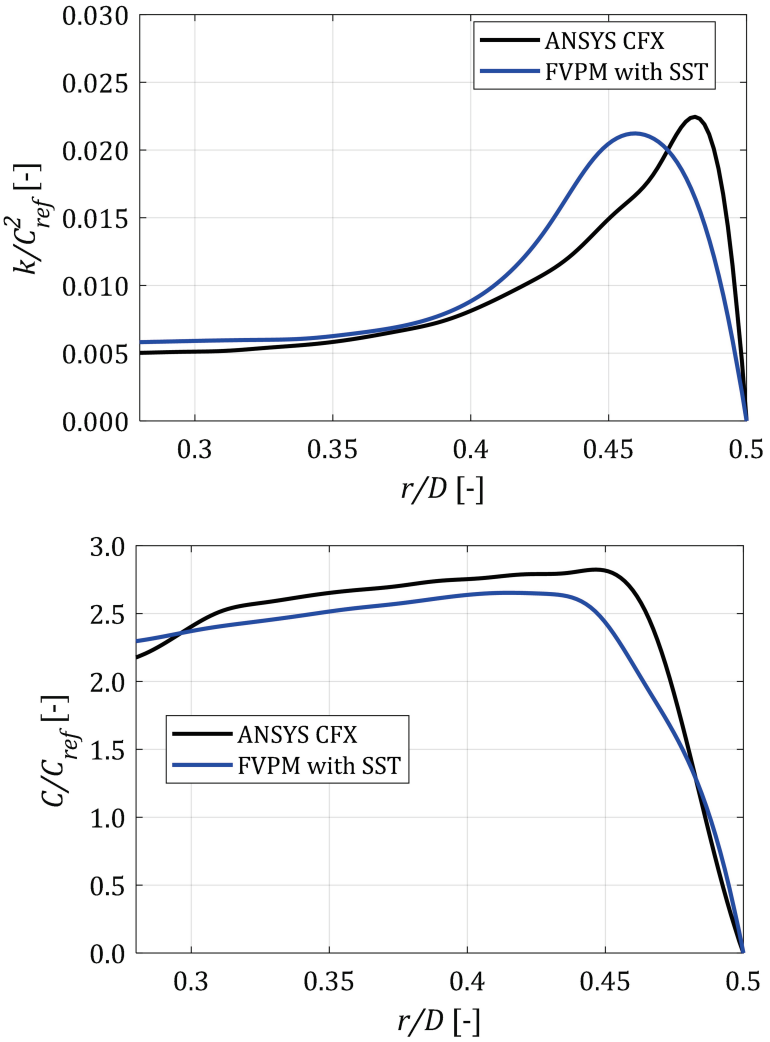


Figure 3.15. Velocity profile (top) and turbulence kinetic energy k (bottom) of open channel flow with SST model at $x = L$; FVPM with moving particles vs. FVM with fixed-size grid. The data have been extracted from the wall (i.e., $r/D_{channel} = 0.5$) to the free surface ($r/D_{channel} \approx 0.25$).

For pipe flow simulation, pressure waves are generated due to the weakly compressible hypothesis used in the FVPM method. These waves are reflected from the outlet boundary and can significantly affect the solution [20]. To reduce these pressure waves, a sink term is added to the right-hand side of momentum equation,

$$\frac{d}{dt}\left(\rho_i V_i \mathbf{C}_\mathrm{i}\right) = \sum_j \left[\left(\rho \mathbf{C} \otimes \dot{\mathbf{x}} - \rho \mathbf{C} \otimes \mathbf{C}\right)_\mathrm{ij} - p_\mathrm{ij}\mathbf{I} + \mathbf{s}_\mathrm{ij}\right] \cdot \boldsymbol{\Delta}_\mathrm{ij} - p_b \mathbf{B}_\mathrm{i} - s(\mathbf{x}_\mathrm{i})\left(\mathbf{U}_\mathrm{i} - \mathbf{U}_\mathrm{ref}\right) \tag{3.3}$$

$s(\mathbf{x}_\mathrm{i})$ is the damping strength which is defined as a linear function of x for the present case,

$$s(x) = 10\,\frac{x_i - x_\mathrm{min}}{x_\mathrm{max} - x_\mathrm{min}} \tag{3.4}$$

where x_min and x_max denotes the minimum and maximum of x of the damping zone applied to the pipe. The zone in which the sink term applied is called the damping zone. For pipe

flow, to apply the damping zone, the pipe length has been extended by 20% and the damping zone applied to the end of the pipe where $x \geq 1.1L$.

3.3.3 Circular open channel flow

The next test case is an open channel flow, which includes free surface effects. A schematic of the test case is shown in Figure 3.10 (bottom). The velocity C and turbulence kinetic energy k have been compared to ANSYS CFX solver results for implemented turbulence models. The results show reasonably good agreement, although the FVPM particles are moving (see Figure 3.14 and Figure 3.15). Since GPU-SPHEROS is a single-phase solver, the gas phase is not covered by the solver. On the other hand, in CFX, both gas and liquid phases are covered with the two-phase flow model, and the free surface has been modeled with VOF. The spatial resolution for the open channel simulations is similar to pipe with $n_p = 30$.

3.3.4 Impinging jet on a flat plate

Impinging jet on a flat plate is chosen as a free surface flow case featuring hydrodynamics close to the Pelton turbine flow. The velocity, pressure coefficient C_p, and free surface location are validated against ANSYS CFX, and experimental data measured by Kvicinsky et al. [5]. The jet particles are injected as an inlet boundary with $n_p = 30$, and the simulations have been performed with both uniform and non-uniform velocity profiles. The numerical and physical parameters impinging jet simulation are given in Table 3.7.

Table 3.7. The numerical and physical parameters for the impinging jet test case.

parameter		value
Courant number	CFL	0.75
Particle overlap ratio	ℓ	0.75
Velocity correction coefficient	λ	0.125
Reference velocity for uniform jet	$C_{ref}^{uniform}$	4.0 m·s^{-1}
Reference velocity for non-uniform jet	$C_{ref}^{nonuniform}$	19.81 m·s^{-1}
Inlet turbulence intensity	I_{turb}	5%
Fluid density	ρ_{ref}	10^3 kg·m^{-3}
Dynamic viscosity	μ	10^{-3} Pa·s
Reynolds number	Re	10^4

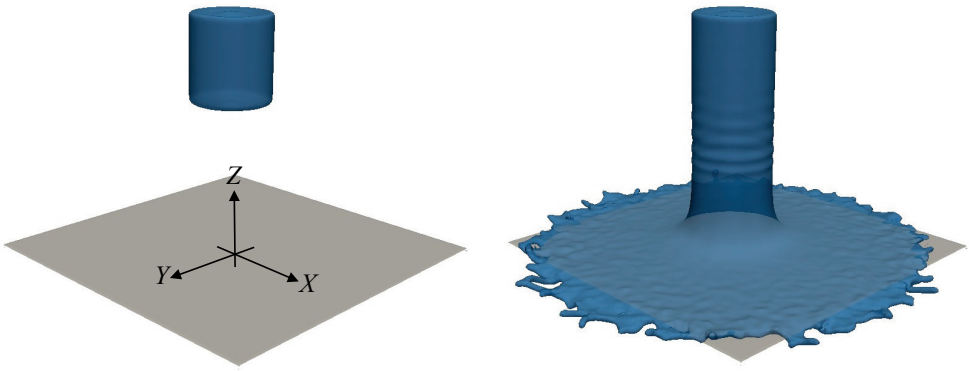


Figure 3.16. The reconstructed free surface of the water jet impinging on a flat plate at $Re = 1.2 \times 10^5$ with uniform inlet velocity profile $C_2 = 4.0$ m s^{-1}.

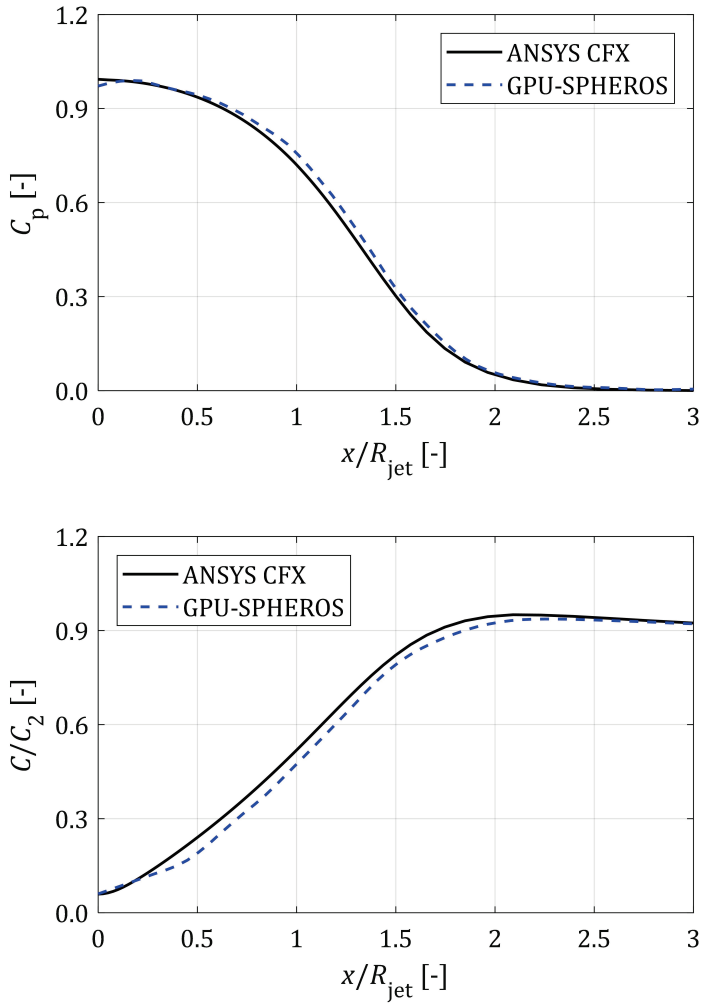


Figure 3.17. The pressure coefficient (top) and velocity ratio (bottom) along the X-axis at $Z = 3.3D_j \times 10^{-3}$ above the wall for a jet impinging on a flat plate. The jet velocity and the Reynolds number are $C_2 = 4.0$ m·s^{-1} and $Re = 1.2 \times 10^5$, respectively. The inlet velocity profile is uniform, and the pressure is averaged in the period of 0.05 s to 0.1 s to filter out the pressure oscillations.

Case a) impinging jet with a uniform velocity profile

In this case, a jet with uniform velocity profile impinges perpendicularly on a plate which is considered as a no-slip impermeable wall boundary. The center of the plate corresponds to the center of the Cartesian coordinate system. The jet inlet has the diameter $D_2 = 0.03$ m located at $Z_{jet} = 2.5D_2$ above the flat plate, and the jet velocity is $C_2 = 4.0$ m s^{-1} with the Reynolds number based on the jet diameter $Re = 1.2 \times 10^5$. The pressure coefficient C_p and velocity profile are sampled along the X-axis for validation. The pressure time-history is averaged over the period from 0.05 sec to 0.1 sec to filter out the pressure oscillations caused by the artificial compressibility. The reconstructed free surface, as well as the pressure and velocity distribution along the X-axis on the plate, are shown in Figure 3.16 and Figure 3.17, respectively. As indicated, there is a good agreement between the result of GPU-SPHEROS and ANSYS CFX 18.1 with less than 3% difference for both pressure and velocity.

Case b) impinging jet with a non-uniform velocity profile

In the second test case, a non-uniform velocity profile of the water jet injected by a nozzle with a reference velocity $C_2^{ref} = 19.81$ m·s^{-1} is used as inlet boundary with the same dimensions as the previous test case. The velocity distribution is shown in Figure 3.18. The pressure coefficient along the X-axis as well as the free surface elevation are successfully validated against Kvicinsky's experimental results [5], as shown in Figure 3.19, with again a fairly well agreement with less than 2% difference.

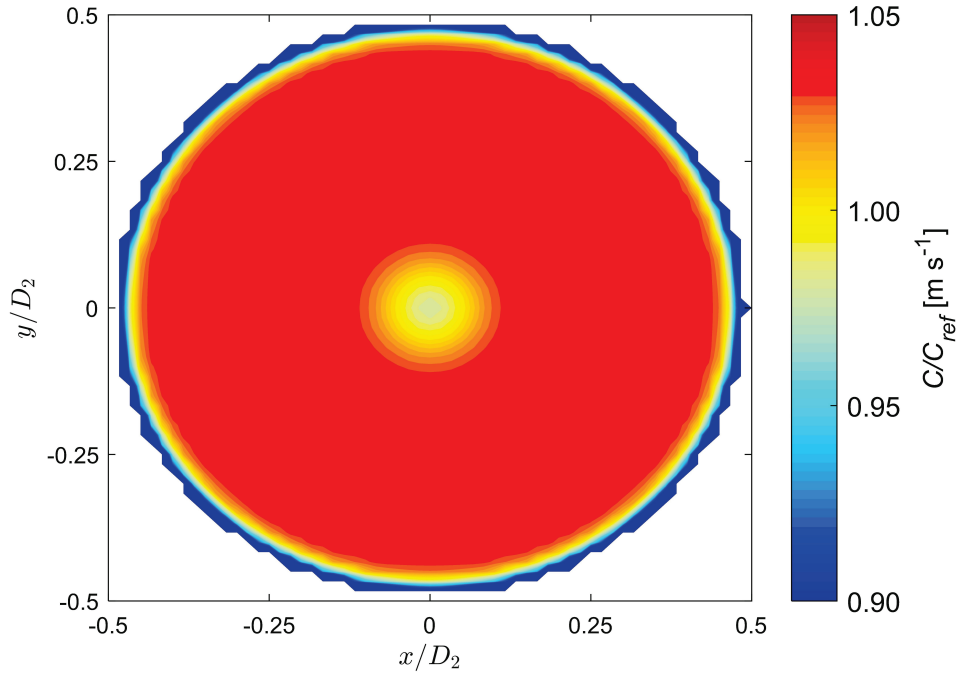


Figure 3.18. The experimental jet velocity distribution at the injector outlet measured by Kvicinsky et al. [5].

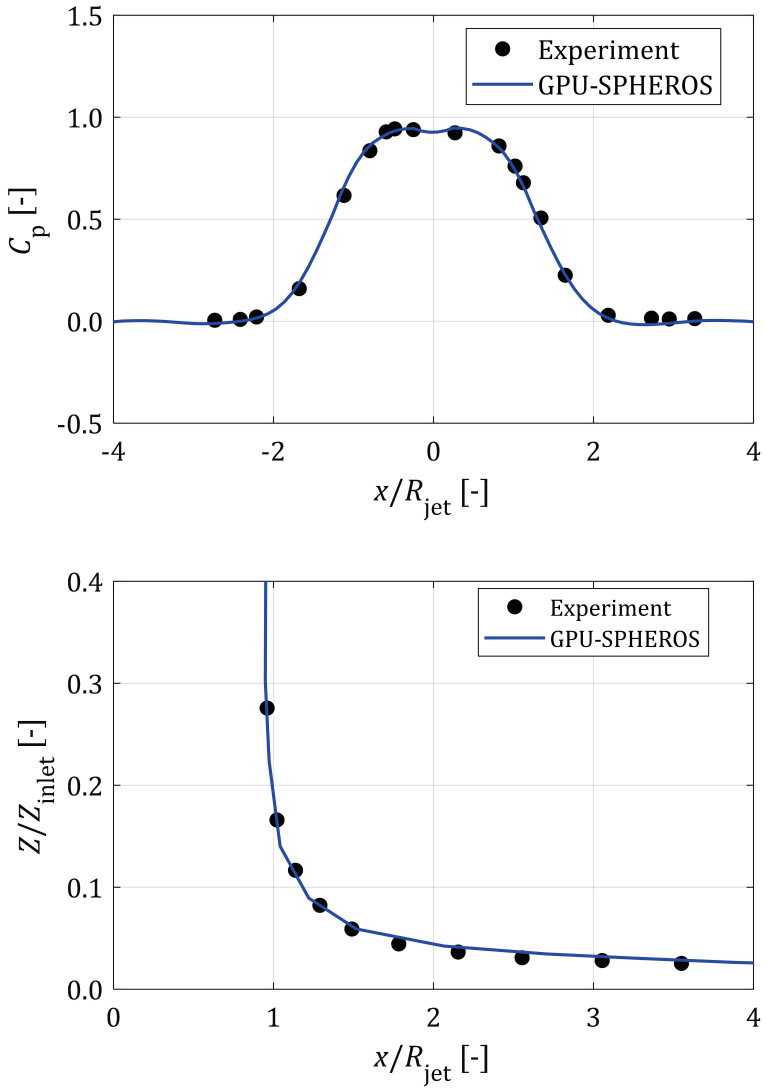


Figure 3.19. The pressure coefficient C_p (top) and free surface elevation (bottom) along the X-axis on the wall for a jet with the non-uniform velocity profile given in [5], impinging on a flat plate. The jet velocity and the Reynolds number are $C_v^{ref} = 19.81$ m·s^{-1} $C_2 = 19.81$ m·s^{-1} and $Re = 5.95 \times 10^5$, respectively. The available experimental data provided by Kvicinsky et al. [5], [6]. The pressure time-history is averaged in the period of 0.05 s to 0.1 s to filter out the pressure oscillations.

Table 3.8. The geometry specification and operating condition for the rotating Pelton turbine validation test case

Parameter		Value
Discharge per nozzle	Q_0	23.01×10^{-1} m$^3\cdot$s^{-1}
Runner rotational speed at BEP	N_{BEP}	534 min^{-1}
Bucket width	B_2	0.130 m
Runner pitch diameter	D_1	0.450 m
Center of inlet	$\mathbf{X}^{inlet}_{center}$	(0.2201, -0.2250, 0.0) m
Jet velocity	C_2	25.91 m$\cdot$s^{-1}
Jet diameter	D_2	33.6×10^{-3} m
Number of buckets	z_b	22

Table 3.9. The numerical and physical parameters for a single jet rotating Pelton flow simulation.

parameter		value
Courant number	CFL	0.75
Particle overlap ratio	ℓ	0.75
Velocity correction coefficient	λ	0.125
Reference velocity for uniform jet	$C^{uniform}_{ref}$	4.0 m·s^{-1}
Reference velocity for non-uniform jet	$C^{nonuniform}_{ref}$	Based on [82]
Inlet turbulence intensity	I_{turb}	Based on [82]
Fluid density	ρ_{ref}	10^3 kg·m^{-3}
Dynamic viscosity	μ	10^{-4} Pa·s
Reynolds number	Re	10^4

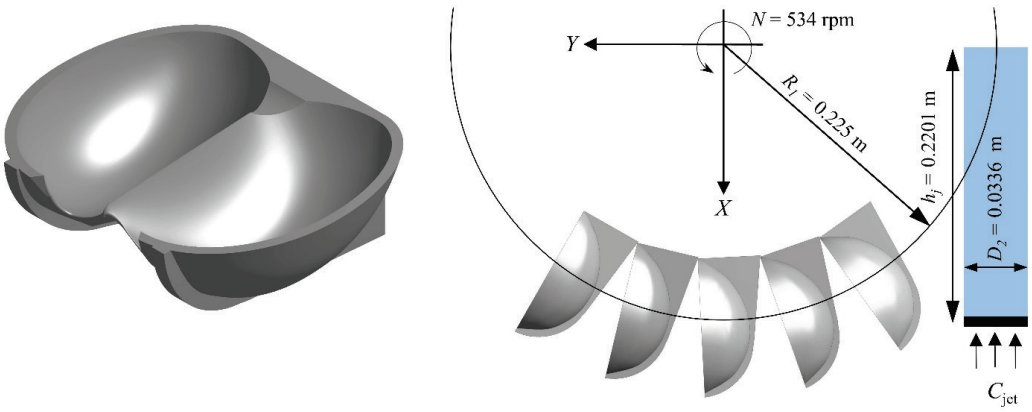


Figure 3.20. The geometry of the bucket (left) and a schematic of the single jet validation test case for rotating Pelton buckets (right).

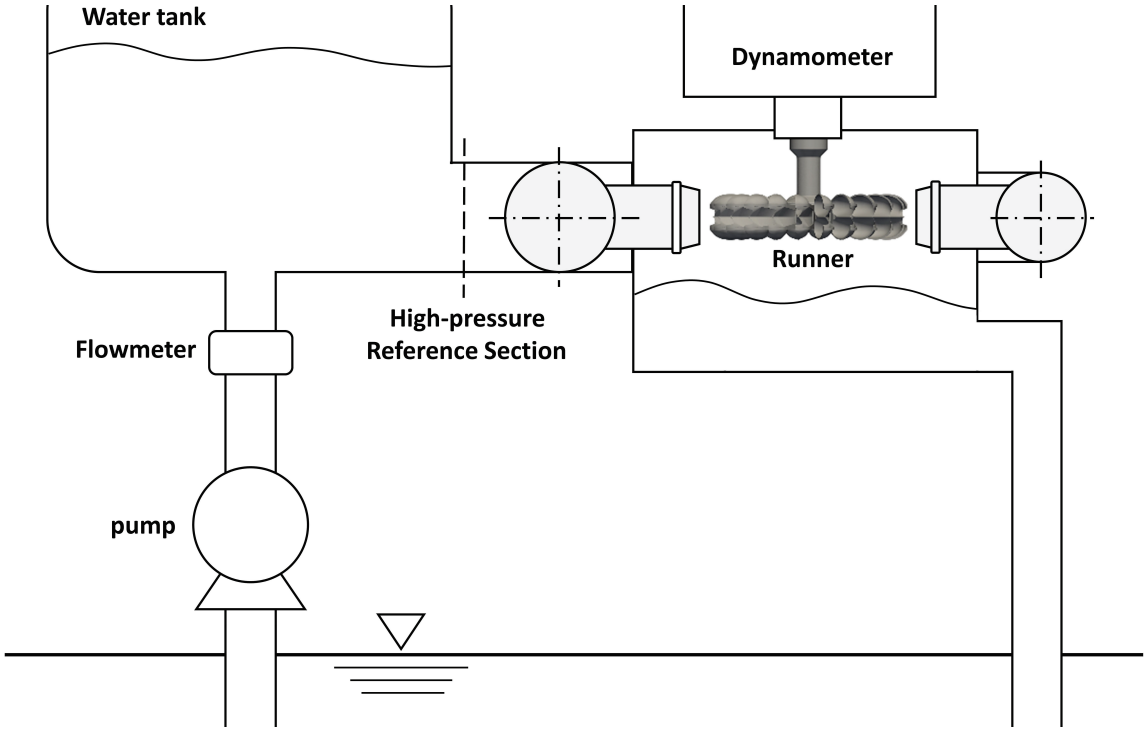


Figure 3.21. The simplified test rig structure in the Hydraulic R&D laboratory, Hitachi Mitsubishi Hydro Corporation consistent with the standards given by International Electrotechnical Commission, IEC 60193:1999, Hydraulic turbines, storage pumps, and pump-turbines – Model acceptance test.

The GPU-SPHEROS simulation results are compared with the results of ANSYS CFX 18.1, as well as model test torque measurements. All the Pelton runner model tests in the current study have been performed at the Hydraulic R&D laboratory, Hitachi Mitsubishi Hydro Corporation. The test rig fulfills all the general items and conditions described by the International Electrotechnical Commission standard, IEC 60193:1999 Hydraulic turbines, storage pumps, and pump-turbines – Model acceptance tests. The tests have been performed in an open test loop where the Pelton runner model is connected to a DC dynamometer to measure the torque. The rated head H is regulated by a circulating pump, whereas the runner rotational speed is adjusted by the dynamometer speed control. The flow discharge Q is measured by an electromagnetic flowmeter. The dynamic pressure $0.5 \times \rho \times C^2$ is calculated based on the reference section area and the flow discharge, whereas the static pressure p_{static} is measured by a manometer. The total pressure p_{total} is then used to calculate the rated head H based on the specific energy balance. The runner torque T and rotational speed N are measured by load cells and an electromagnetic counter, respectively. The efficiency η is calculated based on the supplied power P and hydraulic power P_h. All the measuring equipment is regularly calibrated based on the standards which have the traceability to the national organizations. The simplified structure of the test rig is shown in Figure 3.21.

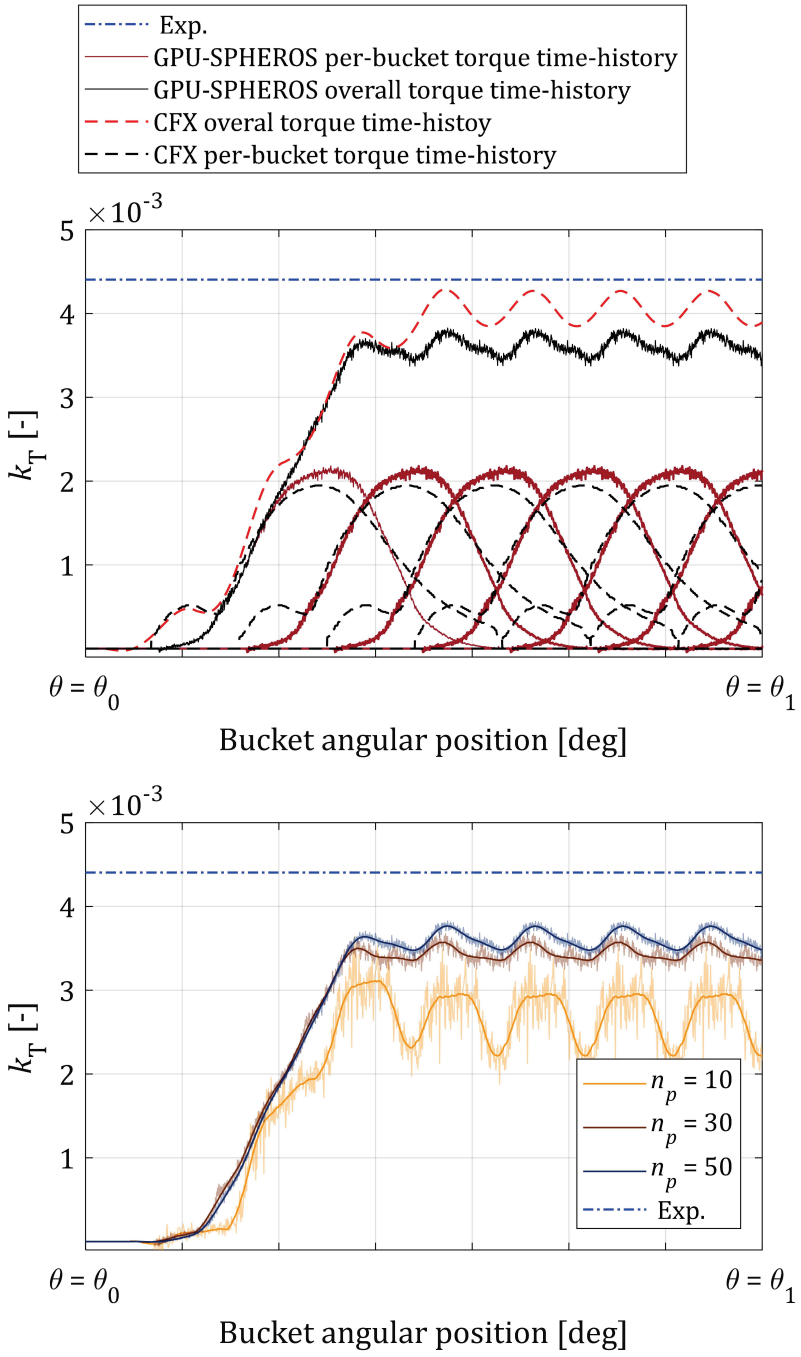


Figure 3.22. Overall torque time-history for a single jet rotating Pelton runner with $n_p = 50$ (top) and the torque convergence study with $n_p = 10$, 30 and 50 (bottom). A curve is fitted from the raw data using the Savitzky-Golay filter. To take advantage of the geometric periodicity for reducing the computational cost, only the torque of the intermediate bucket is applied, and time-shifted torque copies accounting for the bucket periodicity are then superimposed and integrated to compute the overall runner torque time-history. All the numerical simulations, as well as the experimental measurement, have been performed at BEP.

Table 3.10. The computed convergence rate for the time-averaged rotating runner torque. The error computation procedure is based on [94].

parameter	value
δ_1	0.000672 m
δ_2	0.001120 m
δ_3	0.003360 m
$r_{21} = \frac{\delta_2}{\delta_1}$	1.6667 (-)
$r_{32} = \frac{\delta_3}{\delta_2}$	3.0 (-)
$\Pi_{21} = k_T^{\delta_2} - k_T^{\delta_1}$	-0.00017415 (-)
$\Pi_{32} = k_T^{\delta_3} - k_T^{\delta_2}$	-0.00075939 (-)
$p_c = \frac{1}{\ln\left(r_{21}\right)} \times \left\| \ln\left\|\frac{\Pi_{32}}{\Pi_{21}}\right\| + \ln\left(\frac{r_{21}^{p_c} - \mathrm{sign}\left(\frac{\Pi_{32}}{\Pi_{21}}\right)}{r_{32}^{p_c} - \mathrm{sign}\left(\frac{\Pi_{32}}{\Pi_{21}}\right)}\right) \right\|$	0.84

The numerical simulation has been performed at the Best Efficiency Point (BEP) for three different particle number discretizations values $n_p = \{10,30,50\}$ to investigate the influence of the particle size on torque convergence. The overall dimensionless torque factor k_T time-history is shown against the bucket angular position θ in Figure 3.22 where the torque factor is defined as,

$$k_T = \frac{T}{\frac{1}{2}\rho D_1^3 C_2^2} \tag{3.5}$$

There are noticeable differences between the individual bucket torque time-history captured by both solvers. The bucket maximum torque computed by FVPM is higher than the torque calculated by CFX (Kumashiro et al. [82]), while, unlike CFX, it does not predict any torque peak at the beginning of jet-bucket interaction. GPU-SPHEROS also predicts a faster torque decrease after the peak. Overall, both solvers underpredict the average torque, CFX by about 4% and GPU-SPHEROS by 8.5%. The CFX simulation has been performed with a fine grid in the bucket boundary layer with $y^+ \approx 250$ and k-ω SST turbulence model, while GPU-SPHEROS uses a uniform particle size for fluid and wall boundaries ($y^+ \approx 600$) with k-ε turbulence model. In GPU-SPHEROS, a central difference scheme is used to mitigate the numerical diffusion, which could account for this underprediction. Although the underestimation can be reduced from 8.5% to almost 7% by refining the particle spacing to $n_p = 50$, the corresponding computational cost is multiplied by almost 8 since the complexity is of $O(n_p^4)$. The power 4 in the order of complexity is derived based on uniform spatial grid refining in three directions, x, y, and z, and accordingly one order smaller time step applied by spatial refinement. Also, the convergence rate for time-averaged torque is computed and given in Table 3.10. Unlike SPHEROS, which uses upwinding for the convection term, in GPU-

SPHEROS, the Central Difference Scheme (CDS) is used for the convective term computation, which is less dissipative. This leads to an improved time-averaged torque convergence rate from $p_c^{upwind} = 0.39$ in [36] to $p_c^{CDS} = 0.84$ in the current simulations. Altogether, since the torque factor is not altered significantly by refining the particle size from $n_p = 30$ to $n_p = 50$, all the multi-jet runner simulations in the next chapter are performed with $n_p = 30$.

Unlike mesh-based methods, particle-based methods are robust in free surface tracking without inducing excessive numerical diffusion and requirement local grid refinement at the interphase. The FVPM numerical data of the performed Pelton flow simulation is used to visualize the jet-bucket interaction and track the free surface. The reconstructed free surface of the deviated water jet by rotating buckets is shown in Figure 3.23 for different runner angular positions.

3.3.4.1 Water jet with non-uniform velocity profile and turbulence intensity

In this case, a non-uniform jet velocity profile, as well as turbulence variables at the inlet given by [82], is used for simulation. These data are provided by distributor simulation with ANSYS CFX. The velocity profile and turbulence variables k and ε are shown in Figure 3.25. The simulation is performed by both standard and realizable k-ε models and the torque is compared with the experimental data. A Central Difference Scheme (CDS) is used for discretization of the convective term, which is second-order accurate but leads to oscillations, and therefore the raw torque is noisy. Since the CFX is a multi-phase solver, the solution covers both air and water and the values out of the water jet border in Figure 3.25 corresponds to the air.

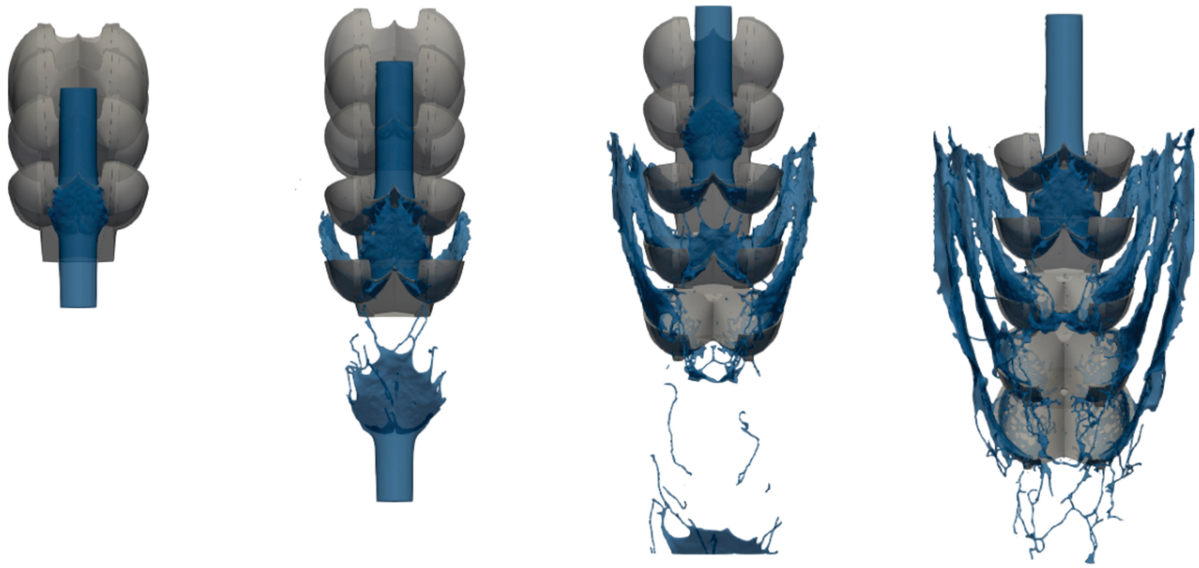


Figure 3.23. The reconstructed water jet free surface deviated by rotating buckets at different runner angular positions θ. The injected jet velocity profile is uniform, and standard k-ε model has been used to capture the turbulence effects. The free surface has been visualized in Paraview open-source data analysis and visualization application.

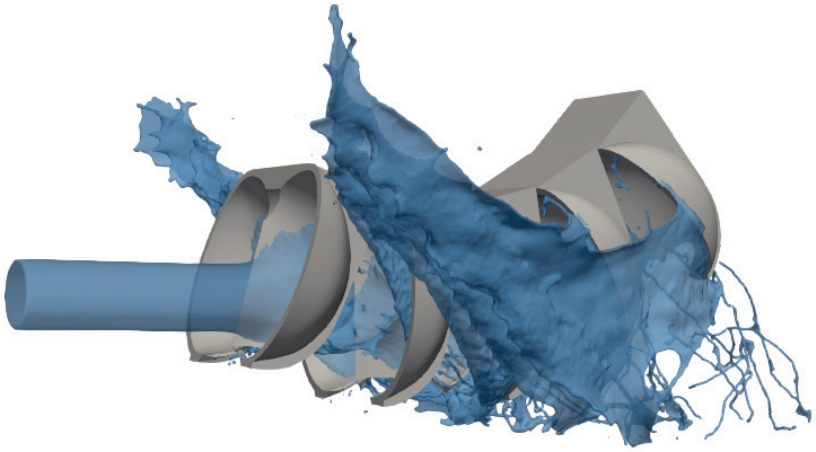
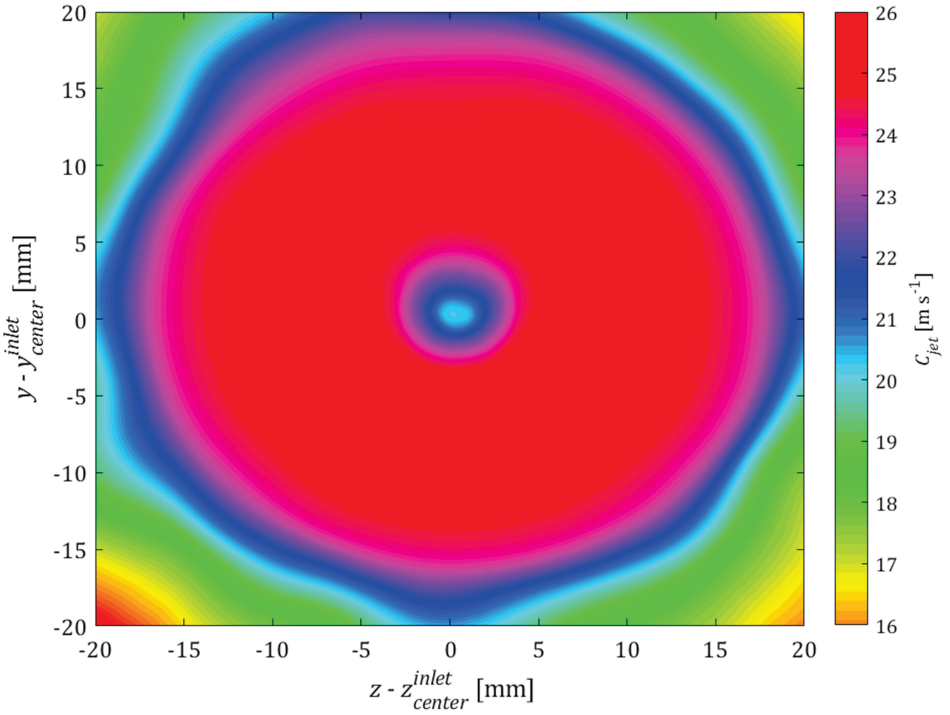
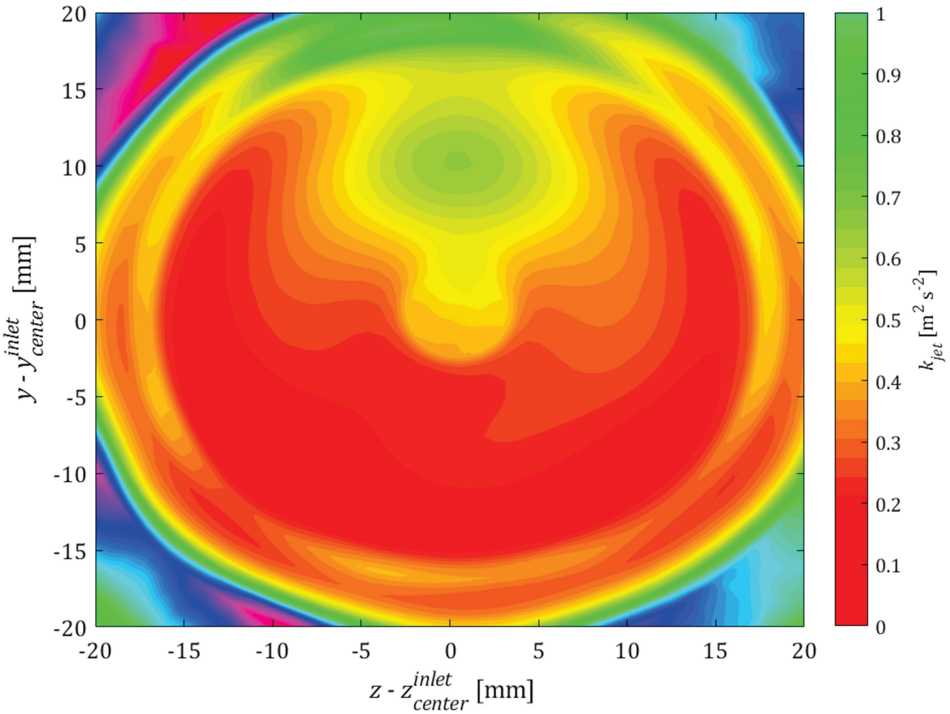


Figure 3.24. The reconstructed free surface of the deviated water jet by rotating Pelton buckets at $\theta = 110°$ with standard k-ε and uniform jet velocity profile.

Although the realizable model is known to provide more accurate results than the standard model for swirling or separated flows, both models give similar results for the case of the Pelton turbine flow. The simulation has only been performed with standard and realizable k-ε models. In theory, the k-ω SST model switches over to high-Re k-ε model for high y^+ values and many advantages of this model compared to k-ε are lost when a coarse grid is used for near-wall zone (with $y^+ < 30$). Furthermore, unlike the k-ε model, in SST, the nearest wall distance d is required for transition between the low-Re k-ω and high-Re k-ε models which are blended based on the distance from the nearest wall. In mesh-based methods, d is computed at the beginning of the simulation and used for all the next time steps while, on the other hand, for particle-based methods, since the particles are moving and their distance to the nearest wall is altered, d should be computed at every time step. A higher computational cost is therefore applied by per-timestep distance computations. In general, k-ω SST is preferred for problems in which the viscous sublayer or buffer layer need to be resolved; otherwise, the method works similar to k-ε in the far field. However, this is not applied to the Pelton turbine flow. The k-ε model is simple, affordable, and reasonably accurate for a wide range of flows.

Since both standard and realizable models provide almost same results for torque, the standard model is used for all the multi-jet Pelton flow simulations performed in the next chapter.


20
15
10
5
0
-5
-10
-15
-20
-20 -15 -10 -5 0 5 10 15 20
$y - y^{inlet}_{center}$ [mm]
$z - z^{inlet}_{center}$ [mm]
26
25
24
23
22
21
20
19
18
17
16
C_{jet} [m s^{-1}]


20
15
10
5
0
-5
-10
-15
-20
-20 -15 -10 -5 0 5 10 15 20
$y - y^{inlet}_{center}$ [mm]
$z - z^{inlet}_{center}$ [mm]
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
k_{jet} [m^2 s^{-2}]

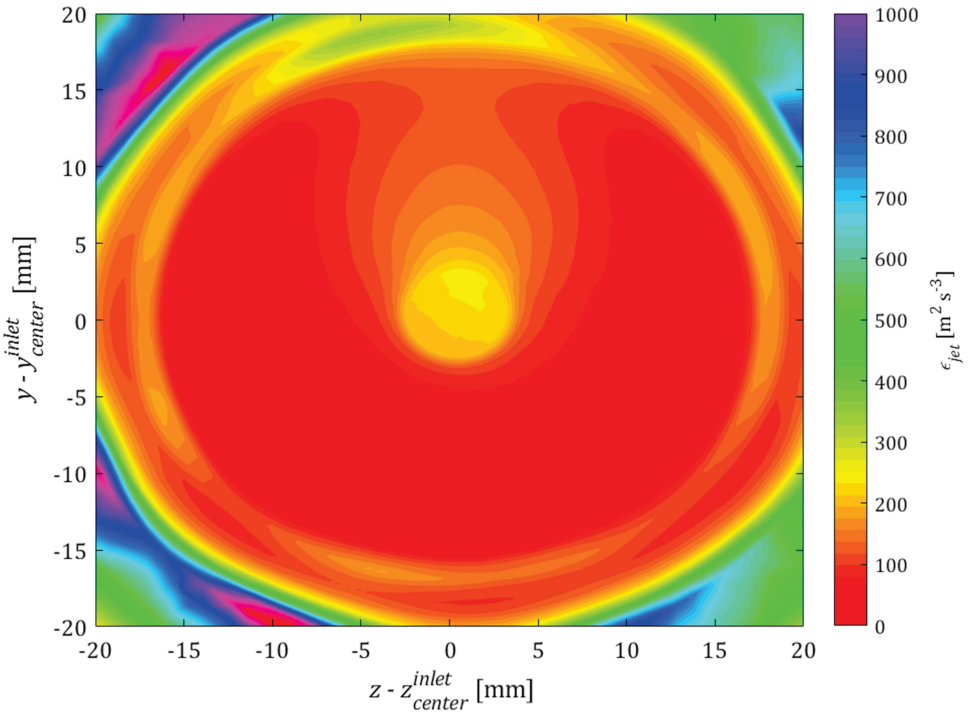


Figure 3.25. Jet velocity profile as well as turbulence variables contours at the inlet boundary. Since CFX solves a multi-phase domain, the values out of circular water jet border correspond to the air.

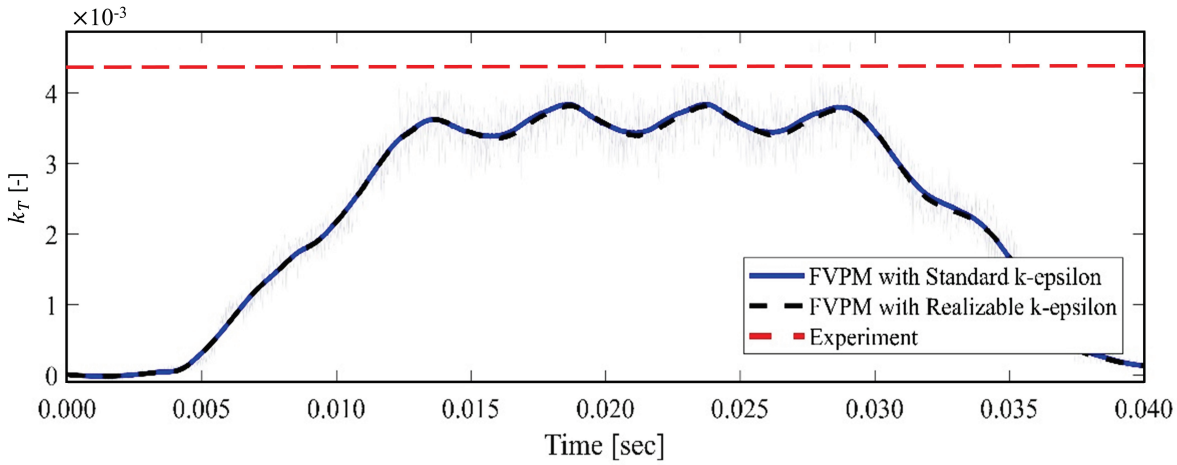


Figure 3.26. The Pelton torque time-history computed with both standard and realizable k-ε turbulence models against the experimental time-averaged torque. The blue and black lines are smoothed torque data derived from filtering the noisy raw data with the Savitzky-Golay method.

3.4 Discussion

In this chapter, the performance of code was visualized and evaluated within a naive roofline model for Tesla P100 SXM2 16GB GPU. The upper performance ceilings are derived based

on both hardware computational throughput and maximum bandwidth. Depending on where each kernel is located in the roofline chart, the performance limiter was determined, and appropriate optimization techniques were then applied.

The optimization priority is given to interaction vectors, which constitutes more than 90% of the overall running time before the optimization. An overestimated fixed-size memory, called memory pool, is allocated for the variable-size arrays to prevent costly memory operations in every time step. Each thread then reads or writes a pre-allocated memory address during the runtime, and the data are updated every iteration. However, to be able to cope with limited global memory size, the particles are batched, and the batches are then released sequentially for computations. Large batches provide enough parallelization to fill the GPU although they require larger memory space. Moreover, the order of the 2-D and 3-D arrays indices is carefully programmed to avoid strided memory access in which the memory transactions can be significantly increased. Each step of the algorithm is handled by a reasonably small separate kernel to avoid imbalanced overuse of GPU resources such as registers which can devastatingly limit the amount of parallelization.

The next optimization candidate was the overall neighbor search, particularly the distance check, which is performed by a single kernel constituted 85% of overall neighbor search running time. To improve the data access performance by the kernel, the particle data are gathered based on a coalesced map before passing into the kernel. Shared memory is then used in the kernel to retain the reused $\mathcal{P}_i$ particles data and diminish the risk of register spilling. The spilled registers into local memory or even caches have a remarkably higher memory latency than the registers themselves or shared memory. The flux and force computations and time integration together represent 5% of the overall computational time and has not been considered for optimization. Altogether, the optimized GPU solver is almost six times faster than its CPU version on a single NVIDIA® Tesla™ P100 GPU vs. a dual-setup Intel® Xeon® E5-2690 v4 Broadwell CPU node with 28 total physical cores.

Once the solver optimized, it has been validated for five different test cases. The test cases cover laminar and turbulence flow for both internal and free surface problems. First, the validity of the code for laminar flow is assessed by a lid-driven cavity simulation at $Re = 400$. The particles are kept fixed in the space, and the fluxes are exchanged between the neighbors. For this fixed-particles simulation, the neighbor list and interaction vectors are only computed once at the beginning of the simulation, but the fluxes and forces are updated in time. The results were validated against the workbench data for both 2-D and 3-D cases. To model the turbulence effects, k-ε and k-ω SST eddy viscosity models have been integrated into ALE-based FVPM with moving particles, and the wall function approach has been implemented and used for near-wall turbulence computations. The turbulence implementations into GPU-SPHEROS have been verified against the ANSYS CFX commercial solver results for pipe and open channel turbulent flows as internal and free surface setup. To prevent unrealistic excessive turbulence build-up around the stagnant zone, the Menter and Kato-Lander limiters were implemented for both models. Once the turbulence implementation was verified, the code

has been used for numerical simulation of the impinging jet on a flat plate featuring hydrodynamics close to the Pelton turbine flow. The simulation was performed with both uniform and non-uniform velocity profiles, and the turbulence limiters are used to prevent excessive turbulence build-up around the stagnation zone. A good agreement is shown between GPU-SPHEROS and both Kvicinsky [5] and CFX results, for the pressure, velocity, and free surface elevation.

As a final test case, GPU-SPHEROS was used to simulate rotating Pelton runner flow as a hydraulic turbine application. The torque time-history was computed for three different spatial resolution, and the convergence rate has been evaluated. The central difference scheme is used for convective term discretization to mitigate the numerical diffusion. The method is, however, first-order accurate showing $p_c = 0.84$ for the convergence slope. Although the predicted torque is underestimated, the convergence rate is more than twice faster than the CPU version (i.e., SPHEROS) which uses upwind scheme for convective flux discretization. Moreover, the geometry has been reconstructed by overlaying a set of spherical particles on the wall boundary, which results in a rugged reconstructed bucket surface. This can lead to a misprediction of outlet flow angles and consequently, momentum transfer.

As a particle-based method, the Pelton free surface flow can be simply tracked with no difficulty. This is shown by the jet-bucket interaction visualized in Paraview open-source data analysis application. Once the solver optimized and validated, it stands ready to be used for numerical simulations of multi-jet Pelton flow as an industrial-size problem.

4 GPU-Accelerated FVPM Simulation of Multi-jet Pelton Turbine Flow

In this chapter, the hydrodynamic performance of a six-jet Pelton turbine is investigated with GPU-SPHEROS in which the power of particle-based methods is exploited. The numerical simulations are performed at eight operating points ranging from $N / N_{BEP} = 89\%$ to $N / N_{BEP} = 131\%$ with N being the runner rotational speed. The torque and efficiency trends, as well as the specific speed range in which the jets interfere, are well-captured, which provides confidence in the use of the computational model for the design optimization of Pelton turbines. The simulations, in particular, show a significant torque and efficiency drop at high rotational speeds, due to jet interference as well as increased load fluctuations at the rotational speeds both smaller and larger than the N_{BEP} caused by jet disturbance, which is likely to amplify structural fatigue damage. Both phenomena are essential factors to take into account in the design process of a Pelton machine.

All the following computations have been performed on Piz Daint, one of the most potent GPU-powered supercomputers in the world equipped with 5'704 GPU nodes. Thanks to GPUs many-core architecture, the computational time has been reduced from weeks to days.

Part of this chapter is a reproduction and modification of a submitted research article to a peer-reviewed journal [96].

4.1 Working principles of Pelton turbines

A Pelton turbine is an impulse-type turbine in which the water potential energy is converted into kinetic energy to generate power. Torque is generated by high-speed water jet(s) impinging the rotating buckets mounted on the runner connected to the shaft. Simplified inlet and outlet velocity diagrams corresponding to the nozzle water jet impinging on a bucket are shown for the pitch diameter in Figure 4.1. The transformed specific energy is derived by the Euler equation, which reads,

$$E_t = U_1 \cdot Cu_1 - U_{\bar{1}} \cdot Cu_{\bar{1}} \tag{4.1}$$

Using the velocity triangles (illustrated in Figure 4.1), the transformed specific energy E_t is simplified as a function of U, C_2, β and Δ,

$$E_t = U(C_2 - U)\left[1 + (1 - \Delta)\cos\beta\right] \tag{4.2}$$

where $U = U_1 = U_{\bar{1}}$ and,

$$\Delta = 1 - \left|W_{\bar{1}}\right| / \left|W_1\right| \tag{4.3}$$

is the energy loss factor inside the bucket. The transformed power $P_t = \rho Q E_t$ is then given by,

$$P_t = \rho Q U(C_2 - U)\left[1 + (1 - \Delta)\cos\beta\right] \tag{4.4}$$

By neglecting the mechanical loss, the maximum power is generated at the best efficiency point in which,

$$U_{BEP} = \frac{1}{2} C_2 \tag{4.5}$$

By substituting U_{BEP} form (4.5) in (4.4) and neglecting the mechanical loss as well as energy loss inside the bucket, i.e., assuming that $\Delta = 0$, the maximum available power is derived as,

$$P_{BEP} = \rho Q U^2 \left[1 + (1 - \Delta)\cos\beta\right] \tag{4.6}$$

By replacing the rotating velocity U with the runner rotational frequency n ($U = \pi n D_1$), (4.6) is rewritten as,

$$P_{BEP} = \pi^2 \rho Q n_{BEP}^2 D_1^2 \left[1 + \cos\beta\right] \tag{4.7}$$

The corresponding torque $T_{BEP} = P_{BEP} / 2\pi n_{BEP}$ is derived as,

$$T_{BEP} = \frac{\pi}{2} \rho\, Q\, n_{BEP} D_1^2 \left[1 + \cos\beta\right] \tag{4.8}$$

The evolutions of the transferred power and torque for a runner is shown in Figure 4.2.

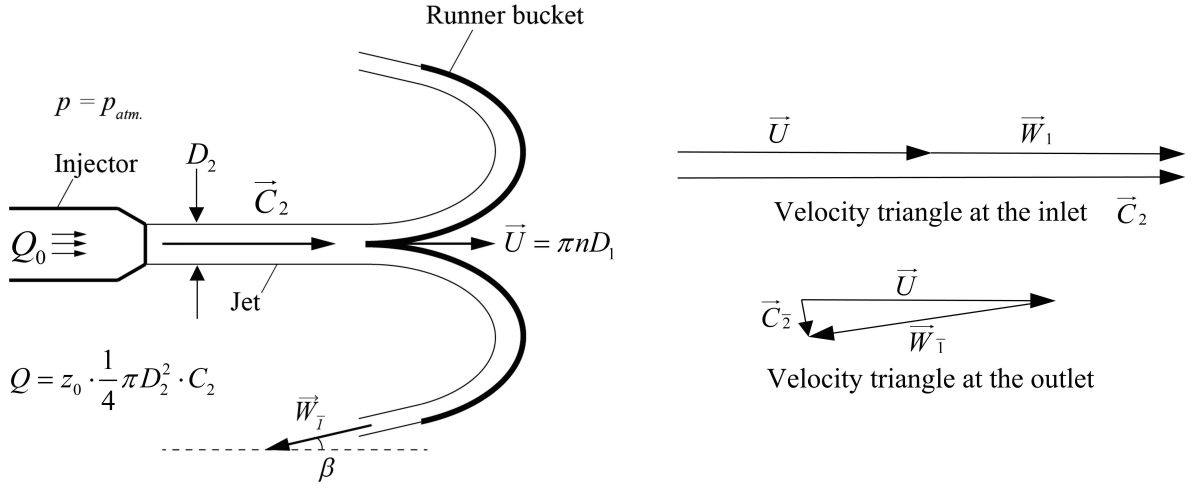


Figure 4.1. A simple 2-D cut of a nozzle impinging a jet on a bucket (left) and velocity triangles at the inlet and outlet of the bucket (right).

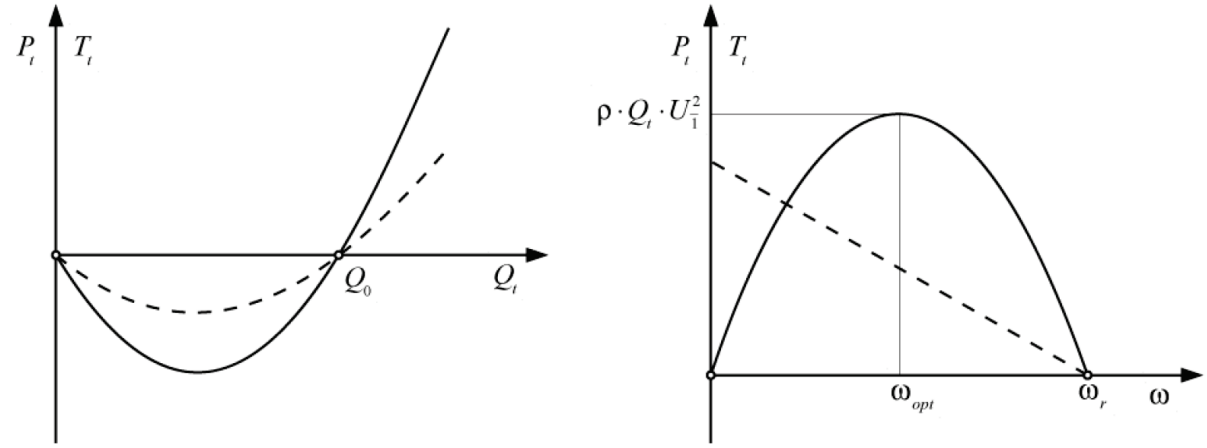


Figure 4.2. Evolutions of transferred power (filled) and torque (dashed) in a turbine runner for a given rotational speed (right) and discharge (left). ω is the rotational speed in rad s^{-1} equal to $2\pi n$. ω_{opt} corresponds to the rotational speed at the best efficiency point.

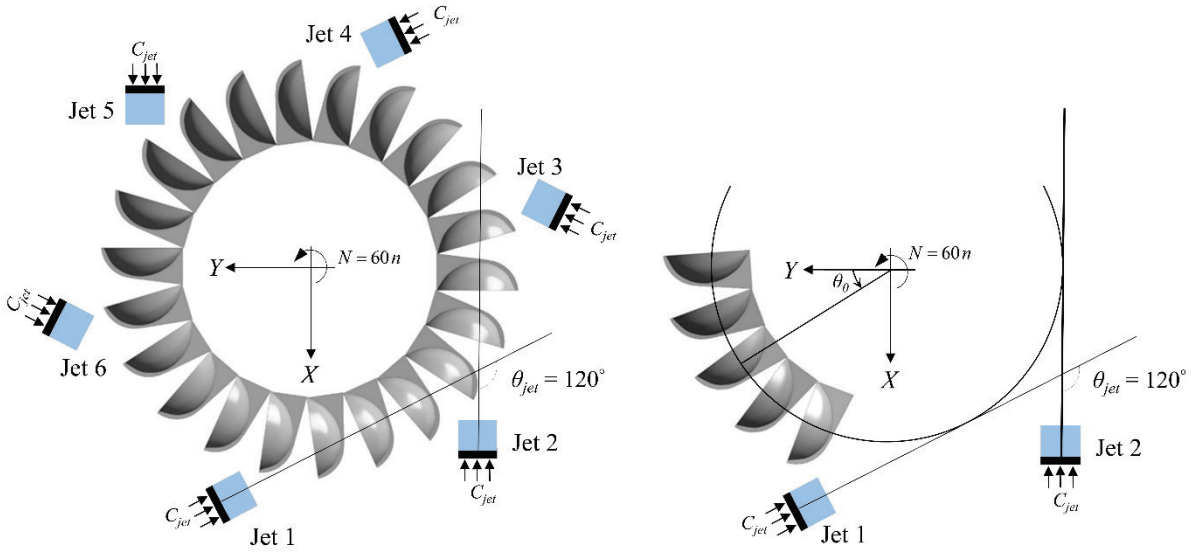


Figure 4.3. Schematic of a six-jet Pelton runner (left) and simplified double-jet numerical simulation setup with only two adjacent jets with 60° between them for jet interference investigation (right). θ is the angle between the middle bucket and the Y-axis. For the simplified setup, similar to the section 3.2.2, only the individual bucket time-shifted torque copies accounting for the bucket periodicity are superimposed and integrated to provide the overall torque time-history.

4.2 Jet interference

4.2.1 Simulation setup

The schematic of a six-jet Pelton runner simulation setup is shown in Figure 4.3. The buckets rotate about the *Z*-axis and θ is the angle between the intermediate bucket and the *Y*-axis. The six-jet simulation setup is simplified by injecting only two adjacent jets with an angle of 60° between them to decrease the computational cost by taking advantage of geometric periodicity.

To investigate the effect of jet interference, the speed factor of the Pelton turbine k_{Cu}, which is defined as,

$$k_{Cu} = \frac{\pi}{60}\frac{ND_1}{\sqrt{2gH}} \tag{4.9}$$

is varied by changing the runner rotational speed N while keeping everything else fixed. This introduces an efficiency change that is a combination of two effects: i) jet interference, and ii) the change of the speed ratio U / C_2, which is equal to ½ at BEP. To isolate the impact of jet interference, the torque obtained from jet 1, not subject to interference, is compared to the torque derived from jet 2, subject to interference with jet 1. Both torque measurements are affected by the changes of the speed ratio values while only the second one experiences the effect of interference. The efficiency drop due to jet interference can then be isolated by comparing the torque derived from each jet.

For the present study, the interaction between the jets is investigated for eight operating points, including $k_{Cu} / k_{Cu}^{BEP} = \{0.89, 0.94, 1.0, 1.05, 1.11, 1.16, 1.22, 1.31\}$ by monitoring the overall and per bucket torque time-history as well as free surface reconstruction. For all the numerical simulations, the rated head H and discharge Q are kept constant, and the runner rotational speed, N, is altered to change the speed factor. Thus,

$$\frac{k_{Cu}}{k_{Cu}^{BEP}} = \frac{N}{N_{BEP}} \tag{4.10}$$

As described in chapter 3, the time-shifted individual bucket torque time-histories are used for the overall torque computation.

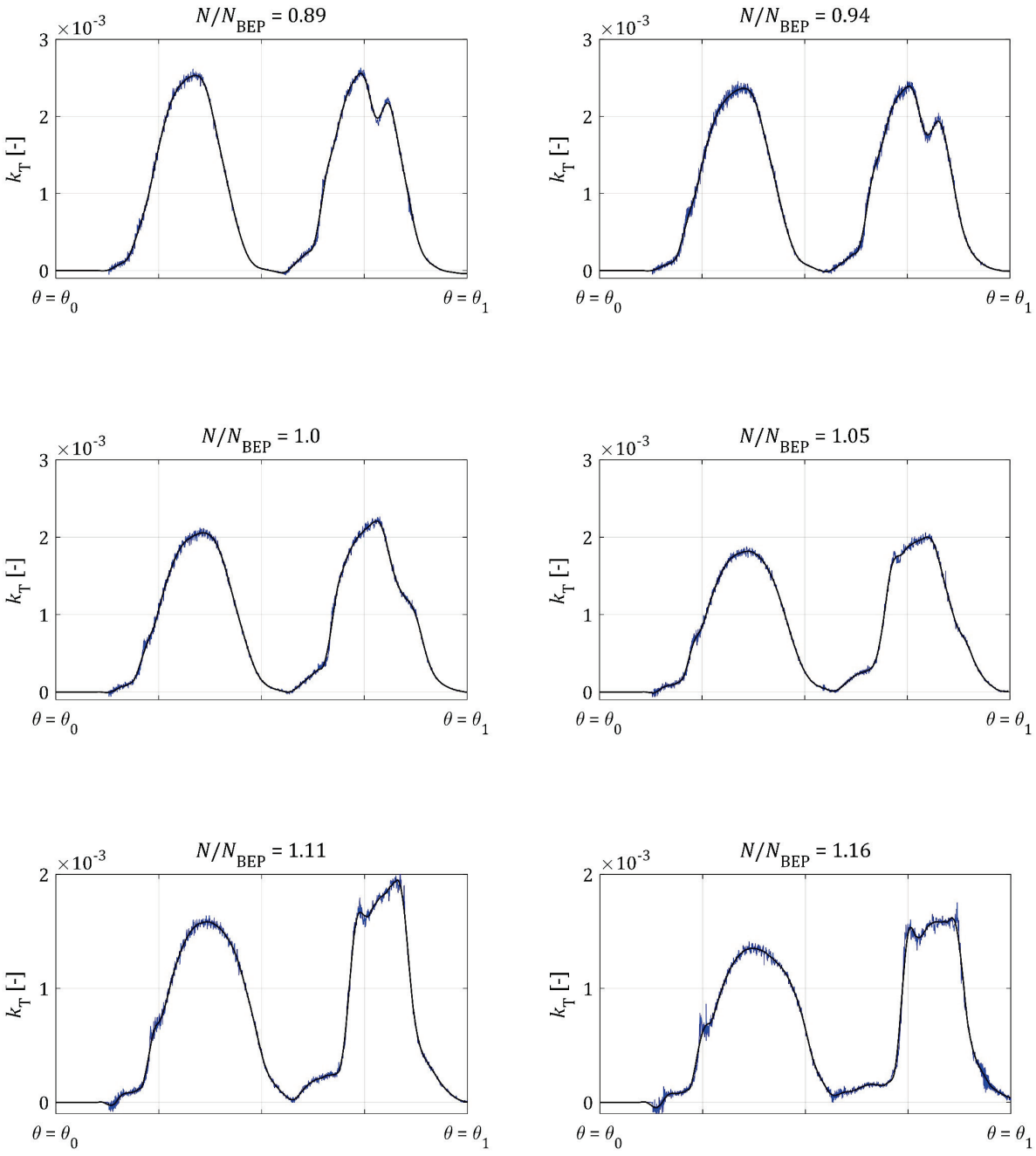
4.2.2 Torque

The torque time-history of the 3rd bucket vs. the bucket rotation θ is shown in Figure 4.4 for all the aforementioned operating points. At $N = N_{BEP}$, both jets generate an independent and almost same torque signal characterized by a smooth transition towards the maximum torque that occurs when the bucket is fully loaded, followed by an equally smooth decline as the water is discharged.

At high rotational speed values, jet 2 enters the bucket before jet 1 has been entirely discharged, affecting the torque induced by jet 2, i.e., jet interference occurs. The pressure

field, forces, torque, and efficiency are consequently affected. The Savitzky-Golay filter has been used to fit a smooth curve from the raw data. In particular, the torque peak of jet 2 becomes much narrower as the speed factor increases.

Jet 2 can also be disturbed by jet 1 which has been formerly deviated, even before entering the bucket. This is called jet disturbance [3] and can even occur at $N < N_{BEP}$. This can explain the drop in the obtained torque from jet 2 around the peak for $N / N_{BEP} = 0.89$, and $N / N_{BEP} = 0.94$. The jet quality and consequently, the induced torque is affected by this jet disturbance.



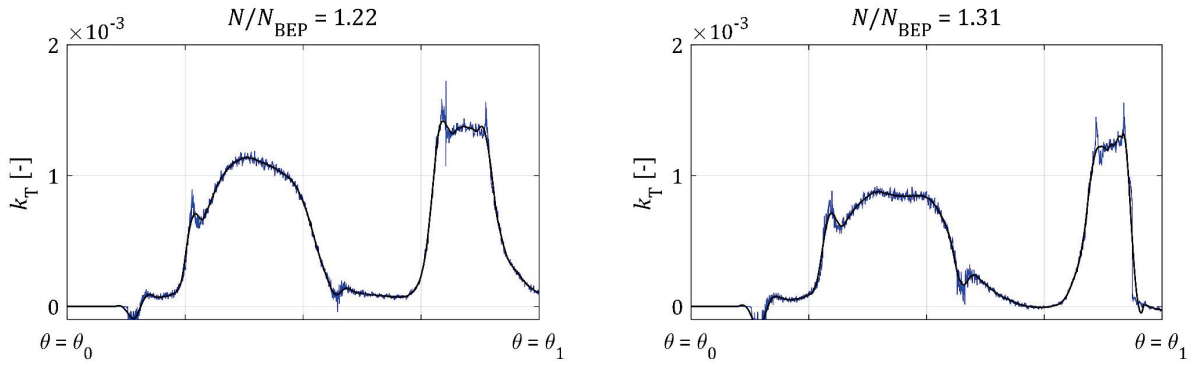
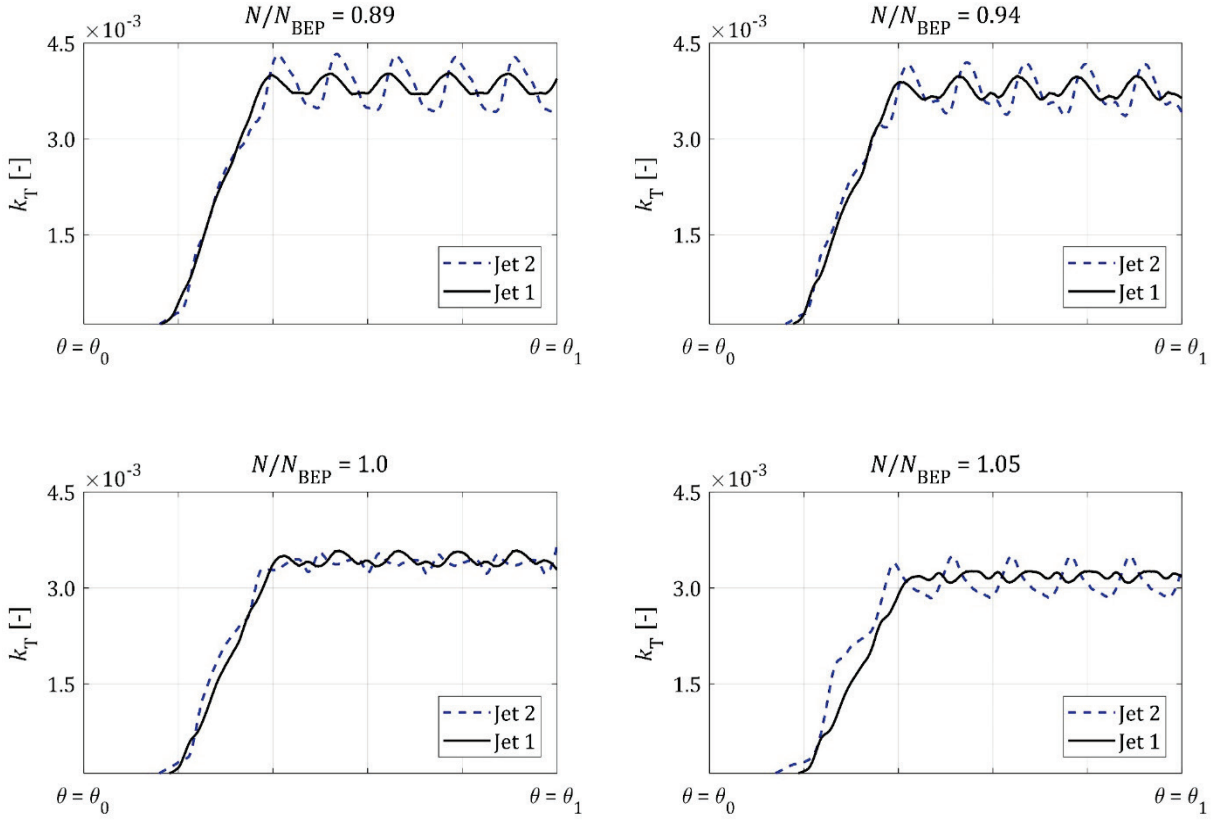


Figure 4.4. The middle bucket torque time-history for eight different operating points $N\ /\ N_{BEP} = \{0.89,0.94,1.0,1.05,1.11,1.16,1.22,1.31\}$. The black curve is fitted from the raw data in blue using the Savitzky-Golay filter. The torque time-history obtained from both jets differ significantly at higher rotational speeds because of jet interference. Jet 2 is disturbed by jet 1 deviated which explains the torque drop (around the peak zone) in the torque time-history from jet 2 at $N\ /\ N_{BEP} = 0.89$ and $N\ /\ N_{BEP} = 0.94$.



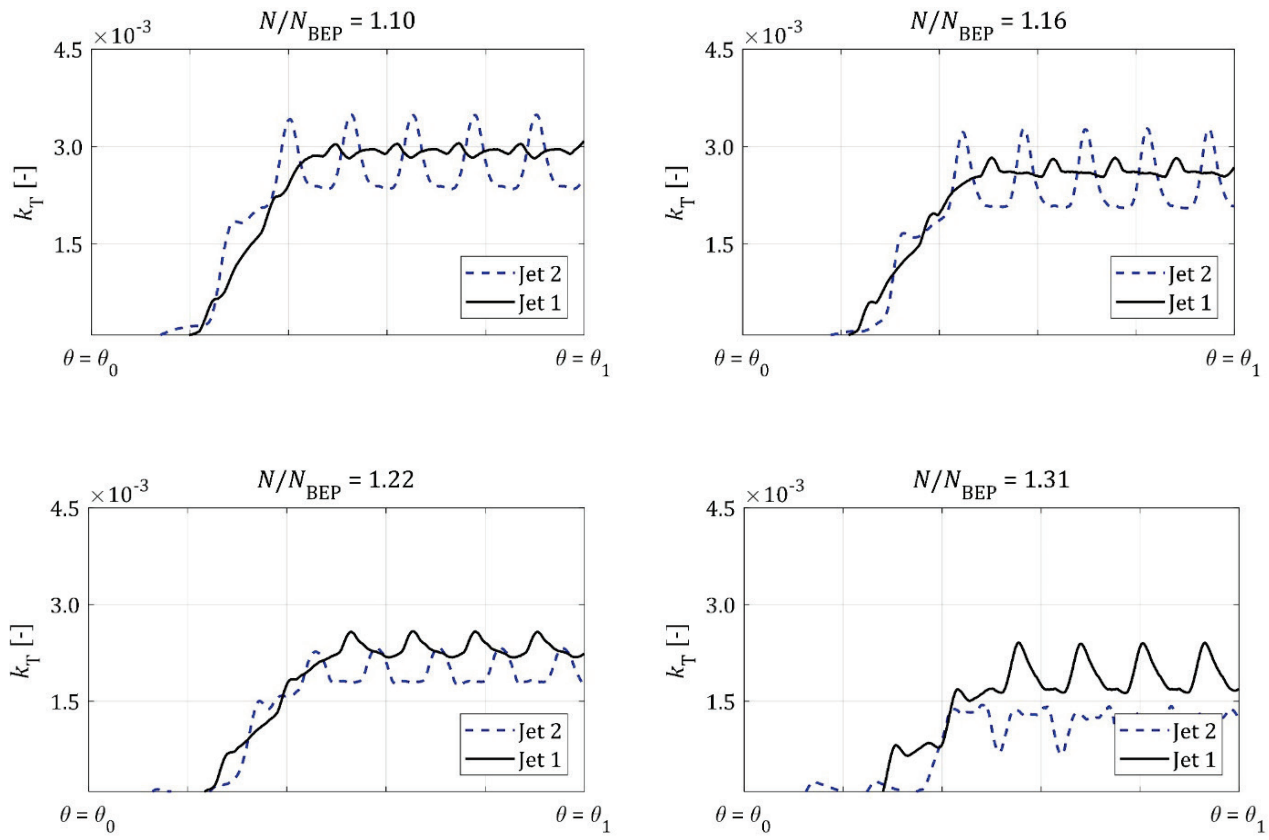


Figure 4.5. The computed overall torque time-histories for dual-jet simulation setup. The solid lines have been calculated based on the torque time-history produced by jet 1 which is always interference-free, whereas the dash lines are based on jet 2 torque time-history which is affected by jet 1.

The overall torque time-history is computed for all the operating points to assess the effect of jet interference on global efficiency. The results are shown in Figure 4.5. At low speed factor values, almost the same overall average torque is generated by both jets, but the interference causes more substantial fluctuations, which might induce increased structural fatigue for the machine. At high rotational speed values, the total average torque derived from jet 2 is significantly lower than that of jet 1 due to the jet interference.

The difference between the time-averaged torque measurements induced by each jet, Δk_T, as well as torque peak-peak fluctuations, $k_T^{fluctuations}$, are shown in Figure 4.6. The difference between the time-averaged overall torques obtained from both jets is computed as,

$$\Delta k_T = \frac{\overline{k}_{T_1} - \overline{k}_{T_2}}{\frac{1}{2}\left(\overline{k}_{T_1} + \overline{k}_{T_2}\right)} \tag{4.11}$$

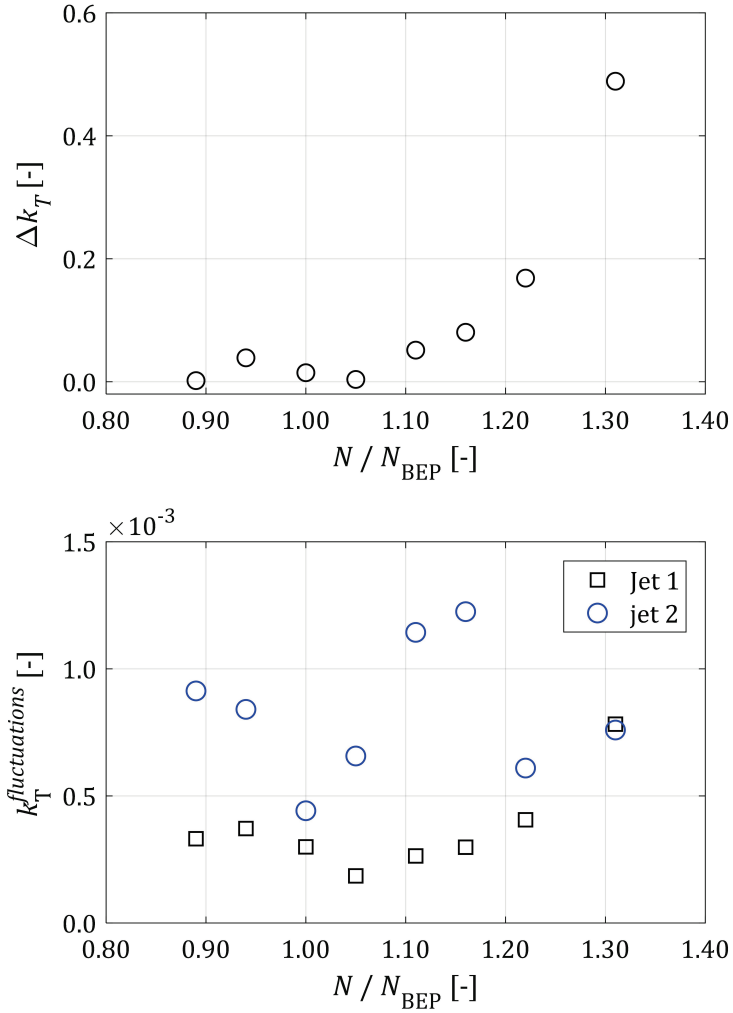


Figure 4.6. The normalized difference between the time-averaged torque produced by each jet (top) and peak-peak fluctuations of each jet torque (bottom). The impact of the jet interference on the torque is intensified by increasing the rotational speeds since Δk_T becomes larger at higher N.

where $\overline{k}_{T_1}$ and $\overline{k}_{T_2}$ are the time-averaged overall resulting torque obtained from jets 1 and 2, respectively. The average torque is strongly affected by jet interference at higher speed factor values, as evidenced by the torque difference between both jets. The difference between the average torque of each jet increases by increasing N / N_{BEP}, wherein the jets are more interference-prone, reaching close to 50% torque less at N / N_{BEP} = 131%.

The torque fluctuation amplitude, which is linked to structural fatigue, is also intensified by jet interference and jet disturbance. The minimum load fluctuations for jet 2 corresponds to the BEP, while the amplitudes can triple at other operating points. For $N < N_{BEP}$, jet 2 is only disturbed (and not interfered) by jet 1 outside the bucket. While jet disturbance has no impact on the average torque, it can significantly increase torque fluctuations.

4.2.3 Efficiency drop

The computed torque and global efficiency obtained from each jet compared to the experimental measurements are shown in Figure 4.7. The global efficiency of a power unit with a rated head H and discharge Q is given by

$$\eta = \frac{\pi}{30}\frac{N \cdot T}{\rho g Q H} \tag{4.12}$$

As indicated in Figure 4.7, there is a sudden efficiency drop at higher rotational speeds where the jets tend to interfere. The torque induced by the primary jet is interference-free while the torque produced by jet 2 can be affected by the formerly injected jet flow. Therefore, the real influence of jet interference on the torque and efficiency can be estimated by comparing the torque and efficiency obtained from each jet as a function of the speed factor.

Even though the computed overall torque is underestimated, as discussed before, the trend is well-captured and is similar to the experimental trend. In particular, the results show that the performance of a Pelton turbine in a broad operating range can be evaluated numerically, but it is necessary that the jet interference is taken into account by the simulations. Although the underestimation can be reduced from 8.5% to almost 7% by refining the particle spacing to $n_p = 50$, the corresponding computational cost is multiplied by almost 8 (since the complexity is of $O(n_p^4)$). For the present study, the spatial discretization resolution is selected based on the available computational resources to perform the simulations in a timely manner with reasonable accuracy. All the computed time-averaged torque factor values obtained by each jet are reported in Table 4.1.

4.2.4 Jet interference visualization

Experimental visualization of the flow in a Pelton turbine is challenging due to its topological complexity and overwhelming water splashing. However, tracking and locating the water free surface is accessible by post-processing of the data provided by numerical solutions. VOF and level set techniques are both diffusive and require dynamic local grid refinement at the air-water interface, whereas particle-based methods are more robust in free surface tracking thanks to their Lagrangian nature. For the present study, the particles' data have been post-processed with an in-house solver developed by the authors to reconstruct and track the free surface. The interaction between the jets at $N / N_{BEP} = 1.0$ and $N / N_{BEP} = 1.31$ for $\theta = \{107, 112, 117, 126\}$ is shown in Figure 4.8 by free surface reconstruction. The reconstructed free surface corroborates the following inconvenient interactions between the adjacent jets:

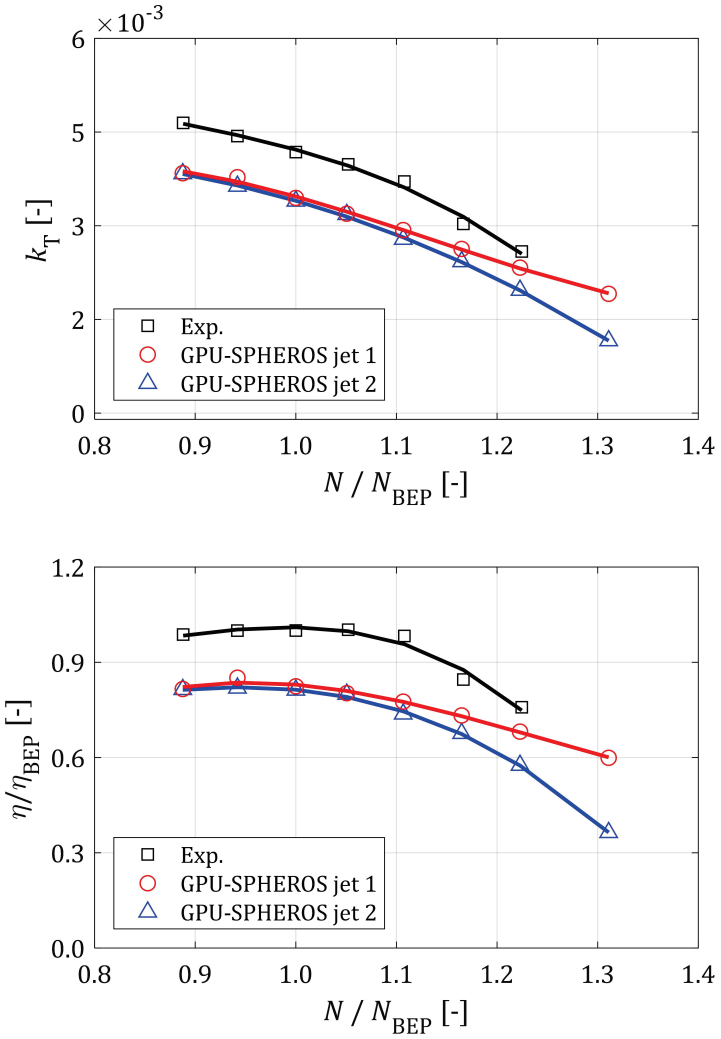


Figure 4.7. The overall time-averaged torque factor k_T (top) and global efficiency η (bottom) obtained from each jet. The time-averaged torque induced by jet 1 is interference-free as there is not any former jet flow while the time-averaged torque produced by jet 2 is affected by the interference with jet 1. Therefore, the interference intensity is assessed by comparing the torque and efficiency between both jets.

- Jet 2 enters a bucket while jet 1 has not fully discharged. This mainly occurs at higher rotational speeds than N_{BEP} inducing an efficiency loss.
- Jet 2 is disturbed by droplets of jet 1 before entering the bucket. This jet disturbance can lead to more substantial load fluctuations.

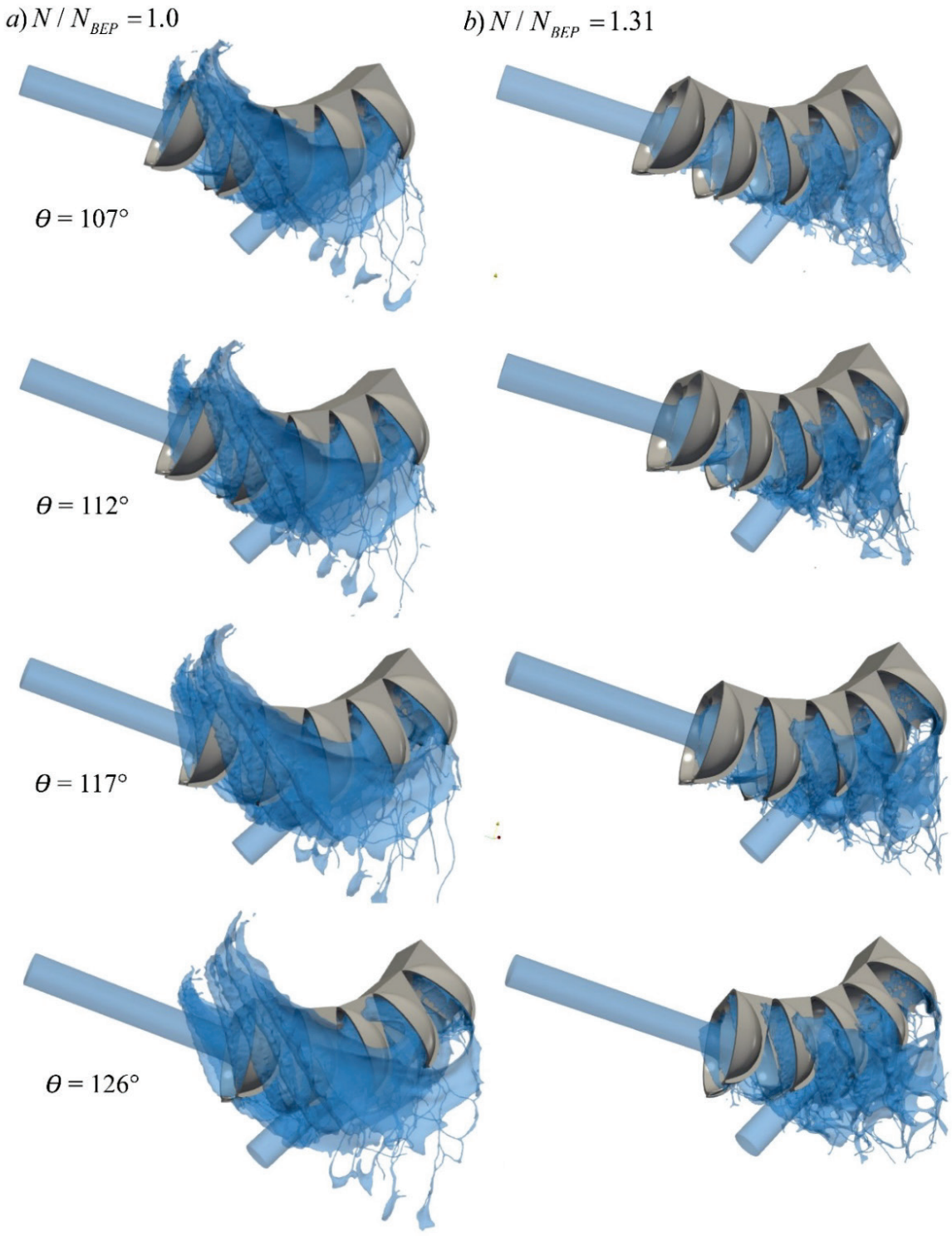


Figure 4.8. The reconstructed free surface for a) $N / N_{BEP} = 1.0$ and, b) $N / N_{BEP} = 1.31$ in four different angular positions of the middle bucket $\theta = \{107, 112, 117, 126\}$. Smaller water sheets have been ejected from the buckets in case (b) when jet 1 is entering the bucket although the per-jet discharge Q_0 is identical in both cases.

Table 4.1. Simulated operating points with the corresponding computed and experimental time-averaged torque factor k_T.

Case#	Operating point $\frac{N}{N_{\text{BEP}}}$ [−]	Exp. time-averaged torque factor (interference considered) $k_T^{Exp.} \times 10^{-3}$ [−]	Numerical time-averaged torque factor induced by jet 2 (interference considered) $k_T^{Numerical} \times 10^{-3}$ [−]
1	0.89	4.64	3.83
2	0.94	4.43	3.67
3	1.00	4.17	3.33
4	1.05	3.98	3.18
5	1.11	3.70	2.78
6	1.16	3.02	2.42
7	1.22	2.58	1.96
8	1.31	Not measured	1.16

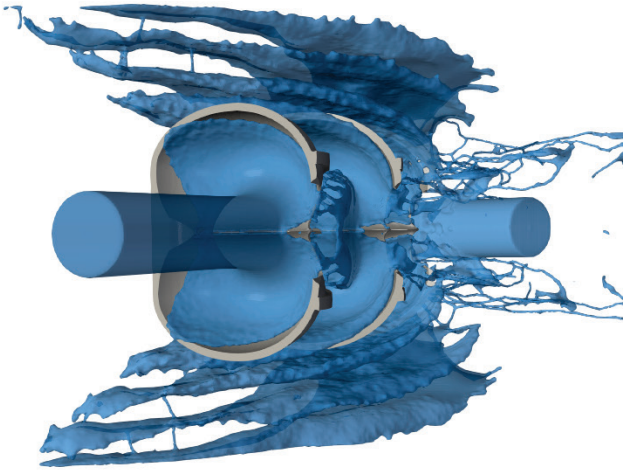


Figure 4.9. The reconstructed water free surface at $N / N_{BEP} = 0.94$ and $\theta = 104°$. Jet 2 is disturbed by the jet 1 cutout flow before entering the bucket.

As shown in the figure, at $N / N_{BEP} = 1.31$, jet 2 is entering a bucket before it is totally discharged. This is evident by comparing the ejected water sheets from buckets in both cases in each row. Smaller water sheets are ejected from the buckets in case (b) when jet 2 is entering even though the per-jet discharge Q_0 is identical in both cases. Moreover, the secondary jet is disturbed by the fraction of jet 1 that is ejected through the bucket cutout, as is shown in Figure 4.9 for $N / N_{BEP} = 0.94$.

Table 4.2. GPU-SPHEROS computing performance for a dual-jet Pelton runner flow simulation setup

Parameter		value
Cluster	-	Piz Daint
CPU	-	Intel® Xeon® E5-2690 v3 @2.60GHz (12 cores)
GPU	-	NVIDIA® Tesla™ P100 16GB
Number of buckets	z_b	5
Number of injected jets	z_0	2
Jet velocity	C_2	25.91 m s^{-1}
Rotational speed	N	534 min^{-1}
Simulation time	t_s	42.5 h
Physical time	t	0.06 s
CFL	-	0.6
Number of used GPUs	N_{GPU}	12
Final number of particles	$n^{final}_{particles}$	1,240,000
Number of particles per jet diameter	n_p	30
Output data saving frequency	S_f	534 × 6 Hz (every one degree)

4.3 Computing Performance

The computational performance for a dual-jet setup Pelton flow simulation at the BEP is given in Table 4.2. The simulation has been performed on Piz Daint with 5704 compute nodes each one equipped with an Intel® Xeon® E5-2690 v3 @2.60GHz (12 cores, 64GB RAM) CPU and a Tesla™ P100 16GB GPU.

4.4 Six-jet full Pelton runner flow visualization

The Pelton turbine torque trend is well-captured by GPU-SPHEROS in a wide operating range (see Figure 4.7), which corroborates the reliability of dual-jet simulation setup reliability used for the present geometry as well as investigated operating range. However, the solver is developed and designed in a manner to handle extensive case-independent realistic free surface simulations. To evaluate this capability, a full six-jet Pelton turbine has been simulated on two Tesla™ P100 GPUs. The simulation is designed for a visual exploration of jet-bucket as well as jet-jet interactions in an off-design condition with N / N_{BEP} = 1.16. As an interference-free case, first, the three non-adjacent jets are injected and deviated by rotating buckets. The domain is then fed by the next three remaining jets, and the jet-jet interactions on the same bucket are visually compared to the interference-free condition. Since the operating point is far from BEP, the interference between the adjacent jets inside the bucket is clearly seen. The reconstructed free surface at different angular positions of buckets is shown in Figure 4.10 and Figure 4.11 in different views. The rotational speed is 622 $min^{-1,}$ and the velocity of the jets is C_2 = 25.91 m·s^{-1}.

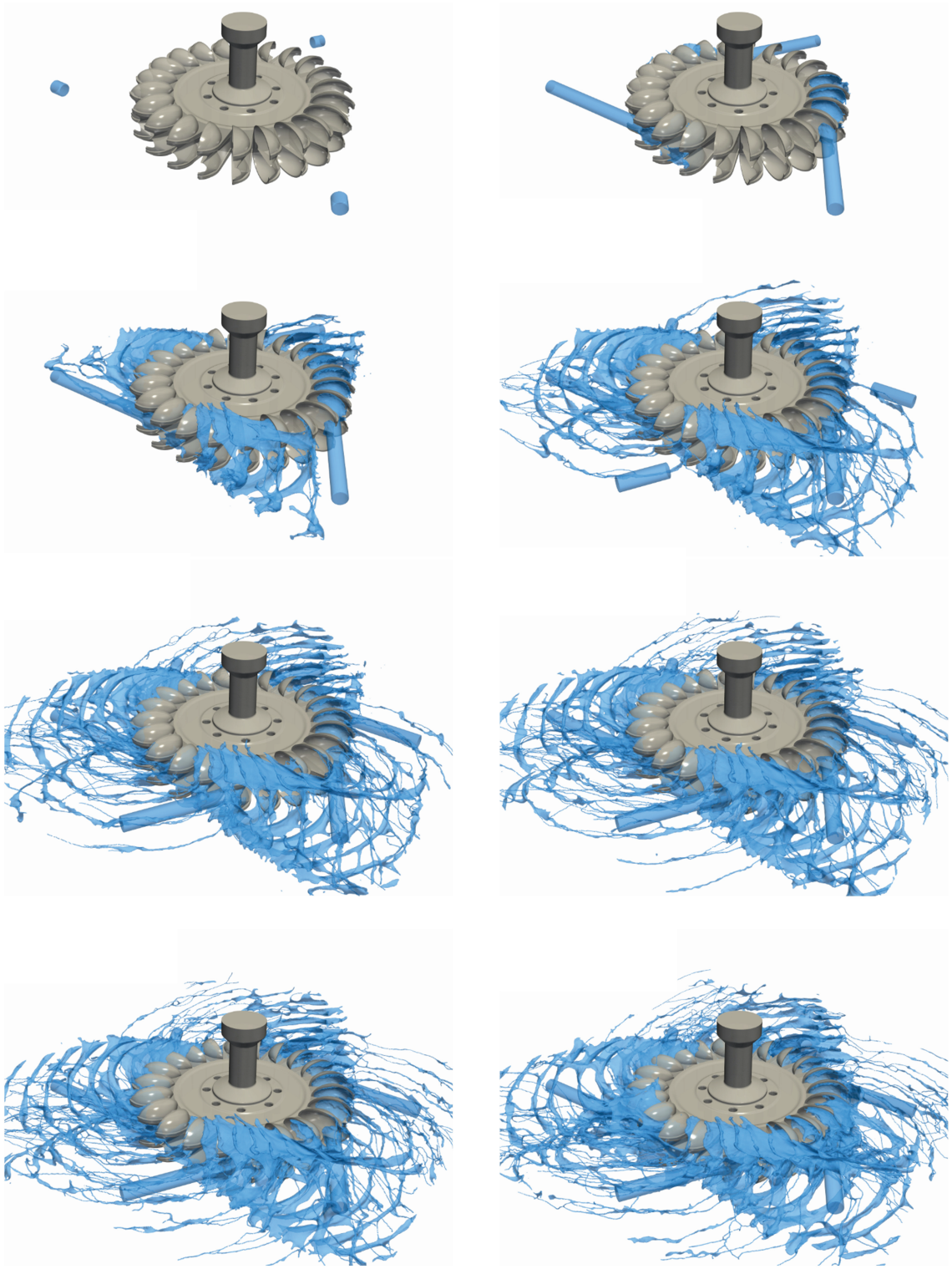
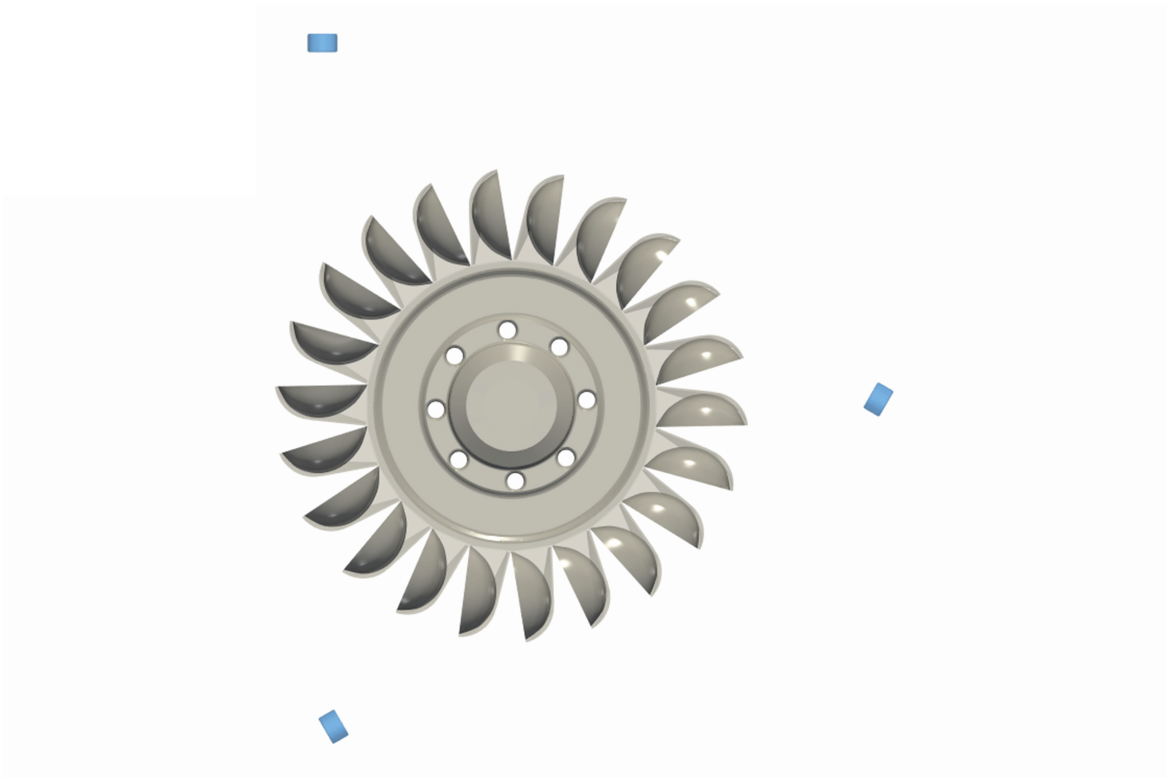
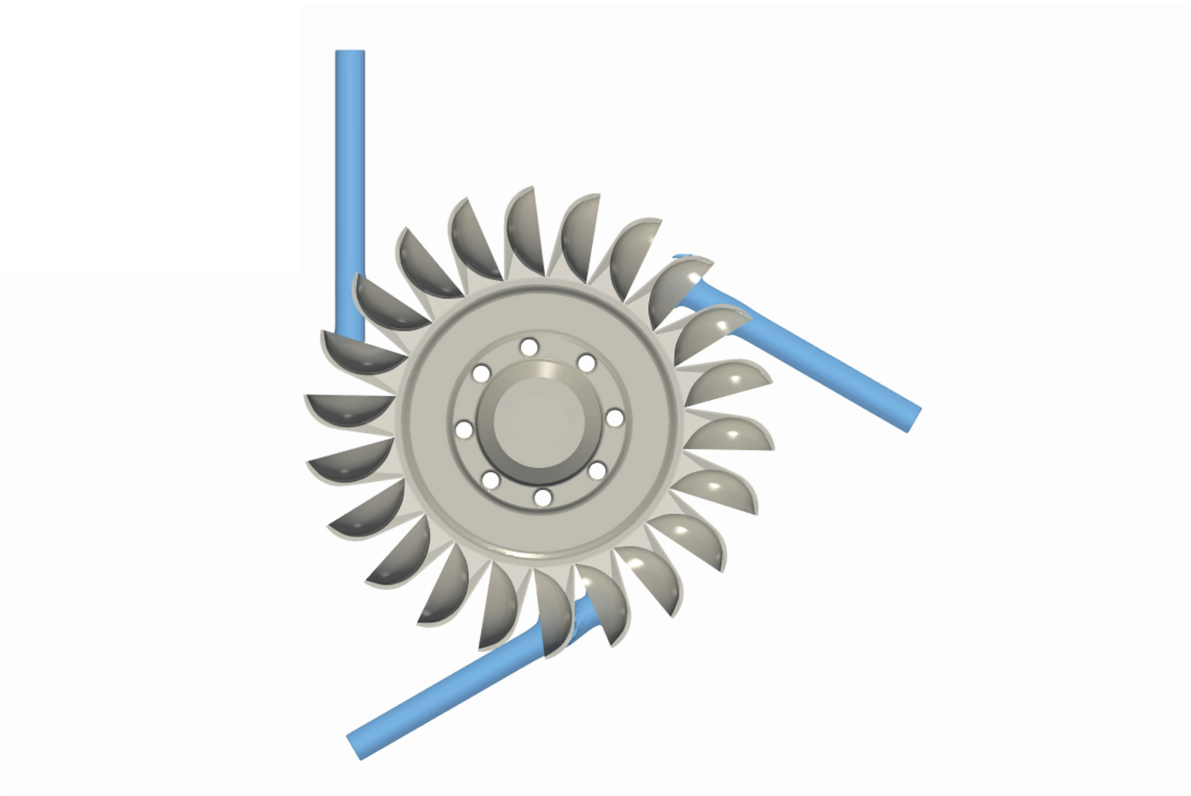
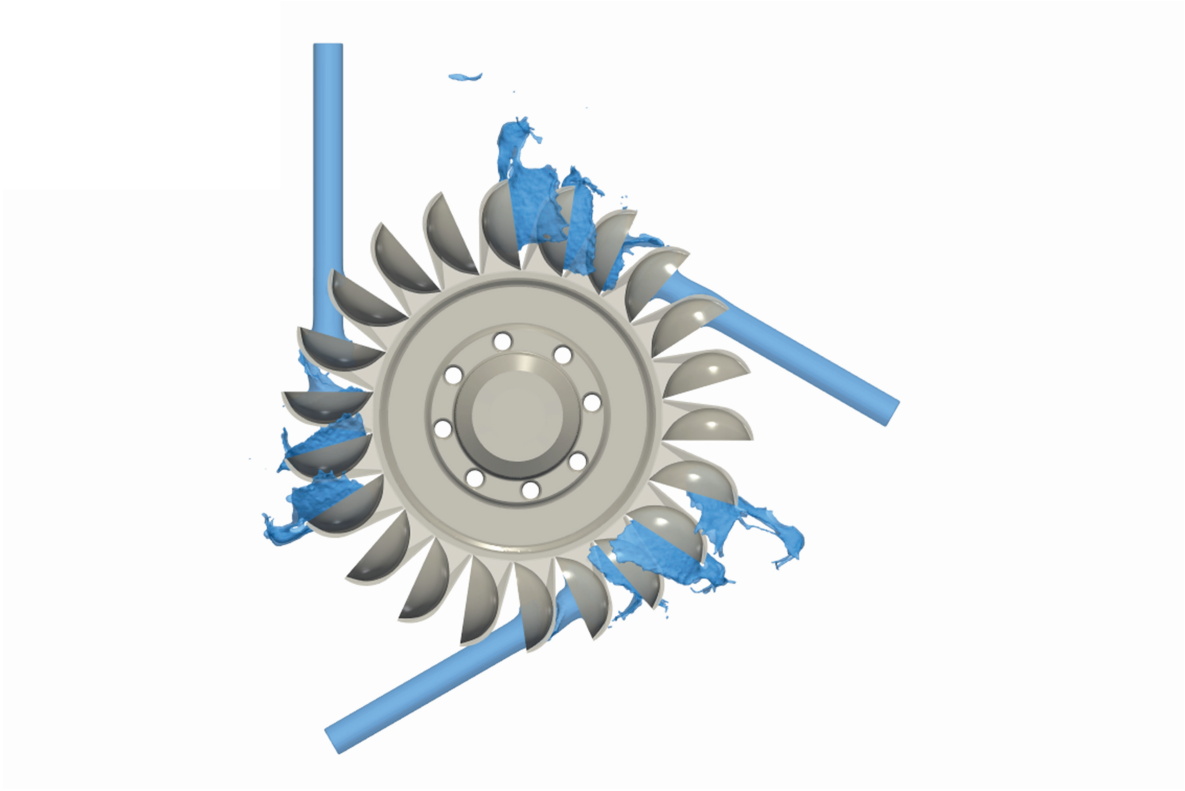
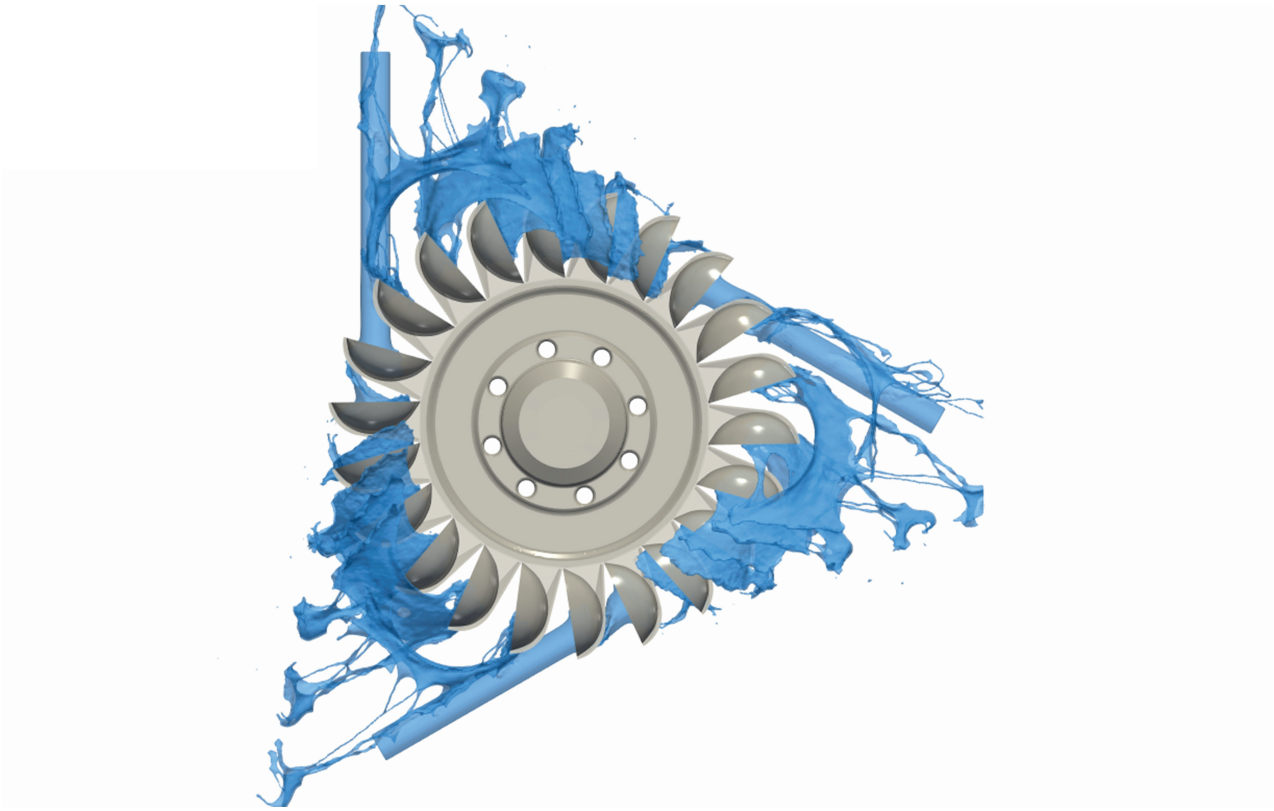


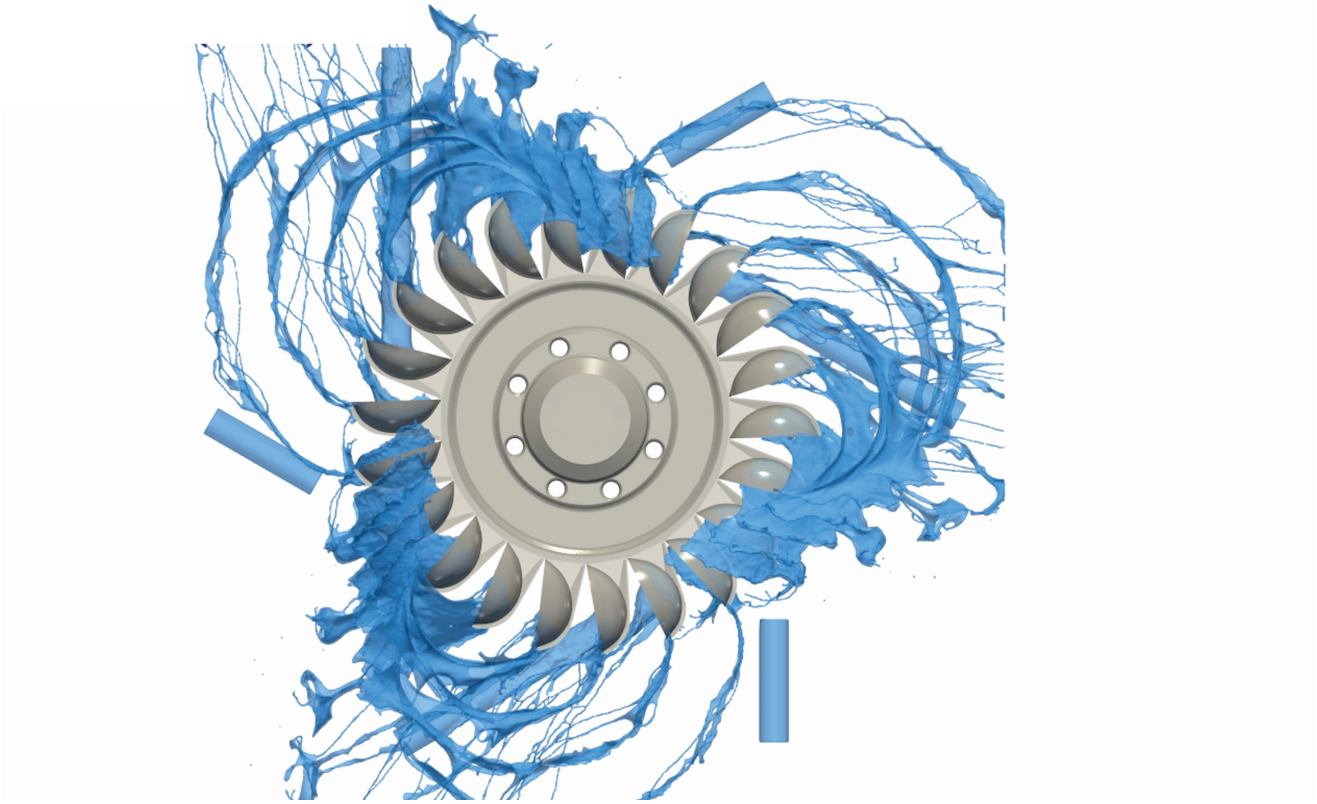
Figure 4.10. The reconstructed free surface of a six-jet Pelton flow in off-design condition, N / N_{BEP} = 1.16. The particle-based methods have no difficulty in tracking a violent free surface.

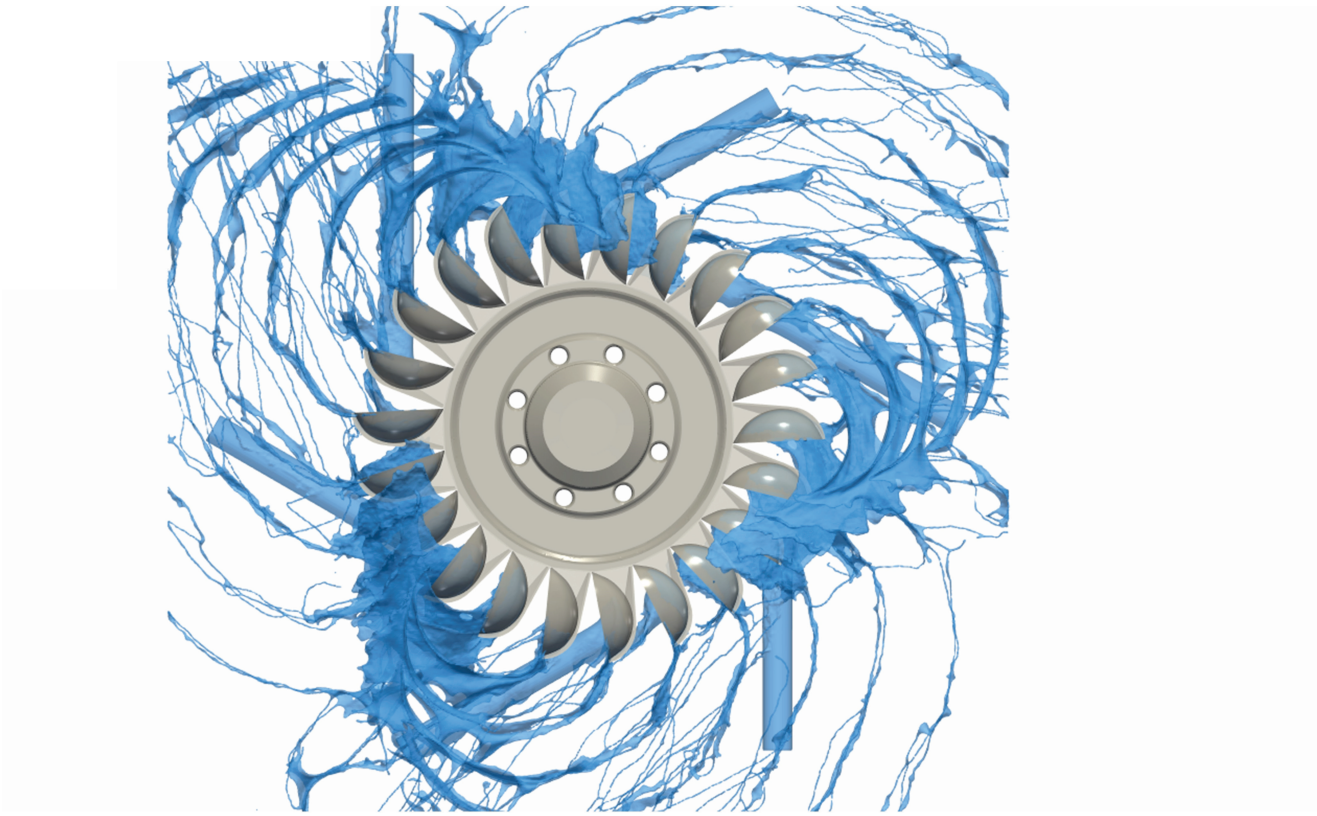












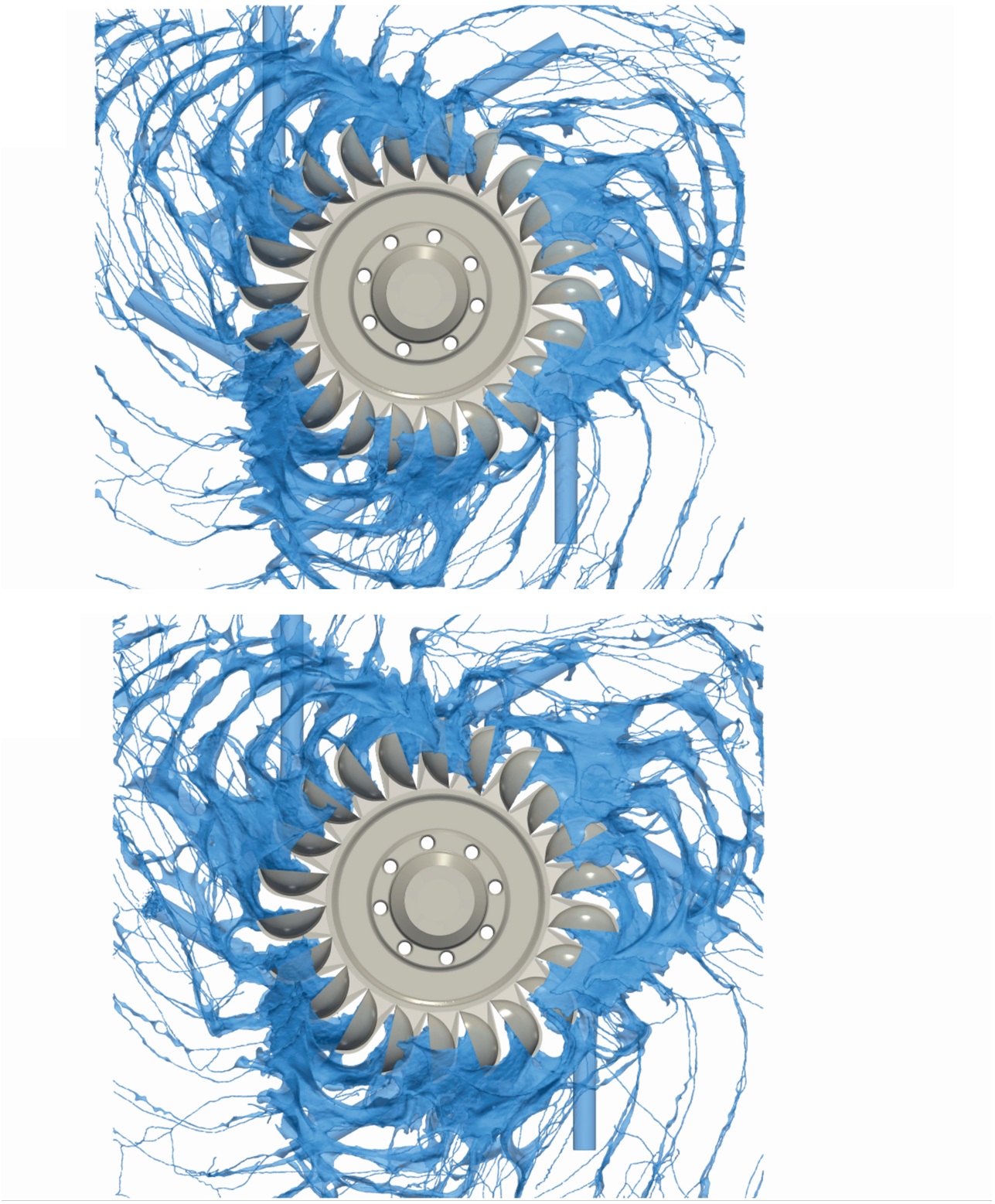


Figure 4.11. The reconstructed free surface of a six-jet Pelton flow at an off-design point $N\ /\ N_{BEP} = 1.16$ at every 50 degrees of bucket angular position.

4.5 Discussion

For a hydropower unit with a given head and discharge, a large number of jets is required to reach higher specific speeds. However, this may raise the risk of jet interference between the jets. In the present study, the interaction between two adjacent impinging jets in a six-jet Pelton runner has been investigated for eight operating points ranging from $N / N_{BEP} = 89\%$ to $N / N_{BEP} = 131\%$. The jet interference inception, which occurs around $N / N_{BEP} = 110\%$, has been accurately estimated for the present case study, in agreement with experimental measurements. Apart from the jet interference that occurs in the same bucket at $N > N_{BEP}$, the numerical results also evidence the occurrence of jet disturbance before entering the bucket. The aforementioned mechanisms create a significant efficiency loss at high rotational speed value; almost 50% at $N / N_{BEP} = 131\%$, as well as amplified load fluctuations at both low and high rotational speed values (doubled at 90%, tripled at 115%), which may accelerate the structural fatigue process in turbine runner. Whereas experimental visualization of Pelton flow is difficult, mostly due to overwhelming water splashing, post-processing of particle-based numerical simulations allows for free surface reconstruction and tracking, which makes it possible to visualize the consequences of jet interference and disturbance, directly and further help the design of the buckets. The code is able to simulate a six-jet Pelton turbine for a quantitative and qualitative investigation of full Pelton hydrodynamics.

5 Conclusion and Outlook

5.1 Conclusion

Industrial flows, such as flow in a Pelton turbine, often involve free surface with moving boundaries wherein conventional mesh-based methods face difficulty to cope with. FVPM is a locally conservative mesh-free numerical simulation method based on the Arbitrary Lagrangian-Eulerian approach that can easily handle such flows, but the traditional implementation is too time-consuming to allow for real-size problem simulations. The method utilizes particle interaction vectors to weigh conservative flux exchange between the neighbor particles and, unlike SPH, is zero-order consistent regardless of variations in particle size.

Within this doctoral research, a GPU-accelerated 3-D FVPM solver, called GPU-SPHEROS, has been developed and used to study the jet interaction in a 6-jet Pelton turbine. The solver has been developed from scratch in CUDA parallel computing platform featuring optimized user-developed CUDA kernels as well as Thrust and CUSP high-performance parallel algorithm libraries. The overall algorithm has three major parts: a) octree based neighbor search comprising tree construction and tree traverse, taking almost 27% of the overall runtime, b) computing particle interaction vectors including spheres intersection, surface partitioning, computing elementary surfaces and computing area vectors and volume, altogether constituting 68% of full runtime and, c) computing mass, momentum, volume and turbulence fluxes and updating variables with a second-order temporal scheme which constitutes only 5% of overall computing time. Widely-used k-ε and k-ω SST RANS methods have been integrated into ALE-based FVPM with moving particles to capture mean flow characteristics.

All the parts have been implemented on GPU, which gives the benefit of minimum data transfer between host and device, even though a fast multi-lane interconnector, NVLink, has been utilized for host-device communications. A roofline model has been used to determine the performance limiters for each part of the code, and appropriate optimization strategies have been applied to both compute and bandwidth-bound kernels. Computing particle interaction vectors is the most time-consuming task, and particular data ordering in memory, memory pre-allocation, and data batching have been performed to improve the performance of this bandwidth-bound part. To improve data locality, the data are sorted using SFC (here, Morton curve) during the neighbor search process. The Thrust parallel radix-sort and gather algorithm has been used for data reordering in memory.

The code has been optimized for NVIDIA® GP100 Pascal architecture. However, to show the code portability, GPU-SPHEROS has been successfully run on NVIDIA® Quadro K2000 GPU with only 37 GFlops of theoretical DP peak performance and 64 GB·s^{-1} of memory bandwidth for a smaller problem. The solver has been validated against experimental data for two major test cases: *i*) impinging jet on a flat plate and, *ii*) torque prediction of a rotating Pelton runner. All the computations have been performed in double precision to mitigate the truncation errors. Altogether, on a single NVIDIA® Tesla™ P100 16GB GPU, the running speed is almost six times faster than a dual-setup CPU node with two Intel® Xeon® E5-2690 v4 Broadwell processors with 28 total physical cores. The hyperthreading is inactivated on CPU.

Once GPU-SPHEROS validated, it was used for numerical simulation of a six-jet Pelton turbine flow as a practical industrial-size free surface problem with the large rotation of boundaries. The jet-bucket and jet-jet interactions have then been studied in a six-jet Pelton runner at eight operating points ranging from $N/N_{BEP} = 89\%$ to $N/N_{BEP} = 131\%$. The torque and efficiency trends, as well as the specific speed range where the jets interfere, have been compared to available experimental data, showing good agreement. All the multi-jet rotating Pelton runner experiments have been performed at Hydraulic R&D laboratory, Hitachi Mitsubishi Hydro Corporation, in which the test rig fulfills the general items and conditions described by the international standard IEC 60193:1999 Hydraulic turbines, storage pumps, and pump-turbines – Model acceptance tests. This validation provides confidence in the use of the present solver for the design optimization of Pelton turbines. This particle-based numerical analysis procedure can be applied to any multi-jet Pelton runner flow.

GPU-SPHEROS is a robust conservative and consistent numerical tool for handling problems with free surface flow and significant boundary motions. All the simulations have been performed on Piz Daint, a GPU-powered supercomputer with 5704 GPU nodes operated by Swiss National Supercomputing Centre – CSCS. The computational time has been reduced from weeks to days, thanks to GPU many-core architecture.

5.2 Outlook

From the author's perspective, the following research items, detailed below, are proposed for future study:

- Further optimizing the code to achieve higher performance;
- Implementing the Reynolds Stress Model (RSM) and Large Eddy Simulations (LES) approach with Sub-Grid Scale (SGS) models to cover a broader range of turbulence applications;
- Implementing cavitation and surface tension models in GPU-SPHEROS for cavitating flows and surface tension-dominated problems;
- Performing numerical simulation of jet interference for a prototype to explore the scale effect on jet interference inception;

- Linking the code with an optimization algorithm for Pelton bucket geometry optimization.

Based on the achieved performance in the roofline model, there is still room for optimization. The preferred candidate is the interaction vectors computations, which on average takes almost 68% of the overall computation time. The next candidate can be the particle neighbor search, which is almost 27% of the overall runtime. Furthermore, GPU-GPU communication performance should be evaluated to find whether the optimization priority is the interaction vectors or multi-GPU communication.

Although two equation RANS models are fast and robust in convergence, they have been developed based on the Boussinesq hypothesis, which is particularly error-prone in predicting fluid flow with highly-anisotropic velocity field, swirling flows or flows with a sudden and abrupt change in average flow strain. This issue can be addressed by using Reynolds Stress Models (RSM), in which all the six individual components of Reynolds stresses $\overline{u'_\alpha u'_\beta}$ are directly computed. Another option would be to use the Large Eddy Simulation (LES) approach. The main idea of LES is to solve only larger scale eddies and ignore the smallest length scales using a low pass filter to reduce the computational costs. Unresolved small scale eddies can be modeled by a Sub-Grid Scale (SGS) Model. Developing and implementing RSM and LES into GPU-SPHEROS will improve the software reliability for a broader range of applications, specifically those with turbulent free surface vortices [93]. Moreover, since the method is ALE-based, it can be used for turbulence-affected internal flow simulations such as Francis and Kaplan runners but with fixed particles and lower computational cost. As a fluid solver, implementing cavitation and surface tension modules are useful for cavitating flow simulations and free surface problems in which the surface tension effect becomes crucial.

Since the reduced scale model of the prototype does not fulfill the Reynolds homology,the jet interference intensity can be affected by the scale [3]. Therefore the numerical simulation of a real-scale multi-jet Pelton turbine is worthwhile in order to assess the scale effects on jet interference inception. Other phenomena such as negative torque generated by jet shock on the backside of the bucket at the beginning of jet-bucket interaction, useful torque produced based on Coanda effect, cutout leakage, optimizing number of buckets, etc. can all be appraised by GPU-SPHEROS.

In the context of improving the design process of Pelton runners, the solver can be used for geometry optimization by linking it to a multi-objective optimization code with a reliable optimization algorithm.

A Weak Formulation of Conservation Law

A weak solution to a PDE is a function in which is not necessarily differentiable but satisfies the PDE [83]. The weak formulation is the approach to solve those PDEs with a not sufficiently smooth solution. Given the hyperbolic conservation law with conserved physical quantity $\mathbf{U}$ and flux function $\mathbf{F}$,

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F} = 0 \tag{A.1}$$

and multiplying (A.1) by a smooth function $\psi = \psi(\mathbf{x}, t)$ with compact support and integrating by part over the whole domain Ω yields

$$\int_t \int_\Omega \psi \frac{\partial \mathbf{U}}{\partial t} d\Omega dt + \int_t \int_\Omega \psi \ \nabla \cdot \mathbf{F} \, d\Omega dt = 0 \tag{A.2}$$

Since φ vanishes outside of the bounded support, i.e., $\psi = 0$ on $\partial\Omega$, (A.1) is rewritten as

$$\int_t \int_\Omega \mathbf{U} \frac{\partial \psi}{\partial t} d\Omega dt = -\int_t \int_\Omega \nabla \cdot \psi \ \mathbf{F} d\Omega dt \tag{A.3}$$

Eq. (A.4) remains meaningful for a discontinues function $\mathbf{U}$ since it does not involve any derivative of $\mathbf{U}$. A locally integrable function $\mathbf{U}$ provides the weak solution of the conservation law if (A.4) holds true for any continuous differentiable test function ψ [83]. In FVPM, the Shepard test function is used for weak formulation (see [20] for more details).

The hyperbolic systems of PDEs are also connected to the conservation law. By integrating (A.1) over the domain Ω,

$$\int_\Omega \frac{\partial \mathbf{U}}{\partial t} d\Omega + \int_\Omega \nabla \cdot \mathbf{F} d\Omega = 0 \tag{A.4}$$

Using the divergence theorem, (A.5) is equivalent to,

$$\frac{\partial}{\partial t} \int_\Omega \mathbf{U} d\Omega \ = \ -\oint_{\partial\Omega} \mathbf{F} \cdot \mathbf{n} ds \tag{A.5}$$

Eq. (A.5) means that the change of $\mathbf{U}$ in time over Ω is equal to the net flux passing through the boundary of the domain $\partial\Omega$, i.e., $\mathbf{U}$ is conserved within Ω.

B RANS-FVPM

Turbulence is a flow regime, which is characterized by chaotic changes in pressure and flow velocity. Generally, unsteady vortices — containing a wide range of scales — appear in turbulent flows. Turbulent flows are challenging to simulate due to unsteady aperiodic motions and random spatial variations in the flow field. Direct Numerical Simulation (DNS) is a method to solve time-dependent Navier-Stokes equations by resolving the full spatial and temporal scales of the turbulence. Although there is no need for any turbulence model in DNS, the computational cost of these methods is prohibitive. However, there are alternative methods to extract mean and large-scale quantities at a reasonable computational time based on turbulence modeling. Reynolds-Averaged Navier–Stokes (RANS) methods and Large Eddy Simulation (LES) are the main alternatives, which can respectively give the time-averaged mean value and the direct resolved large-scale eddies by applying low pass eddy length-scale filters. In the current doctoral research standard and realizable k-ε as well as k-ω SST models have been integrated into ALE-based FVPM to capture mean turbulent flow characteristics. These models are widely used and validated for both internal and free surface flows [62],[63].

B.1. Reynolds-Averaged Navier-Stokes

The idea behind the RANS is Reynolds decomposition, where the instantaneous velocity is decomposed into time-averaged and fluctuating components [85]. The velocity components are decomposed as

$$\mathbf{C}(\mathbf{x},t) = \bar{\mathbf{C}}(\mathbf{x}) + \mathbf{c}'(\mathbf{x},t) \tag{B.1}$$

By substituting the decomposed velocity in the momentum equation, a new term $\tau_{\alpha\beta} = -\rho\overline{c'_\alpha c'_\beta}$ appears, which is called Reynolds stress. The Reynolds stress tensor can be defined by Eq. (B.2) as a function of strain rate tensor, which is known as Boussinesq approximation or Boussinesq hypothesis. This approximation relates the Reynolds stress tensor to the velocity gradients through the eddy viscosity μ_t (which is assumed isotropic), i.e.,

$$-\rho\overline{\mathbf{C}'_\alpha \mathbf{C}'_\beta} = 2\mu_t \mathbf{S}_{\alpha\beta} - \frac{2}{3}\rho k \boldsymbol{\delta}_{\alpha\beta} \qquad (\alpha,\beta = 1,2,3) \tag{B.2}$$

The turbulence kinetic energy k can is assumed as a function of velocity fluctuations

$$k = \frac{1}{2}\sum_{\alpha=1}^{3} c_{\alpha}'^{2} \tag{B.3}$$

where c' is assumed to be isotropic. The Boussinesq approximation is reliable in a broad range of applications. However, in some physics such as predicting flows with sudden and abrupt changes in the strain of the averaged flow and swirling flows [86][87] as well as cases with highly anisotropic, the approximation is error-prone. This approximation allows to model turbulence with eddy viscosity models instead of solving the system of equations for 6 unknown Reynolds stress terms $-\rho\overline{c_{\alpha}' c_{\beta}'}$.

B.2. Realizability

Positivity of normal stresses

Based on the Boussinesq approximation, the normal stresses are written as function of strain rate,

$$\overline{C_{\alpha}'^{2}} = 2k\left(\frac{1}{3} - C_{\mu}\frac{k}{\varepsilon}\mathbf{S}_{\alpha\alpha}\right) \tag{B.4}$$

since the normal stresses $C_{\alpha}'^{2}$ are always non-negative, the right side of the Eq. (B.4) should be always non-negative. However, the following condition can occur for large strain rate which is not physically realizable [88],

$$\frac{k}{\varepsilon}\mathbf{S}_{\alpha\alpha} > \frac{1}{3C_{\mu}} \approx 3.7 \tag{B.5}$$

a non-constant C_{μ} is used to the realizability issue.

Cauchy-Schwarz inequality

The Cauchy-Schwarz inequality should be also satisfied for a turbulence model to be realizable. The following procedure is to obtain a constraint for Boussinesq-based eddy viscosity models. A typical approximation for the Reynolds stresses $\tau_{\alpha\beta}$ (as a second-order symmetric tensor) based on the Boussinesq hypothesis is given by Eq. (B.2). The turbulence stresses can then be written in the following general form [84]

$$\tau_{\alpha\beta} = \zeta\boldsymbol{\delta}_{\alpha\beta} - \frac{1}{\chi}\mathbf{S}_{\alpha\beta} \tag{B.6}$$

with the positive constant scalers ζ and χ while χ satisfies the condition $\chi > 1$. The xy Cauchy-Schwarz condition reads

$$\tau_{xx}\tau_{yy} \geq \tau_{xy}^{2} \tag{B.7}$$

Substituting the inequality turbulence stresses τ_{xx}, τ_{yy} and τ_{xx} by their corresponding expressions in (B.6), the xy Cauchy-Schwarz condition is derived as

$$\zeta^2\chi^2 - \zeta\chi\left(S_{xx} + S_{yy}\right) + S_{xx}S_{yy} - S_{xy}^2 \geq 0 \tag{B.8}$$

In general, this quadratic has two real roots $\chi_{1,2}$, which is calculated based on the quadratic formula

$$\chi_{1,2} = \frac{1}{\zeta}\left[\frac{S_{xx} + S_{yy}}{2} \pm \sqrt{\left(\frac{S_{xx} - S_{yy}}{2}\right)^2 - S_{xy}^2}\right] \tag{B.9}$$

Since the strain rate tensor **S** is traceless, the Eq. (B.9) is rewritten as,

$$\chi_{1,2} = \frac{1}{\zeta}\left[-\frac{S_{zz}}{2} \pm \sqrt{\left(\frac{S_{xx} - S_{yy}}{2}\right)^2 - S_{xy}^2}\right] \tag{B.10}$$

The *xy* Cauchy-Schwarz inequality should be valid for the solutions with $\chi \leq \chi_1$ and $\chi \geq \chi_2$. Furthermore, all the normal stresses $\tau_{\alpha\alpha}$ should satisfy the non-negativity constraint which yields,

$$\chi \geq \frac{1}{\zeta}\left(-\frac{S_{zz}}{2}\right) \tag{B.11}$$

meaning that only the larger root of (B.10) is valid and satisfies the Cauchy-Schwarz inequality. Thus,

$$\chi_{xy} \geq \frac{1}{\zeta}\left[-\frac{S_{zz}}{2} + \sqrt{\left(\frac{S_{xx} - S_{yy}}{2}\right)^2 - S_{xy}^2}\right] \tag{B.12}$$

An analogous procedure is applied to *xz* and *yz* components of χ,

$$\chi_{xz} \geq \frac{1}{\zeta}\left[-\frac{S_{yy}}{2} + \sqrt{\left(\frac{S_{xx} - S_{zz}}{2}\right)^2 - S_{xz}^2}\right] \tag{B.13}$$

$$\chi_{yz} \geq \frac{1}{\zeta}\left[-\frac{S_{xx}}{2} + \sqrt{\left(\frac{S_{yy} - S_{zz}}{2}\right)^2 - S_{yz}^2}\right] \tag{B.14}$$

And the final χ which satisfies all the Cauchy-Schwarz inequalities reads,

$$\chi \geq \max\left[1.0, \chi_{xy}, \chi_{xz}, \chi_{yz}\right] \tag{B.15}$$

where χ_{xy}, χ_{xz}, χ_{yz} are computed based on (B.12–B.14). By comparing (B.2) and (B.6), the relation between the constant χ and eddy viscosity is derived as a function of eddy viscosity,

$$\chi = \frac{\rho}{2\mu_t} \tag{B.16}$$

By substituting (B.16) into (B.15), the new form of the inequality (B.15) is derived as,

$$\mu_t \le \frac{\rho}{2\max\left[1.0,\ \chi_{xy}, \chi_{xz}, \chi_{yz}\right]} \tag{B.17}$$

Furthermore, for the realizability, all the normal stresses must be non-negative, i.e.,

$$\tau_{xx} = \zeta - \frac{1}{\chi} S_{xx} \ge 0 \tag{B.18}$$

$$\tau_{yy} = \zeta - \frac{1}{\chi} S_{yy} \ge 0 \tag{B.19}$$

$$\tau_{zz} = \zeta - \frac{1}{\chi} S_{zz} \ge 0 \tag{B.20}$$

and the final constraint for the eddy viscosity μ_t is derived as (B.21),

$$\mu_t \le \frac{\frac{1}{2}\rho\zeta}{\max\left[\left|S_{xx}\right|, \left|S_{yy}\right|, \left|S_{zz}\right|\right]} \tag{B.21}$$

B.3. k-ω SST blending functions

For the k-ω SST model, the blending functions play a crucial role in success the method. The formulation is based on the nearest wall distance as well as fluid variables [69],

$$F_x = \tanh\left[\arg_x^4\right] \qquad (x = 1, 2) \tag{B.22}$$

where,

$$\arg_1 = \min\left[\max\left(\frac{\sqrt{k}}{\beta'\omega d}, \frac{500\upsilon}{\omega d^2}\right), \frac{4\rho k}{CD_{k\omega}\sigma_{\omega 2} d^2}\right] \tag{B.23}$$

and,

$$\arg_2 = \max\left[\frac{2\sqrt{k}}{\beta'\omega d}, \frac{500\upsilon}{\omega d^2}\right] \tag{B.24}$$

The cross diffusion term is given by

$$CD_{k\omega} = \max\left[2\rho\frac{1}{\sigma_{\omega 2}}\frac{1}{\omega}\nabla k\ \nabla\omega, 10^{-10}\right] \tag{B.25}$$

and the model constants are $\beta' = 0.09$, $\alpha_1 = 0.555$, $\beta_1 = 0.075$, $\alpha_{k1} = \alpha_{\omega 1} = 2$, $\alpha_2 = 0.44$, $\beta_2 = 0.828$, $\alpha_{k2} = 1$ and $\alpha_{\omega 2} = 1.168$.

B.4. Automatic wall treatment

The analytical expression for ω in the viscous sublayer and logarithmic region is given by [89],

$$\omega_{visc} = 6\frac{\mu}{\beta'\rho d^2} \tag{B.26}$$

$$\omega_{\log} = \frac{\left[\left(\frac{\mu}{\rho}\left\|\frac{\partial \mathbf{C}_{\text{par}}}{\partial \mathbf{n}_{wall}}\right\|\right)^2 + \left(a_1 k\right)^2\right]^{\frac{1}{4}}}{a_1 \kappa\, d} \tag{B.27}$$

where $a_1 = 0.31$ and von Kármán constant is $\kappa = 0.41$. The eddy frequency of the wall-adjacent particle is blended between $\omega_{\log}$ and ω_{visc} using the following expression,

$$\omega = \sqrt{\omega_{visc}^2 + \omega_{\log}^2} \tag{B.28}$$

and the shear velocity is computed by blending between the logarithmic region and viscous sublayer,

$$u_\tau = \sqrt[4]{\left(u_\tau^{visc}\right)^4 + \left(u_\tau^{\log}\right)^4} \tag{B.29}$$

with,

$$u_\tau^{visc} = \frac{\mu}{\rho}\left\|\frac{\partial \mathbf{C}_{\text{par}}}{\partial \mathbf{n}_{wall}}\right\| \tag{B.30}$$

and,

$$u_\tau^{\log} = \frac{\left\|\mathbf{C}_{\text{par}}\right\|}{\frac{1}{\kappa}\log\left(Ey^+\right)} \tag{B.31}$$

E is a constant .in logarithmic law equal to 9.793.

C Morton Index Generation

For a given point $P = P(x,y,z)$, the Morton code is generated by bitwise interleaving. For instance, for $P(x,y,z) = (5,9,1)$ in space, the corresponding binary coordinate in base 2 is $(5,9,1)_{10} = (0101,1001,0001)_2$ in which bitwise interleaving the bits results in: $(010001000111)_2 = (1095)_{10}$. The example is illustrated in Figure C.1.

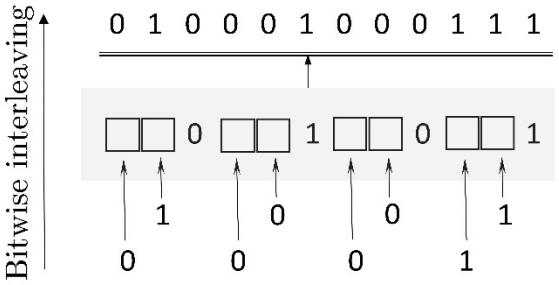


Figure C.1. Morton code generation procedure for $P = P(x,y,z)$ by bitwise interleaving of each coordinate x, y, and z.

The kernel presented in listing C.1 takes three 64-bit unsigned integers and returns the corresponding Morton key. Since the use of look-up tables is inefficient on GPUs, because of the many parallel threads that want to access the same memory [72], the Morton code is generated directly by binary masking and interleaving. First, the floating-point coordinates are converted into 64-bit unsigned integers by shifting the reference frame to the lower left corner of the domain and then multiply the new positions by the size of the domain. Afterward, the coordinates' binary numbers are expanded for bitwise interleaving to generate 1-D corresponding Morton key of each point $P = P(x,y,z)$.

Listing C.1. The kernel, which takes particles coordinate as input and generates the respective Morton keys in parallel as output.

```
__global__ void mCodeCalc(double* x, double* y, double* z,
                                    uint64_t* mCode, int num){

      unsigned int id = blockIdx.x * blockDim.x + threadIdx.x;

      if(id < num){

         //Make sure that we are not going out of bounds
         double globalXMin = 0.0;
         double globalXMax = 1.0;
         double globalYMin = 0.0;
         double globalYMax = 1.0;
         double globalZMin = 0.0;
         double globalZMax = 1.0;

         double xScaled = (x[id] - globalXMin) / (globalXMax - globalXMin);
         double yScaled = (y[id] - globalYMin) / (globalYMax - globalYMin);
         double zScaled = (z[id] - globalZMin) / (globalZMax - globalZMin);

         uint64_t X =  min(max(xScaled * 1048576.0, 0.0), 1048575.0);
         uint64_t Y =  min(max(yScaled * 1048576.0, 0.0), 1048575.0);
         uint64_t Z =  min(max(zScaled * 1048576.0, 0.0), 1048575.0);

         uint64_t v, xx, yy, zz;

         v  =  X;
         v  = (v | (v << 32)) & 0x7fff00000000ffff;
         v  = (v | (v << 16)) & 0x00ff0000ff0000ff;
         v  = (v | (v <<  8)) & 0x700f00f00f00f00f;
         v  = (v | (v <<  4)) & 0x30c30c30c30c30c3;
         xx = (v | (v <<  2)) & 0x1249249249249249;

         v  =  Y;
         v  = (v | (v << 32)) & 0x7fff00000000ffff;
         v  = (v | (v << 16)) & 0x00ff0000ff0000ff;
         v  = (v | (v <<  8)) & 0x700f00f00f00f00f;
         v  = (v | (v <<  4)) & 0x30c30c30c30c30c3;
         yy = (v | (v <<  2)) & 0x1249249249249249;

         v  =  Z;
         v  = (v | (v << 32)) & 0x7fff00000000ffff;
         v  = (v | (v << 16)) & 0x00ff0000ff0000ff;
         v  = (v | (v <<  8)) & 0x700f00f00f00f00f;
         v  = (v | (v <<  4)) & 0x30c30c30c30c30c3;
         zz = (v | (v <<  2)) & 0x1249249249249249;

         mCode[id] = xx * 4 + yy * 2 + zz;
      }
   }
```

Bibliography

[1] Lester A. Pelton, Camptonville CA, US Patent 233,692, 26 Oct. 1880

[2] Patel K, Patel B, Yadav M and Foggia T, 2010 Romania. Development of Pelton turbine using numerical simulation, 25th IAHR Symposium on Hydraulic Machines and Systems

[3] T. Kubota, Observation of jet interference in 6-nozzle Pelton turbine, Journal of Hydraulic Research 27 (1989) 759-768

[4] Zoppe, B., Pellone, C., Maitre, T., and Leroy, P., 2006. "Flow analysis inside a Pelton turbine bucket". J. Turbomach., 128, pp. 500–511.

[5] S. Kvicinsky, Methode d'analyse des Ecoulements 3d a Surface Libre: Application aux Turbines Pelton, École Polytechnique Fédérale de Lausanne, doctoral Thesis N° 2526 (2002).

[6] S. Kvicinsky, J-L Kueny, F. Avellan, E. Parkinson. Experimental and numerical analysis or free surface flows in a rotating bucket. The proceeding of the 21 IAHR symposium on hydraulic machinery and systems 2002.

[7] A. Perrig, F. Avellan, J.L. Kueny, M. Farhat, Flow in a Pelton Turbine Bucket Numerical and Experimental Investigations, Journal of Fluids Engineering128 (2006) 350-358, DOI: 10.1115/1.2170120

[8] A. ŽIDONIS, G. A. AGGIDIS, Pelton turbine: Identifying the optimum number of buckets using CFD, Journal of Hydrodynamics, 2016,28(1):75-83, DOI: 10.1016/S1001-6058(16)60609-1

[9] Santolin, A., Cavazzini, G., Ardizzon, G., and Pavesi, G., 2009. "Numerical investigation of the interaction between jet and bucket in a Pelton turbine". Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy, 223, pp. 721–728.

[10] Jost, D., Meznar, P., and Lipej, A., 2010. “Numerical prediction of a Pelton turbine efficiency”. In Proceedings of the 25th IAHR Symposium on Hydraulic Machinery and Systems, Timisoara.

[11] D. Benzon, A. Židonis, A. Panagiotopoulos, G. A. Aggidis, J. S. Anagnostopoulos, D. E. Papantonis, Impulse Turbine Injector Design Improvement Using Computational Fluid Dynamics J. Fluids Eng 137 (4) 2015, 041106

[12] D. Benzon, A. Židonis, A. Panagiotopoulos, G. A. Aggidis, J. S. Anagnostopoulos, D. E. Papantonis , Numerical Investigation of the Spear Valve Configuration on the Performance of Pelton and Turgo Turbine Injectors and Runners, J. Fluids Eng 137(11), 111201 (Nov 01, 2015) (8 pages)

[13] A. Rossetti, G. Pavesi, G. Ardizzon, A. Santolin, Numerical Analyses of Cavitating Flow in a Pelton Turbine, J. Fluids Eng 136(8), 081304 (2014); doi: 10.1115/1.4027139

[14] Koukouvinis PK, Anagnostopoulos JS, Papantonis DE. Flow analysis inside a Pelton turbine bucket using smoothed particle hydrodynamics HYDRO inter-national conerence 2010.

[15] Darwish, M.; Moukalled, F. (2006). "Convective Schemes for Capturing Interfaces of Free-Surface Flows on Unstructured Grids". Numerical Heat Transfer Part B. 49: 19–42.

[16] J-C Marongiu, F. Leboeuf, J. Caro & E. Parkinson (2010) Free surface flows simulations in Pelton turbines using a hybrid SPH-ALE method, Journal of Hydraulic Research, 48:S1, 40-49, DOI: 10.1080/00221686.2010.9641244

[17] Anagnostopoulos JS, Papantonis DE. A numerical methodology for design optimization of Pelton turbine runners. . Hydro; 2006. p. 25–7.

[18] C. Vessaz, E. Jahanbakhsh, F. Avellan, Flow Simulation of Jet Deviation by Rotating Pelton Buckets Using Finite Volume Particle Method, Journal of Fluids Engineering 137 (2015) 7, 074501 1-7

[19] M. Junk, “Do finite volume methods need a mesh?”, in: M. Griebel, M. Schweitzer (Eds.), Meshfree Methods for Partial Differential Equations, in: Lecture Notes in Computational Science and Engineering 26, Springer, Berlin Heidelberg (2003) 223–238. http://dx.doi.org/10.1007/978-3-642-56103-0 15.

[20] E. Jahanbakhsh, C. Vessaz, A. Maertens, F. Avellan, “Development of a Finite Volume Particle Method for 3-D fluid flow”, Computer Methods in Applied Mechanics and Engineering 298 (2016) 80-107.

[21] A. Hérault, G. Bilotta, R. A. Dalrymple (2010), SPH on GPU with CUDA, Journal of Hydraulic Research, 48:sup1, 74-79, doi: 10.1080/00221686.2010.9641247.

[22] Y. Xia, F. Han, J. Zhou, T Kubota, numerical prediction of dynamic performance of pelton turbine, Journal of Hydrodynamics, Ser.B, 2007,19(3):356-364.

[23] C. Vessaz, E. Jahanbakhsh, F. Avellan, FPM simulations of a high-speed water jet validation with CFD and experimental results, in: P. Gourbesville, J. Cunge, G. Caignaert (Eds.), Advances in Hydroinformatics, in: Springer Hydrogeology, Springer, Singapore, 2014, pp. 419–431. http://dx.doi.org/10.1007/978-981-4451-42-0 34.

[24] A. Kosior, H. Kudela, "Parallel computations on GPU in 3D using the vortex particle method", Computers & Fluids (80) 423-428, 2013, https://doi.org/10.1016/j.compfluid.2012.01.014

[25] B. Nayroles, G. Touzot, and P. Villon. -Generalizing the finite element method: Diffuse approximation and diffuse elements. Computational Mechanics, 10(5):307–318, 1992.

[26] Price, Daniel J (2009). "Astrophysical Smooth Particle Hydrodynamics". New Astron.rev. 53 (4–6): 78–104. arXiv:0903.5075. doi:10.1016/j.newar.2009.08.007.

[27] Libersky, L.D.; Petschek, A.G. (1990). Smooth Particle Hydrodynamics with Strength of Materials, Advances in the Free Lagrange Method. Lecture Notes in Physics. 395. pp. 248–257. doi:10.1007/3-540-54960-9_58. ISBN 978-3-540-54960-4.

[28] L.D. Libersky; A.G. Petschek; A.G. Carney; T.C. Hipp; J.R. Allahdadi; F.A. High (1993). "Strain Lagrangian hydrodynamics: a three-dimensional SPH code for dynamic material response". J. Comput. Phys. 109 (1): 67–75. Bibcode:1993JCoPh.109...67L. doi:10.1006/jcph.1993.1199.

[29] J. Bédorf, E. Gaburov, S. P. Zwart, "A sparse octree gravitational N-body code that runs entirely on the GPU processor", Journal of Computational Physics, Vol. 231, N° 7, pp. 2825-2839, 2012.

[30] Jahanbakhsh E, Vessaz C, Maertens A, Avellan F, 2016, Development of a Finite Volume Particle Method for 3-D fluid flow Computer Methods in Applied Mechanics and Engineering, 298, pp. 80-107.

[31] D. Hietel, K. Steiner, J. Struckmeier, A finite-volume particle method for compressible flows, Math. Models Methods Appl. Sci. 10 (9) (2000) 1363–1382. http://dx.doi.org/10.1142/S0218202500000604.

[32] R.M. Nestor, M. Basa, M. Lastiwka, N.J. Quinlan, Extension of the finite volume particle method to viscous flow, J. Comput. Phys. 228 (5)(2009) 1733–1749. http://dx.doi.org/10.1016/j.jcp.2008.11.003.

[33] N.J. Quinlan, R.M. Nestor, Fast exact evaluation of particle interaction vectors in the finite volume particle method, in: Meshfree Methods for Partial Differential Equations V, Springer, Berlin Heidelberg, 2011, pp. 219–234. http://dx.doi.org/10.1007/978-3-642-16229-9 14.

[34] E. Jahanbakhsh, F. Avellan, Simulation of Silt Erosion Using Particle-Based Methods, Doctoral Thesis N° 6284 (2014), EPFL, DOI: 10.5075/epfl-thesis-6284

[35] E. Jahanbakhsh, A. Maertens, N. J. Quinlan, C. Vessaz, F. Avellan, "Exact finite volume particle method with spherical-support kernels", Computer Methods in Applied Mechanics and Engineering 2017 (317) 102–127.

[36] C. Vessaz, "Finite Particle Flow Simulation of Free Jet Deviation by Rotating Pelton Buckets", École Polytechnique Fédérale de Lausanne (EPFL), doctoral thesis N° 6470 (2015).

[37] C. Vessaz, E. Jahanbakhsh and F. Avellan; "Flow Simulation of Jet Deviation by Rotating Pelton Buckets Using Finite Volume Particle Method", Transactions of the ASME, Journal of Fluids Engineering, Vol. 137 (7), doi:10.1115/1.4029839, 2015.

[38] C. Vessaz, L. Andolfatto, F. Avellan, C. Tournier, "Structural and Multidisciplinary Optimization" (ISSN:1615-1488), (2017) 55: 37. doi:10.1007/s00158-016-1465-7

[39] Cheng J, Grossman M, McKercher T. Professional CUDA ® c programming. John Wiley & Sons Inc; 2014.

[40] Vasily Volkov, "Understanding Latency Hiding on GPUs", University of California at Berkeley, doctoral thesis (2016).

[41] CUDA C Programming Guide, PG-02829-001_v9.1, NVIDIA | March 2018

[42] M. G. Venkata, P. Shamis, N. Imam, M. G. Lopez, "OpenSHMEM and Related Technologies. Experiences, Implementations, and Technologies, Second Workshop", OpenSHMEM 2015 Annapolis, MD, USA, August 4–6, 2015 Revised Selected Papers, Lecture Notes in Computer Science, LNCS Sublibrary: SL2 – Programming and Software Engineering, DOI 10.1007/978-3-319-26428-8

[43] D. lee, I. Dinov, B. Dong, B. Gutman, I. Yanovsky, A. W. Toga, "CUDA optimization strategies for compute- and memory-bound neuroimaging algorithms", Computer Methods and Programs in Biomedicine 106 (2012) 75-187.

[44] NVIDIA Tesla P100 whitepaper (NVIDIA 2016), WP-08019-001_v01.1 | 1

[45] NVIDIA® GameWorks™ Documentation Rev. 1.0.181212 ©2014-2018. NVIDIA Corporation.

[46] D. Peng, W. Rick, L. Piotr, T. Stanimire, P. Gregory, D. Jack, "From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming", Parallel Computing 38 (8) pp. 391-407, 2012. doi:10.1016/j.parco.2011.10.002.

[47] J. Yang, Y. Wang, Y. Chen, "GPU accelerated molecular dynamics simulation of thermal conductivities", J. Comput. Phys., Vol. 221, pp. 799–804, 2007

[48] J. Tolke, "Implement of a lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA", Comput. Vis. Sci., Vol. 13, pp. 29–39, 2010

[49] M. J. Goldsworthy, "A GPU–CUDA based direct simulation Monte Carlo algorithm for real gas flows", Computers & Fluids, Vol. 94, pp. 58-68, 2014, https://doi.org/10.1016/j.compfluid.2014.01.033

[50] S. Tomov, M. McGuigan, R. Bennett, G. Smith, J. Spiletic, "Benchmarking and implementation of probability-based simulations on programmable graphics card", Comput. Graph. 29, pp. 71–80, 2005

[51] M. Papadrakakis, G. Stavroulakis, A. Karatarakis, "A new era in scientific computing: domain decomposition methods in hybrid CPU–GPU architectures", Computer Methods in Applied Mechanics and Engineering, Vol. 200, pp. 1490–1508, 2011

[52] G. Stavroulakis, D. G. Giovanis, V. Papadopoulos, M. Papadrakakis, "A GPU domain decomposition solution for spectral stochastic finite element method", Computer Methods in Applied Mechanics and Engineering (2017), https://doi.org/10.1016/j.cma.2017.08.042

[53] A. Karatarakis, P. Karakitsios, M. Papadrakakis, "GPU accelerated computation of the isogeometric analysis stiffness matrix", Computer Methods in Applied Mechanics and Engineering, Vol. 269, pp. 334–355, 2014

[54] A. Hérault, A. Vicari, C. del Negro, and R.A. Dalrymple, "Modeling Water Waves in the Surf Zone with GPU-SPHysics", in Proceeding of the 4th International SPHERIC Workshop, Nantes, France, 2009.

[55] A.J.C. Crespo, J.M. Domínguez, B.D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. García-Feal, DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH), Computer Physics Communications 187 (2015) 204-216, ISSN 0010-4655, https://doi.org/10.1016/j.cpc.2014.10.004.

[56] A. E. Nocentino, P. J. Rhodes, "Optimizing memory access on GPUs using Morton order indexing", in Proceedings of the 48th Annual Southeast Regional Conference, ACM, New York, USA, 2010.

[57] D. Valdez-Balderas, J. M. Dominguez and B. D. Rogers, "Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters", Journal of Parallel and Distributed Computing, vol. 73, no. 11, pp. 1483-1493, 2013.

[58] J.L. Cercos-Pita, "AQUAgpusph, A new free 3D SPH solver accelerated with OpenCL", Computer Physics Communications 192 (2015) 295–312.

[59] Batchelor GK. Introduction to fluid dynamics. Cambridge University Press, U.K; 1974

[60] J. Monaghan, "Simulating free surface flows with SPH", J. Comput. Phys. 110 (2) (1994) 399–406. http://dx.doi.org/10.1006/jcph.1994.1034.

[61] R. Fatehi, M.T. Manzari, A consistent and fast weakly compressible smoothed particle hydrodynamics with a new wall boundary condition, Internat. J. Numer. Methods Fluids 68 (7) (2012) 905–921. http://dx.doi.org/10.1002/fld.2586.

[62] Pope S, 2000, Turbulent Flows, Cambridge University Press

[63] Chen C J, Jaw S, 1998, Fundamentals of turbulence modeling, Taylor & Francis, ISBN 1-56032

[64] ANSYS Fluent 12.0 Theory Guide 2009, ANSYS Inc.

[65] Frank M, White, 2003 Fluid Mechanics, 5th edition, McGraw-Hill

[66] Versteeg H K, Malalasekera W, 2007 England An introduction to computational fluid dynamics: The finite volume method, Pearson Education Ltd.

[67] Shih T H, Liou W W, Shabbir A, Yang Z, Zhu J, 1995 A New k-ε Eddy-Viscosity Model for High Reynolds Number Turbulent Flows - Model Development and Validation, Computers Fluids, 24 (3):227-238.

[68] Reynolds W C 1987 Fundamentals of turbulence for turbulence modeling and simulation, Lecture Notes for Von Karman Institute Agard Report No. 755.

[69] F.R. Menter, M. Kuntz, R.B. Langtry, Ten Years of Industrial Experience with the SST Turbulence Model, turbulance heat and Mass Transfer 4 (2003) 625-632

[70] N. A. Wukie, P D. Orkwis, A p-Poisson wall distance approach for turbulence modeling, 23rd AIAA Computational Fluid Dynamics Conference (2017), Denver, Colorado 10.2514/6.2017-3945.

[71] Witepaper, NVIDIA Tesla P100, WP-08019-001_v01.1 | 1

[72] J. Bédorf, E. Gaburov, S. P. Zwart, "A sparse octree gravitational N-body code that runs entirely on the GPU processor", Journal of Computational Physics, Vol. 231, N° 7, pp. 2825-2839, 2012.

[73] R. Amorim, M. Liebmann, G. Haase, R.W. dos Santos, Comparing CUDA and OpenGL implementations for a Jacobi iteration, HPCS'09 p.22-32, DOI:10.1109/HPCSIM.2009.5192847

[74] D. Winkler, M. Meister, M. Rezavand, W. Rauch, gpuSPHASE—A shared memory caching implementation for 2D SPH using CUDA, Computer Physics Communications 213 (2017) 165–180.

[75] M.G. Malheiros, M. Walter, Spatial sorting: An efficient strategy for approximate nearest neighbor searching, Computers & Graphics 57 (2016) 112–126.

[76] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, M. Püschel, "Applying the roofline model". IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS): 76–85, 2014

[77] Y.J. Lo, S. Williams, B.V. Straalen, T.J. Ligocki, M.J. Cordery, N.J. Wright, M.W. Hall, L. Oliker, "Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis", International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (2014), DOI: 10.1007/978-3-319-17248-4_7

[78] https://docs.nvidia.com/cuda/profiler-users-guide/index.html#metrics-reference-6x (accessed on 06 Feb. 2019)

[79] https://thrust.github.io/doc/group___gathering.html (last access on Feb 28th, 2019)

[80] U. Ghia, K. Ghia, C. Shin, High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, J. Comput. Phys. 48 (3) (1982) 387–411. http://dx.doi.org/10.1016/0021-9991(82)90058-4.

[81] K.L. Wong, A.J. Baker, A 3D incompressible Navier–Stokes velocity–vorticity weak form finite element algorithm, Internat. J. Numer. Methods Fluids 38 (2) (2002) 99–123. http://dx.doi.org/10.1002/fld.204.

[82] T Kumashiro, S Alimirzazadeh, A Maertens, E Jahanbakhsh, S Leguizamón, F Avellan, K Tani, Numerical investigation of the jet velocity profile and its influence on the Pelton turbine performance, IOP Conf. Ser.: Earth Environ. 2019 Sci. 240 072006

[83] A. Bressan, Hyperbolic Systems of Conservation Laws, REVISTA MATEMÁTICA COMPLUTENSE Volumen 12, número 1: 1999, 135-200

[84] M. Ulitsky, A General Realizability Method for the Reynolds Stress for 2-Equation RANS Models (2009), Lawrence Livermore National Laboratory (LLNL-TR-414246).

[85] Adrian R J, Christensen K T, Liu Z C, 2000 Analysis and Interpretation of instantaneous turbulent velocity fields, Experiments in Fluids. 29 pp. 275–290.

[86] Wilcox D C, 1988 Multiscale Model for Turbulent Flows, AIAA Journal, 26 (11) pp. 1311-1320.

[87] Kazuhiko S 1998 Recent development in Eddy Viscosity Modeling of Turbulence, R&D Review of Toyota CRDL 33 (1),

[88] Speziale C G 1991 Analytical methods for the development of Reynolds-stress closures in turbulence, Ann. Rev. Fluid Mechanics 23 pp. 107-157.

[89] ANSYS Solver Theory Guide, Release 12.1, 2009

[90] Wei, T. and W. W. Willmarth (1989). Reynolds-number effects on the structure of a turbulent channel flow. J. Fluid Mech. 204, 57-95.

[91] M. Kato and B. E. Launder, The Modeling of Turbulent Flow Around Stationary and Vibrating Square Cylinders, Proc. 9th Symposium on Turbulent Shear Flows, Kyoto, 1993, pp. 10.4.1-10.4.6.

[92] Cimbala Y, Çengel A, John M, 2006. Fluid mechanics: fundamentals and applications (1st Ed.). McGraw-Hill Higher Education. pp. 321-329. ISBN 0072472367.

[93] S Mulligan, G D. Cesare, J Casserly, R Sherlock, Understanding turbulent free-surface vortex flows using a Taylor-Couette flow analogy, December 2018, Scientific Reports 8(1),DOI: 10.1038/s41598-017-16950-w

[94] Ismail B. Celik, Urmila Ghia, Patrick J. Roache, Christopher J. Freitas, Hugh Coleman, Peter E. Raad, Procedure for Estimation and Reporting of Uncertainty Due to Discretization in CFD Applications, J. Fluids Eng. 130(7), 078001 (Jul 22, 2008)

[95] S. Alimirzazadeh, E. Jahanbakhsh, A. Maertens, S. Leguizamón, F. Avellan, GPU-accelerated 3-D Finite Volume Particle Method, Computers and Fluids 171 (2018) 79–93

[96] S. Alimirzazadeh, T. Kumashiro, S. Leguizamón, E. Jahanbakhsh, A. Maertens, C. Vessaz, K. Tani, F. Avellan, GPU-Accelerated Numerical Analysis of Jet Interference in a Six-Jet Pelton Turbine using Finite Volume Particle Method, Submitted to Journal of Renewable Energy (Current status: under review)

Siamak ALIMIRZAZADEH

Tel: +41 78 6406071
Email: siamak.alimirzazadeh@epfl.ch
https://people.epfl.ch/siamak.alimirzazadeh

Education

- Ph.D. in Energy, École Polytechnique Fédérale de Lausanne (EPFL) 2015-2019
  Thesis title: GPU-Accelerated 3-D Finite Volume Particle Method Applied to Pelton Turbine Flow
  Supervisor: Prof. F. Avellan
  My research was mainly focused on fluid mechancis. A Particle-based fluid solver has been developed and accelerated on GPU from scratch using novel parallel algorithms. The solver has been validated against various experimental test cases results and will be released as an open source solver very soon.
  (My doctoral study is in progress, and I am supposed to defend my dissertation on July 16, 2019)
- M.Sc. in Marine Eng. and Hydrodynamics, Sharif University of Technology 2010-2013
  Thesis title: Design and Fabrication of an Optimized Wake Equalizing Duct (WED) for Traditional Vessels (based on both CFD with ANSYS CFX commercial solver as well as experimental measurement)
  Supervisor: Prof. M. S. Seif
- B.Sc. in Marine Eng. and Hydrodynamics, AmirKabir University of Technology 2005-2010

Research interests

- Computational Fluid Dynamics – CFD (7 years of experience in developing/using both mesh-based and particle-based methods)
- Experimental Fluid Mechanics (Experienced in working with wind and water tunnel)
- High-performance in-house CFD sovler development (implementation + validation vs. the experiments)
- Turbulence Modeling
- Multi-Phase Flow

Publications

Journal papers

- S. Leguizamón, E. Jahanbakhsh, S. Alimirzazadeh, A. Maertens, F. Avellan, FVPM numerical simulation of the effect of particle shape and elasticity on impact erosion, Wear volumes 430-431 (2019), pp. 108-119
- S. Alimirzazadeh, E. Jahanbakhsh, A. Maertens, S. Leguizamón, F. Avellan, GPU-accelerated 3-D Finite Volume Particle Method, Computers and Fluids 171 (2018) 79–93
- S. Alimirzazadeh, T. Kumashiro, S. Leguizamón, E. Jahanbakhsh, A. Maertens, C. Vessaz, K. Tani, F. Avellan, GPU-Accelerated Numerical Analysis of Jet Interference in a Six-Jet Pelton Turbine using Finite Volume Particle Method, Submitted to Journal of Renewable Energy (Currently under review)
- S. Leguizamón, E. Jahanbakhsh, S. Alimirzazadeh, A. Maertens, F. Avellan, FVPM Numerical Simulation of the Effect of Particle Shape and Elasticity on Impact Erosion, Wear 430-431 (2019) 108-119
- S. Leguizamón, E. Jahanbakhsh, A. Maertens, S. Alimirzazadeh, F. Avellan, A multiscale model for sediment impact erosion simulation using the finite volume particle method, Wear Vol. 392-393 (2017) 202-212

- S. Alimirzazadeh, S. Z. Roshan, M. S. Seif, Experimental study on cavitation behavior of propellers in the uniform flow and in the wake field, J Braz. Soc. Mech. Sci. Eng. (2016) 38: 1585
  This research has been performed in the K23 water tunnel.
- A. Najafi, S. Alimirzazadeh, M. S. Seif, RANS simulation of interceptor effect on hydrodynamic coefficients of longitudinal equations of motion of planing catamarans, J Braz. Soc. Mech. Sci. Eng. (2015) 37: 1257.
- S. Z. Roshan, S. Alimirzazadeh, M. Rad, RANS simulations of the stepped duct effect on the performance of ducted wind turbine, Journal of Wind Engineering and Industrial Aerodynamics 145 (2015) 270-279
  This research has been performed in a wind tunnel, and CFD has been utilized for further investigation.
- S. Alimirzazadeh, S. Z. Roshan, M. S. Seif, Unsteady RANS simulation of a surface piercing propeller in oblique flow, Applied Ocean Research 56 (2016) 79-91

Conference papers

- S. Leguizamón, E. Jahanbakhsh, A. Maertens, S. Alimirzazadeh, F. Avellan, Multiscale Simulation of the Hydroabrasive Erosion of a Pelton Bucket: Bridging Scales to Improve Accuracy, 13th European Turbomachinery Conference ETC, 08-12 April 2019, Lausanne, Switzerland (**With Best Paper Award**)
- S. Alimirzazadeh, T. Kumashiro, S. Leguizamón, E. Jahanbakhsh, A. Maertens , K. Tani, F. Avellan, RANS simulation of a Pelton turbine using the finite volume particle method accelerated on GPU, 13th International SPHERIC Workshop (2018) at Galway Ireland
- S. Alimirzazadeh, T. Kumashiro, S. Leguizamón, A. Maertens, E. Jahanbakhsh, K. Tani, F. Avellan, GPU-Accelerated Pelton Turbine Simulation Using Finite Volume Particle Method Coupled with Linear Eddy Viscosity Models, 29th IAHR Symposium on Hydraulic Machinery and Systems (IAHR 2018), Kyoto, Japan
- T. Kumashiro, S. Alimirzazadeh, A. Maertens, E. Jahanbakhsh, S. Leguizamón, F. Avellan, K. Tani, Numerical investigation of the non-uniform water jet and its influence in a Pelton turbine bucket, 29th IAHR Symposium on Hydraulic Machinery and Systems (IAHR 2018), Kyoto, Japan
- S. Alimirzazadeh, E. Jahanbakhsh, A. Maertens, S. Leguizamón, F. Avellan, A GPU-accelerated versatile solver based on the finite volume particle method, 12th international SPHERIC Workshop (2017), Ourense, Spain
- S. Leguizamón, E. Jahanbakhsh, A. Maertens, S. Alimirzazadeh, F. Avellan, A multiscale model for the simulation of sediment impact erosion of metallic targets using the finite volume particle method, 12th international SPHERIC Workshop (2017), Ourense, Spain
- S. Leguizamon, E. Jahanbakhsh, A. Maertens, C. Vessaz, S. Alimirzazadeh, F. Avellan, Impact erosion prediction using the finite volume particle method with improved constitutive models, IOP Conference Series: Earth and Environmental Science, Volume 49, Hydro-Abrasive Erosion (2016)

Research Projects and Teaching Experience

- Developing a GPU-Accelerated 3-D Finite Volume Particle CFD Solver 2015-2018
  This project has been performed within the Swiss Competence Center on Energy Research Supply of Electricity, GPU-SPHEROS project Grant No. 17568.1 PFEN IW, with the support of the Swiss Commission for Technology and Innovation (CTI). I have developed the code from scratch for my PhD, and I have managed the project to meet the deadline.
- Integrating RANS turbulence models into 3-D Finite Volume Particle Solver 2017-2018
  Standard and realizable k-ε, as well as SST, have been implemented in GPU-SPHEROS within the framework of a collaborative research with R&D laboratory Hitachi Mitsubishi Hydro Corporation (Hitachi - Japan).
- A researcher at Marine Engineering Research Center MERC, Sharif University of Technology
  Working on both numerical simulations and experimental analysis of marine propellers 2013-2015
- Superintendent of K23 water tunnel of Sharif University of Technology 2013-2014
  Performing experimental measurements on fully-submerged cavitating marine propellers.
- RANS simulation of KRISO Container Ship (KCS) propeller hydrodynamic performance Fall 2013
  The KCS propeller (SVA CP 1193) hydrodynamic performance was simulated in ANSYS CFX, and the effects of different turbulence modelswith different y^+ have evaluated (Performed at Marine Engineering Research Centre, Sharif University of Technology).
- Computer programming for Particle Image Velocimetry (PIV) and Laser Induced Fluorescence (LIF). Spring 2012
  PIV and LIF test have been performed for an injecting jet, and heat transfer in a water cube, respectively. A C++ code

was developed based on the cross-correlation algorithm for PIV and image Processing for LIF to analyze the results. (Under the supervision of Prof. M. B. Shafii and Dr. M. Zabetian, Sharif University of Technology).

Computer skills

- Programming skills: C, C++, CUDA, MATLAB
- Software skills: ICEM CFD, ANSYS CFX, ANSYS DesingModeler, LibreCad, Paraview
- General skills: Linux, Windows, vim, OpenSSH, SLURM, Microsoft Office

Language

- English Skills (Fluent in both speaking and writing)
- Persian (Native)