5-2014

Particle Modeling of Fuel Plate Melting during Coolant Flow Blockage in HFIR

Hiraku Nakamura
University of Tennessee - Knoxville, hiraku@utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Hiraku Nakamura entitled "Particle Modeling of Fuel Plate Melting during Coolant Flow Blockage in HFIR." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Nuclear Engineering.

Arthur E. Ruggles, Major Professor

We have read this dissertation and recommend its acceptance:

David H. Cook, Lawrence W. Townsend, Michael W. Guidry

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Particle Modeling of Fuel Plate Melting during Coolant Flow Blockage in HFIR

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Hiraku Nakamura
May 2014

Dedication

Dedicated in memory of my grandmother, aunt, and uncle:

Eiko Toyama, Ayako Toyama, and Mitsuru Toyama

Acknowledgements

This dissertation was made possible through the support of many. I would like to express my profound gratitude here.

I would like to thank my advisor, Dr. Ruggles for his patient guidance and all his hard work.

I would like to thank my supervisor, Dr. Cook for always taking his time to answer my questions and his constant willingness to help.

I would like to thank Dr. Townsend for his generous guidance and interest.

I would like to thank Dr. Guidry for his encouragements and helpful suggestions.

I would also like to thank my friends and my family for their continuous support and encouragement.

I am grateful for the financial support of the GTRI to the LEU Conversion Program for the HFIR.

Abstract

Cooling channel inlet flow blockage has damaged fuel in plate fueled reactors and contributes significantly to the probability of fuel damage based on Probabilistic Risk Assessment. A Smoothed Particle Hydrodynamics (SPH) model for fuel melt from inlet flow blockage for the High Flux Isotope Reactor is created. The model is coded for high throughput graphics processing unit (GPU) calculations. This modeling approach allows movement toward quantification of the uncertainty in fuel coolant flow blockage consequence assessment. The SPH modeling approach is convenient for following movement of fuel and coolant during melt progression and provides a tool for capturing the interactions of fuel melting into the coolant. The development of this new model is presented. The implementation of the model for GPU simulation is described. The model is compared against analytical solutions. Modeling of a scaled fuel melt progression is simulated for different conditions showing the sensitivities of melting fuel to conditions in the coolant channel.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

ANS	Advanced Neutron Source
API	Application Programming Interface
BR2	Belgian Reactor 2
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
FCI	Fuel Coolant Interaction
GPU	Graphics Processing Unit
HEU	Highly Enriched Uranium
HFIR	High Flux Isotope Reactor
IAEA	International Atomic Energy Agency
LEU	Low-Enriched Uranium
LOFA	Loss of Flow Accident
MPS	Moving Particle Semi-implicit
MTR	Materials Testing Reactor
ORNL	Oak Ridge National Laboratory
ORR	Oak Ridge Reactor
PDE	Partial Differential Equation
PRA	Probabilistic Risk Assessment
SAR	Safety Analysis Report
SCK·CEN	Belgian Nuclear Research Centre
SPH	Smoothed Particle Hydrodynamics
VTK	Visualization Toolkit

List of Attachments

Chapter 1

Introduction

1.1 High Flux Isotope Reactor

The High Flux Isotope Reactor (HFIR) is a research reactor for neutron scattering research and isotope production at Oak Ridge National Laboratory (ORNL). The HFIR is a pressurized reactor moderated and cooled by light water. Figure 1.1 shows the reactor vessel, which is located inside a pool of water. The reactor was designed for 100 MW operation but currently operates at 85 MW. Water cools the reactor flowing at nominally 16,000 gal/min (1009 L/s) where approximately 13,500 gal/min flows through the core. The core, shown in Figure 1.2, is composed of two concentric annuli with 171 involute fuel plates in the inner annulus and 369 involute fuel plates in the outer annulus. Each fuel plate is 24 in. long, of which 20 in. contains the active fuel U_3O_8. The fuel plate is 0.050 in. thick and separated by a coolant channel that is also 0.050 in. in thickness. This high performance reactor has a power density that is about 17 times that of a commercial nuclear power plant [1]. A typical fuel operational cycle is 23 to 26 days at 85 MW.

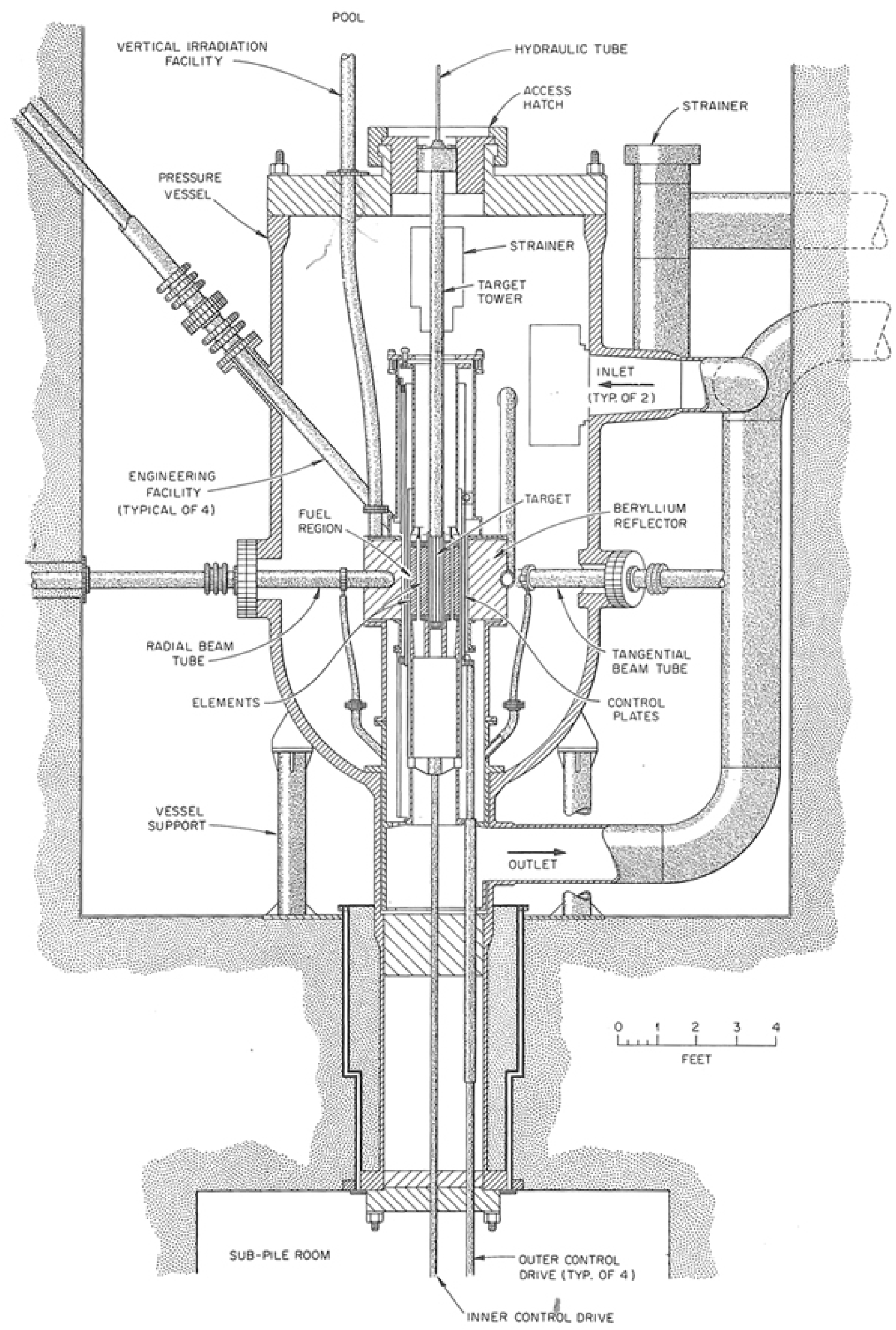

Figure 1.1: HFIR dimensions [2]

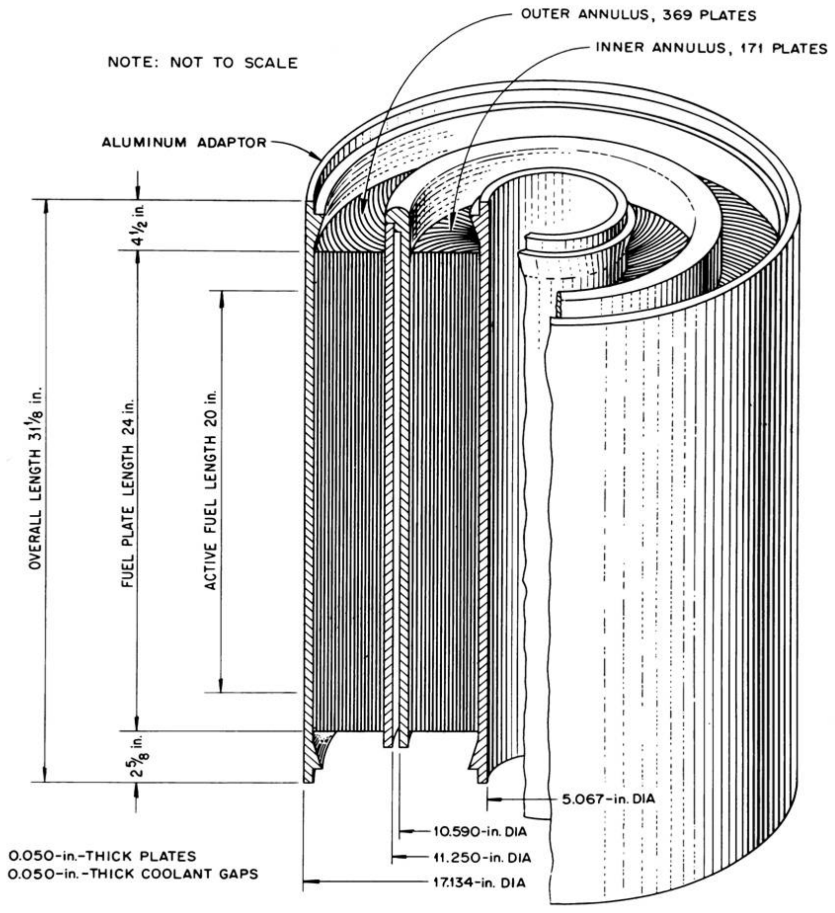

Figure 1.2: HFIR Core [3]

1.1.1 Flow Blockage

Flow blockage occurs when there is an obstruction at the entrance to the coolant channel, which may be in the form of broken internal reactor component parts or foreign objects introduced during maintenance, which impede flow along the coolant channel. The degree of inlet blockage may vary from partial to total for a single channel and could also span multiple channels. Blockage can occur at the inlet or downstream, but inlet flow blockages are the most likely because of the downward core flow direction and are the focus of safety analysis efforts.

A Probabilistic Risk Assessment (PRA) study of HFIR concluded core damage due to flow blockage accounts for 20% of the combined internal and external-events induced core damage frequency [4]. For internal events, flow blockage core damage frequency accounts for 49% [5] of the fuel damage risk. Materials Testing Reactor (MTR) type reactors are usually designed with downflow and material test experiments and refueling offers opportunity for foreign objects to enter the primary flow circuit just upstream of the core several times during each year of operation. The relatively high power density of MTR reactors makes inlet flow blockage a likely precursor to fuel damage. Two fuel damage events caused by inlet flow blockage have occurred in MTR reactors as will be presented in some detail in Section 1.1.3.

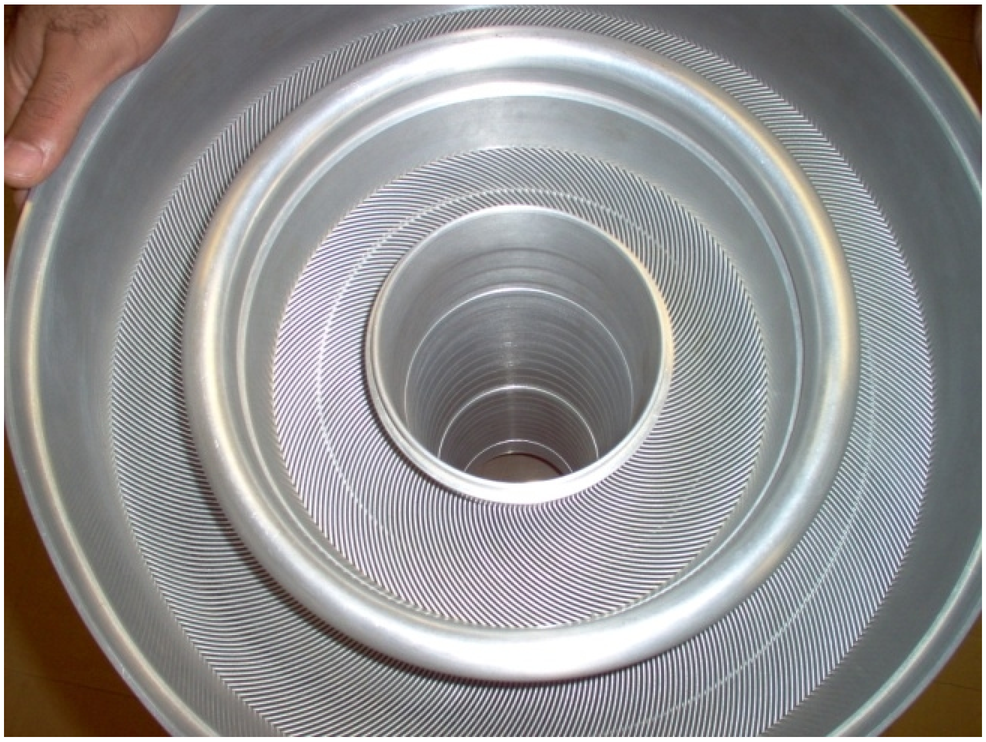

Figure 1.3: Top view of HFIR core to scale [6].

1.1.2 Flow Blockage studies

Several flow blockage studies have been performed for MTR [7, 8, 9]. Some of the flow blockage studies are shown in Table 1.1. All studies listed in Table 1.1 are numerical, mostly using RELAP or the commercial CFD software Fluent. Only the last study, the Advanced Neutron Source flow blockage study compares numerical results to experimental data.

Table 1.1: Flow Blockage Studies

Studies	What was studied	Result Summary
MTR LOFA [10, 11]	Various codes compared	Two hotspots present, flow inversion by natural convection
MTR LOFA [12]	Fluent 12 2D	Clad temp. below melting, flow reversal seen
MTR FB [7]	9 channel, partial and full blockage using RELAP5	No boiling with total blockage, adjacent channels remove heat
MTR FB [8]	2D Fluent 6.2 comparison to MTRTHA 1D	Hot channel at 90% blockage predicts boiling
MTR FB [9]	95% blockage single channel	8sec., clad melting temp. reached
ANS [13]	Effect of blockage shape and position on downstream flow property using Fluent	Shape has effect on flow, reattachment length

The first study [10] is a CFD analysis of a fast loss of flow accident (LOFA) in the IAEA generic 10 MW MTR using Fluent 6.2.16. This study compares Fluent results to RTRTH, RELAP5/3.2, PARET, PETRAC-PC, COSTAX, EUREKA, COBRA and NSTRI for benchmarking loss of flow transients. It models a down flow reactor initially operating at steady state followed by a pump coast down. Control rods scram when exponentially decaying flow rate with a time constant of one second is 85% of the nominal value. When the flow reaches 85% nominal, the power drops due to the scram. At around three seconds it is thought that the flow transitions into laminar flow thus decreasing heat transfer resulting in increasing temperature. In a similar study [11] of slow LOFA (time constant of

25 s) the result is nearly identical to the fast loss of flow transient.

The work described in Ref. [12] uses Fluent 12 to analyze the fast loss-of-flow accident in the IAEA 10 MW generic research reactor under a hot channel condition with one channel subject to inlet area contractions up to 80%. All loss of flow accidents simulations in [12] predict clad temperature below melting. Past studies considered 2D steady state and this study considers 2D transients. The channel conditions assumed a constant pressure drop and constant temperature at the channel exit. This assumption had little effect on peak temperature for short term runs. Mass flow rates increase in adjacent channels as blockage ratio increases and showed little impact on channels further away. Flow reversal occurs in all three channels simultaneously. Boiling was found to be inevitable for blockage ratios above 20%.

A RELAP mod 3.3 study [7], investigated partial (95%) and full blockage of a channel in an IAEA 10 MW MTR assembly without scram. Nine channels were modeled with one channel using valve component model (for restricting flow) to simulate blockage. Symmetrically cooled fuel showed almost no difference in temperature distribution between partial and full blockage. Under asymmetric cooling, the temperature in the blocked channel was similar to the outer clad temperature of the neighboring fuel plate. The analysis showed no boiling occurs during total blockage due to heat transfer to adjacent channels. However, a similar study [8] looking at flow blockage for the generic MTR found boiling occurring in the blocked channel. It used Fluent 6.2.16 in 2D for flow blockage analysis of a hot and average power channel. The hot channel at 90% blockage predicted boiling, whereas the average channel shows no boiling at full blockage.

Another study [9] of the IAEA 10 MW MTR flow blockage of a single assembly found cladding melting temperatures. Results were obtained using the RELAP5/mod3.3 valve component to simulate inlet flow blockage for 95% and full closure. All transients started

50 seconds after the simulation reached steady state. For 95% blockage, negative void reactivity feed-back lowered the power to a “sufficient” level in about 200 seconds to preserve fuel integrity. For full blockage, the calculation is stopped after eight seconds of transient when the cladding reaches melting temperature. The paper [9] states this result to be “unrealistic and highly conservative” as the point kinetics reactivity model does not take into account the local power change due to void distribution in the channel.

A thesis [13] experimentally studied the effect of shape and position of the object blocking a channel in ANS MTR type reactor. The results were also compared with Fluent simulations under the same flow conditions in 2D. The study found that the shape of the object blocking the flow impacted the heat transfer recovery length downstream.

1.1.3 Fuel Melts

Flow blockage(s) reduce flow which lowers the fuel-to-coolant heat transfer coefficient and elevates the coolant bulk temperature. During the short period of time when fission power is maintained at 100% or near 100%, both of these effects cause an elevation in the clad and coolant temperature. As the clad surface temperature increases beyond saturation, water bubbles will start to form on the surface and start boiling. The bubbles can further elevate resistance to coolant flow, contributing to further degradation in heat transfer from the fuel plates and transition to steam cooling or film boiling. Eventually, section(s) of the fuel with the highest power density, which is usually near the center, reach the melting point of the aluminum used in the fuel matrix material. When the clad surface temperature reaches the melting temperature, the molten fuel will start to flow out into the coolant. The molten fuel may travel along the coolant channel and cool in the unfueled section of the fuel plate. More melting downstream of the initial melt location can be caused by the molten fuel if it further blocks flow of the coolant. This feedback mechanism will likely continue until the abnormal activity caused by melted fuel in the primary coolant is detected and the reactor is scrammed.

Flow blockages have resulted in fuel melt in high performance plate fueled reactors. In 1963, a 24 MW material testing reactor, the Oak Ridge Reactor (ORR) at ORNL, experienced flow blockage by a neoprene gasket material and melted fuel as shown in Figure 1.4. In 1975, the BR2 reactor at SCK·CEN in Belgium melted fuel after a screwdriver blocked a fuel channel [14]. The BR2 reactor fuel plates melted at high power density/heat flux locations in a manner that appears similar to the ORR case as shown in Figure 1.5. Figure 1.6 shows the melted fuel plates and blocked coolant channel. The reactor operated for a few hours at 48 MW before stopping due to high fission product activity [15]. It was estimated that 40 - 120 cm^2 of the fuel plate melted and 6 g of uranium was lost to the coolant [14].

Partial core coolant inlet flow blockage is unique in that it is an accident that may progress significantly before detection and reactor scram. The initial flow blockage has the potential to melt fuel that will migrate to cooling channels, thus blocking flow to additional plates and causing more damage. In reactor designs such as the HFIR, fuel melting due to a small blockage can go undetected by the safety system. Further, initial void production caused by a blockage of flow will depress power, but the reactor control system will move to restore full power until activated coolant passes radiation monitors and causes an emergency shut down. In the case of the HFIR, the transport time for radionuclides to reach the coolant radiation monitors is about two seconds.

There are two predicted core melt fractions for the HFIR, 14% [16] and 24% [17]. The 14% core melt prediction assumes that an initial blockage causes a melting plate that will slump against the neighboring plate in a domino manner, causing melting of the adjacent plate. This process proceeds for three seconds until the melting is detected and the reactor scrams [16]. It assumes a heat transfer coefficient of zero which result in 77 fuel plates melting before the reactor scrams. This conclusion is a time based melting calculation with no credit for partial heat transfer from the unblocked cooling channel. The 24% core melt is based on the fraction of the core that must melt to override a one dollar (maximum available) reactivity addition from reactor control system-driven servo response. The servo response is motivated by vapor production from the flow blockage [17]. Large flow blockages may result in less fuel damage due to reactor shutdown by safety system flow and pressure sensors, which are expected to initiate a reactor scram in a more prompt timescale.

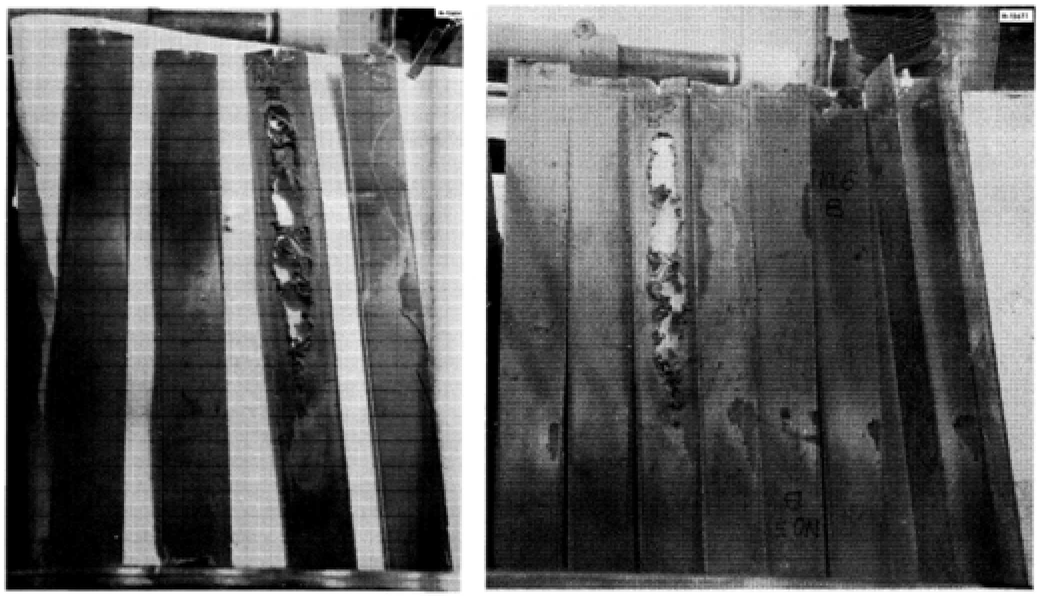
Figure 1.4: Convex (left) and concave (right) side of the fuel plates from ORR [18].

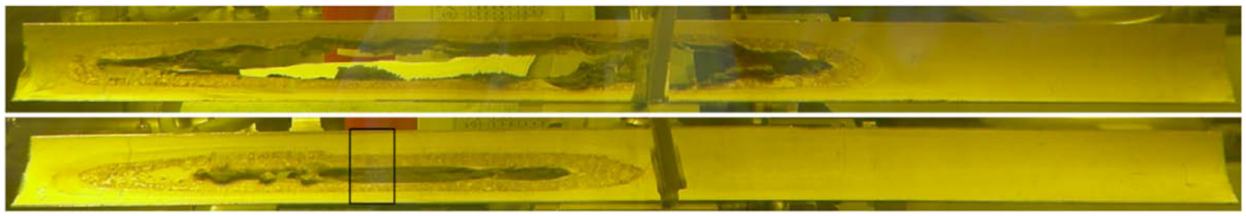
Figure 1.5: The BR2 melt [14]. Reproduced here with permission from Elsevier.

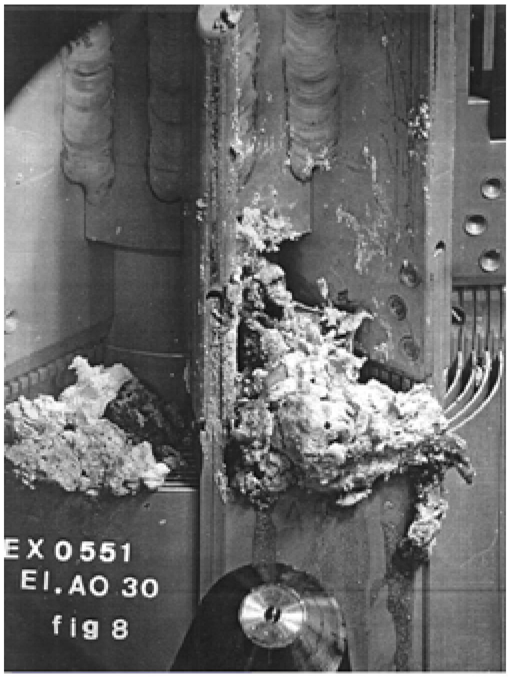

Figure 1.6: The BR2 relocated melt [15].

1.1.4 Current Fuel Melt Progression Hypothesis and Consequence

This section summarizes the progression and consequence of fuel melting as taken from a recent review of this topic [19]. The initial concern during flow blockage is whether fuel melting occurs. For low probability events some degree of melting is allowed by the safety basis. For those cases, the consequence analysis may credit integrity of the reactor vessel or the reactor confinement to limit off site dose levels.

Figure 1.7 shows the different event paths following flow blockage. The most benign outcome following flow blockage is no melting. In this case, inlet blockage will likely go undetected until refueling. The conservative assumption taken in the current safety analysis report (SAR) is for fuel melting to take place. Molten fuel may relocate, continue melting or mix with coolant. The event path of concern is melt propagation and premixing of fuel and coolant leading to a steam explosion, as it is the outcome most likely to cause vessel failure and off site release of radiation. The SAR assumes a flow blockage will lead to melting fuel that then results in Fuel Coolant Interaction (FCI). FCI occurs when hot molten fuel mixes with sub-cooled coolant leading to a steam explosion. A more detailed description of FCI and steam explosion follows.

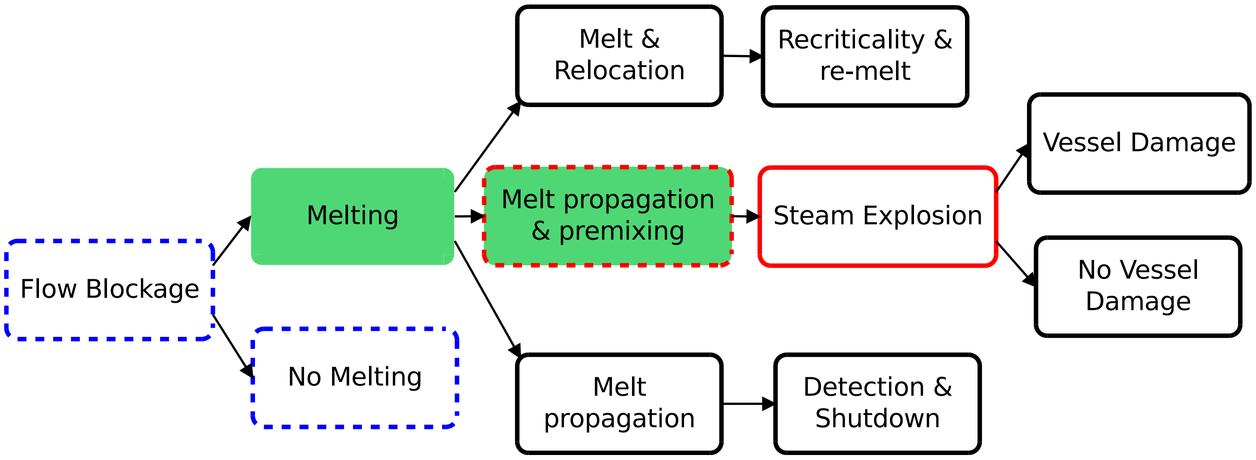

Figure 1.7: Event sequence following flow blockage [19].

Under flow blockage conditions, the decrease (or loss) of heat removal capacity results in rapid fuel heat up. For high performance reactors like HFIR, the adiabatic heat up rate is over ∼1300 °C/s. The aluminum melts at 660 °C and is initially near 164 °C, so fuel melt occurs rapidly following a complete loss of cooling.

As molten fuel downstream of a blockage transfers heat, a stable vapor film is formed between the melt and surrounding coolant which limits the rate of heat transfer. However, the molten fuel continues to generate power and heat up. The stable vapor film surrounding the melt can become unstable allowing the water to contact the molten aluminum, creating high heat transfer rates and a correspondingly large volume production of steam. This can lead to the formation of shock waves [20]. Aluminum will strip oxygen away from water at high temperatures, leading to exothermic chemical energy release, and production of free hydrogen. The expanding shock wave could damage additional fuel and create fine fuel fragments which could promote further energy release and chemical reaction between Aluminum cladding and water [20]. This increase in interfacial area contributes positive feedback to the FCI process. The shock wave and rapid expansion of high pressure vapor could stress the reactor vessel and could lead to off site release of radioactive material if the pressure vessel fails.

The ORR and the BR2 both melted fuel but did not experience a steam explosion. At present, a mechanistic tool for simulating melt conditions from flow blockage to the early fuel interaction with the coolant is not available for MTR fueled reactors. Therefore, the current HFIR SAR conservatively assumes flow blockage will lead to melting followed by a steam explosion [21] (Figure 1.7 red). The energy release of the steam explosion is used in a structural code to determine the potential vessel damage.

1.2 Fuel Melt Progression Model

Figure 1.8 shows a single fuel plate along with its 2D representation. The schematic shows the melting of the fuel matrix (red) and Aluminum cladding (gray) into the coolant channel (blue). The inlet flow is downward in the direction of gravity. As the materials heat up, their boundaries deform. Some of the molten material may separate and disperse. The introduction of melt material will affect the flow field in the channel which in turn affects the heat transfer. The relevant physics are:

- Neutronics: The rate of heat generation due to fission deposited inside the plate and carried along with the molten fuel. The production of vapor reduces moderator density and also impacts local power production.

- Chemistry: The build up of Aluminum oxide which affects heat transfer and potential ignition of Aluminum during FCI.

- Microstructure: Formation of fission gas bubbles and their release beginning at the fuel plate blistering temperature.
- Structural mechanics: The internal stresses caused by the thermal expansion, fission gas and external stress caused by fluid flow.
- Fluid mechanics: Flow of coolant, bubbles, and melt and their interactions.
- Phase changes: The boiling of coolant, the melting and solidification of fuel.
- Heat Transfer: The transfer of heat from the fission source and the effect on phase change.

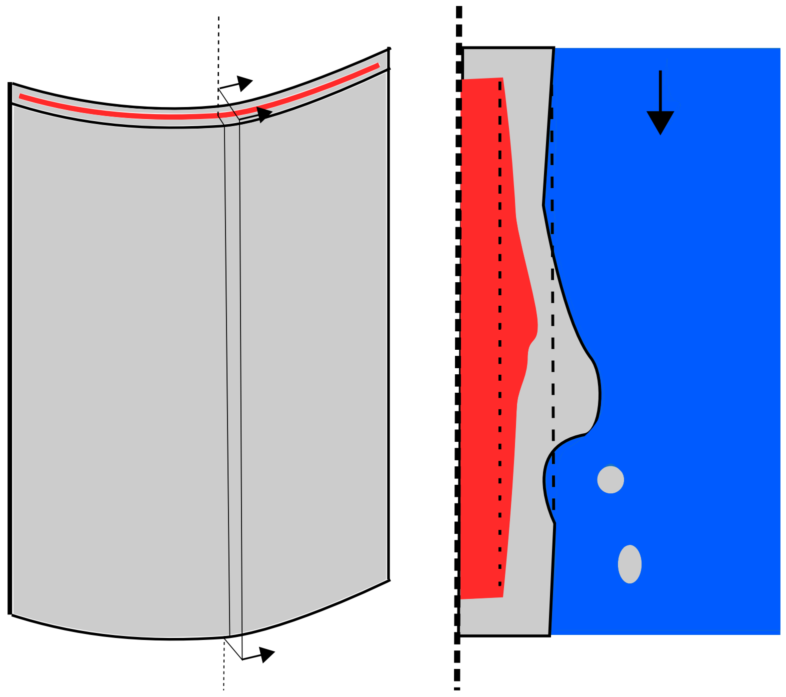

Figure 1.8: Cut plane (left) and schematic fuel plate melting (right) of half the plate.

1.3 Contributions

The Smoothed Particle Hydrodynamics (SPH) model developed herein addresses fluid mechanics, phase change, and heat transfer, the last three of the phenomena listed in the previous section. This model formulates a mechanistic multicomponent melt model based on SPH. The model is tested and used to simulate fuel plate melt during flow blockage. The original contributions of this work follow:

- flow blockage is modeled mechanistically through fuel melt for MTR reactor fuel.
- multicomponent SPH model is developed and implemented on GPU. Multifluid SPH works have been theoretical and no algorithms are discussed. Most prior SPH implementations are single phase, or involve one free surface.
- multicomponent Smoothed Particle Hydrodynamics is combined with heat transfer and phase change. No prior treatment of this multiphysics case exists in SPH.
- inflow/outflow computational "domain" boundary created in SPH.

1.4 Structure of the Dissertation

- Chapter 2 discusses the theory of SPH and the discretization for fluid mechanics and heat transfer. A phase change model is combined with the fluid and heat transfer model to create a multicomponent melt model, the first of its kind for SPH.
- Chapter 3 describes the implementation of the SPH model for Compute Unified Device Architecture (CUDA), the programming language for NVIDIA GPU.
- Chapter 4 compares the model against analytical solutions. This provides validation of the new SPH model implementation.

- Chapter 5 presents the simulation of fuel melting.
- Chapter 6 presents a summary of this work, reiteration of original contributions, and suggested future improvements.

Chapter 2

Melting Model

2.1 Eulerian and Lagrangian Flow

In computational fluid dynamics (CFD) and structural codes, the predominate computational approach is to discretize the domain and approximate the governing partial differential equation(s). Solutions based on a mesh fixed in space through which the fluid moves are Eulerian. The Eulerian approach to CFD is well developed and robust for many applications.

Typically, traditional mesh-based computational methods struggle with moving boundary problems (e.g. melting fuel). In Figure 2.1, the three left figures show mesh-based Eulerian methods and the right-most figure shows a meshless Lagrangian method. The figures represent two material properties with the region in blue moving towards the white region to the right. In the context of melting fuel plates, the molten fuel (blue) is moving into the coolant (white). In order to understand the effects of molten fuel, the understanding of the interface between the two materials is crucial. One must predict where this interface is located in time.

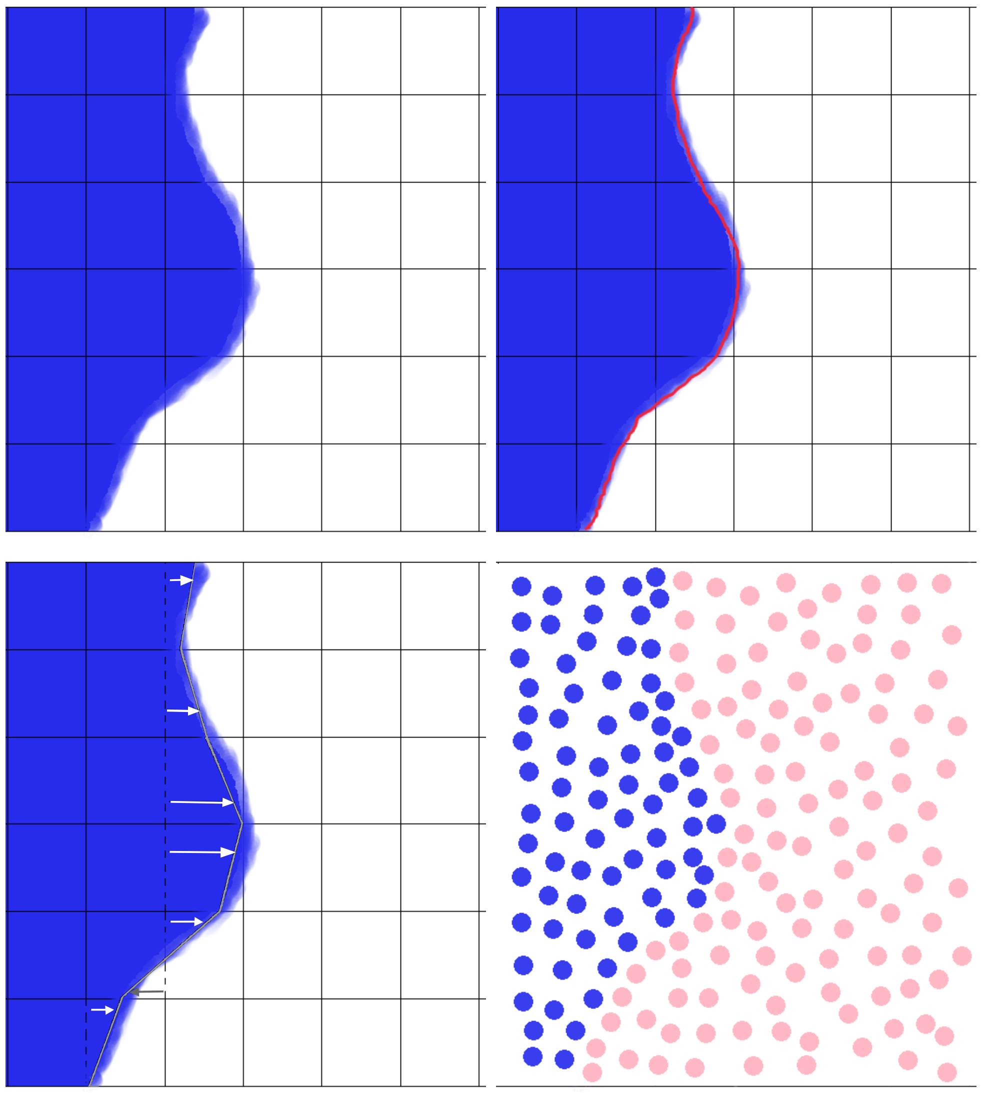

Figure 2.1: Mesh-based Eulerian (top and bottom left) and meshless Lagrangian method (bottom right).

In the top left most figure, the interface lies within the width of one mesh. This rather coarse mesh is not sufficient to resolve the interface and is also likely to be numerically unstable for large differences in material properties. The interface may be resolved by interface capture or tracking methods as the top left figure. This is the most common approach for mesh-based methods. These methods, such as volume of fluid or level set, require solving additional partial differential equations (PDE) describing the convection at the interface, plus mass, energy, and momentum transport equations across the interface. For problems with two phases of a single component, the interface tracking approach is acceptable. But for the simulation of melting fuel plates, a model of two materials with two phases each (solid and liquid fuel, liquid and vapor coolant) interface tracking becomes impractical for Eulerian methods.

Instead of computing extra PDEs, the interface can be explicitly tracked by moving the mesh to follow the material as shown in the bottom right. This method is suitable for structural problems where small displacements can be accurately tracked. However, for large material deformation during melts, the mesh will eventually entangle and fail. In addition, moving meshes is computationally expensive, especially for finite element methods because a system of linear equations must be reformulated with every time step. All these methods arise to handle convective flow of the fluid and the moving fluid boundary in a fixed computational mesh. Moving boundary problems are very difficult for mesh-based methods.

An alternative is to take a meshless Lagrangian scheme as shown in Figure 2.1 (bottom right). In contrast to the previous methods, the material properties reside with particles which move with the flow. The moving boundary problem that challenges mesh-based Eulerian methods is then handled naturally. The Lagrangian method explicitly moves the material with the flow, which makes schemes for interface tracking unnecessary.

2.2 Introduction to Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is one of the first meshless methods and was originally introduced independently by Lucy [22] and Gingold and Monaghan [23] in 1977. Most Lagrangian particle methods originate from SPH. SPH has attracted considerable interest and has wide engineering application. Some applications of SPH include free surface problems(e.g. wave breaking, sloshing), elasticity and fracture, high explosive detonation, underwater explosions, fluid-structure interaction, multiphase flow, and freezing of alloys [24]. It is also commonly used in physics based visual simulation for games and films [25].

Kernel approximations and particle approximation are the two key aspects of SPH. The SPH method transforms the differential equation of fluid dynamics into particle summations. Field variables and their derivatives for the governing equations are represented by smoothing functions called kernels.

2.2.1 Mathematical Basis

The basic idea of SPH is to approximate the field function by an integral representation at a position. The continuous SPH is based on the integral representing an arbitrary function $A(r)$ as

$$A(r) = \int_{\Omega} A(r')\delta(r - r', h)dr' \tag{2.1}$$

where δ is the Dirac delta function. In order to apply SPH using floating-point arithmetic, the Dirac delta function is replaced by a smoothing function W, such that the arbitrary function $A(r)$ becomes

$$\langle A(r) \rangle = \int_{\Omega} A(r')W(r - r', h)dr' \tag{2.2}$$

where h is a measure of support on W and W is the weighting function. In the context of particle i, h represents the sphere of influence of A in Ω_i. The h is called the smoothing length and W is known as the kernel.

The kernel function is constructed such that

$$\int_{\Omega} W(r,h)dr = 1, \tag{2.3}$$

this is the normalization condition.

The kernel satisfies the Delta function property as the kernel support length h approaches zero.

$$\lim_{h\to 0} W(r-r',h) = \delta(r-r') \tag{2.4}$$

The kernel function is radially symmetric and only locally supported in that,

$$W(r,h) = C(n)\begin{cases} f(r) & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where $C(n)$ is a coefficient that depends on the dimension $\mathbb{R}^n$. By equation (2.5), the kernel is defined to have only local interaction.

Figure 2.2 shows a kernel function W for a particle at the center. The interaction is limited to neighboring particles within $|\vec{\mathbf{r}}| < r_e$. Like the Delta function, the kernel is a positive and monotonically decreasing function. Monotonicity guarantees that the strength of the interaction diminishes with increasing distance.

Furthermore, the derivative of the kernel W

$$\nabla W = -\nabla W \tag{2.6}$$

is symmetric. This implies that interaction of particles i and j are such that, $i \rightarrow j$ is equivalent to $i \leftarrow j$.

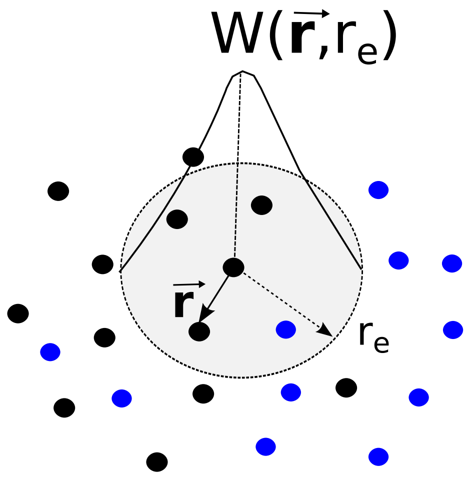

Figure 2.2: Kernel and its interaction in 2D.

A field quantity $A(r)$ (scalar or vector) by Taylor-series expansion is approximated as [26]:

$$A(r) = \int_{\Omega} A(r')W(r - r', h)dr' + \mathcal{O}(h^2) \tag{2.7}$$

It is also possible to construct higher order approximations by adding more terms to the expansion. However, doing so can be problematic where the nth derivative of the kernel takes a negative value (e.g. negative density evaluation). This condition will be discussed in Section 2.3.4. In practice, more terms also increase computational requirement. Hence, second-order accurate SPH is common.

The domain of the problem is approximated by particles. The particle represents field properties and material properties. Interactions that many occur such as heat transfer and momentum transfer between different materials is explicitly calculated as particle interactions. The different particle colors represented in Figure 2.2 interact within the region of influence of the kernel W. In SPH all particles have at least the property of position $\mathbf{r}$, mass m, and density ρ. The field A is then written by dropping the error term in equation (2.7).

$$\begin{aligned} A(r) \cong \langle A(r) \rangle &= \int_{\Omega} A(r')W(r-r',h)dr' \\ &= \int_{\Omega} \frac{A(r')}{\rho(r')} W(r-r',h)\rho(r')dr' \end{aligned} \tag{2.8}$$

In equation (2.8) the ρdr is the mass of the particle. The integral is the summation over all the particles in the domain Ω.

$$\langle A_i \rangle = \sum_j A_j W_{ij} \frac{m_j}{\rho_j} \tag{2.9}$$

where $W_{ij} = W(r_i - r_j, h)$

2.2.2 Differentiation

The derivative of field $A(r)$ is

$$\nabla A(r) \cong \langle \nabla A(r) \rangle = \int_{\Omega} \nabla A(r')W(r-r',h)dr' \tag{2.10}$$

Taking equation (2.10) and integrating by parts

$$\langle \nabla A(r) \rangle = \int_{\partial\Omega} A(r')W(r-r',h)\hat{\boldsymbol{n}}dS - \int_{\Omega} A(r')\nabla W(r-r',h)dr' \tag{2.11}$$

where $\hat{\boldsymbol{n}}$ is the unit vector normal on the surface S. The smoothing function W by definition has compact support (equation (2.5)). Therefore, integrating over the domain outside h makes the first term on the right hand side of equation (2.11) zero. Near the domain boundary this term is not zero and therefore requires special consideration.

The above equation (2.11) simplifies to

$$\begin{aligned} \langle \nabla A(r) \rangle &= -\int_\Omega A(r')\nabla W(r-r',h)dr' && (2.12)\\ &= \int_\Omega A(r')\nabla W(r-r',h)dr' && (2.13) \end{aligned}$$

By the symmetry of the kernel W the two right hand side terms above are equal.

Following equation (2.13), the derivatives of an arbitrary field $A(r)$ in SPH is the derivative on the kernel. The kernel is defined to have the form,

$$W = \frac{C}{h^d} f(r,h) \tag{2.14}$$

where C is the normalization constant, the superscript d is the dimension, and f is some function. The normalization term, $\frac{C}{h^d}$ is explicitly represented. This term ensures the condition in equation (2.3) is met. This term is obtained by integrating over the domain Ω as

$$\int_\Omega f(r,h)d\Omega = kh^d \tag{2.15}$$

Then the normalization term for the kernel is $1/kh^d$ and rewritten as C/h^d. The kernel W must at least be the same order as the differential operator in order to express the operation. For example, the first derivative is

$$\nabla W = \frac{C}{h^{d+1}} f'(r,h) \tag{2.16}$$

There are various formulations of the gradient. One such form for the gradient of a field A is:

$$\begin{aligned}\langle \nabla A(r) \rangle_i &= \sum_j \frac{m_j}{\rho_j} A_j(r) \nabla_i W(r_{ij}, h) && (2.17)\\ &= \sum_j \frac{m_j}{\rho_j} A_j(r) W'(r_{ij}, h) \hat{\mathbf{r}}_{ij} && (2.18)\end{aligned}$$

where m is the mass and ρ is the density of a point (particle). However, if A is a constant this form may not vanish. To correct this [27]

$$\nabla A_i = \frac{1}{\Phi_i} \sum_j m_j \frac{\Phi_j}{\rho_j} (A_j - A_i) \nabla W_{ij} \qquad (2.19)$$

where Φ is a differentiable function. This is the common first derivative form.

The Laplacian is written as

$$\begin{aligned}\left\langle \nabla^2 A(r) \right\rangle_i &= \sum_j \frac{m_j}{\rho_j} A_j(r) \nabla_i^2 W(r_{ij}, h) && (2.20)\\ &= \sum_j \frac{m_j}{\rho_j} A_j(r) W''(r_{ij}, h) && (2.21)\end{aligned}$$

2.2.3 Kernel functions

Any choice of kernel functions meeting the requirements discussed in the previous section can be used. Indeed, there are many kernel functions in SPH. The choice of the kernel function will influence accuracy, stability, and cost as a function of attributes of the simulation. High order kernels require more computational resources. Piecewise kernels have branching by conditional statements that are costly for GPU computation. Some example kernel functions are described in this section.

Gaussian

The Gaussian function lacks compact support. This means that interactions between every particle in the domain must be evaluated. Obviously, such computation is undesirable and to circumvent this a cut-off distance is used. The modified form is known to have the best stability properties [28]. The Gaussian kernel is

$$W = \frac{C}{h^d} \exp(-q^2) \tag{2.22}$$

where $C = 1/\pi$ in 2D and $1/\pi^{3/2}$ in 3D, d is the dimension, $q = r_{ij}/h$ and $r_{ij} = |r_i - r_j|$ the distance between particle i and j .

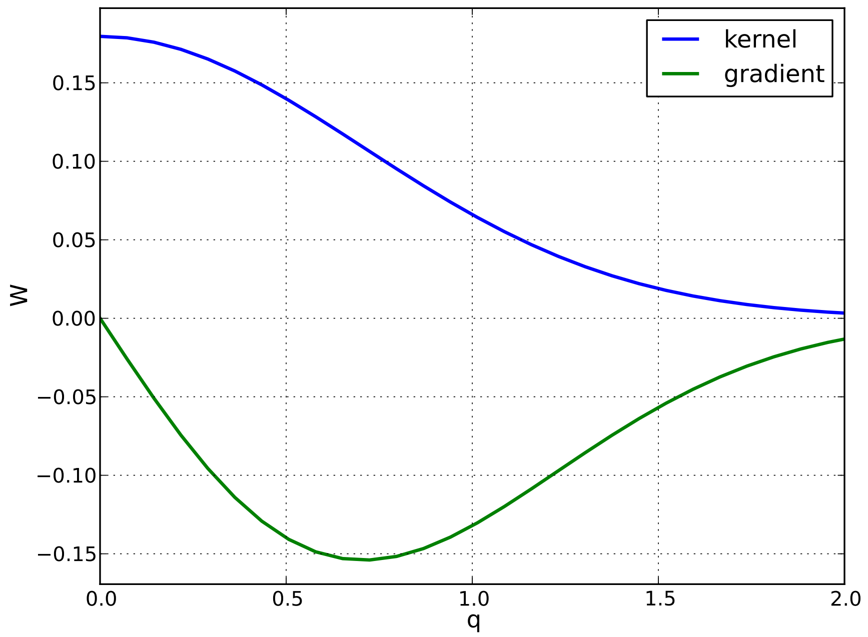

Figure 2.3: Gaussian kernel

Piecewise Cubic spline

The cubic spline introduced by Monaghan and Lattanzio (1985) [29] is one of the most popular kernels.

$$W = \frac{C}{h^d} \begin{cases} 4 - 6q^2 + 3q^3 & \text{if } 0 \leq q \leq 1 \\ (2 - q^2)^3 & \text{if } 1 \leq q \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{2.23}$$

where the constant $C = 10/28\pi$ in 2D and $1/4\pi$ in 3D. This kernel has a cut-off distance of $2h$. Higher order kernels of this class are also possible choices and may improve stability [28].

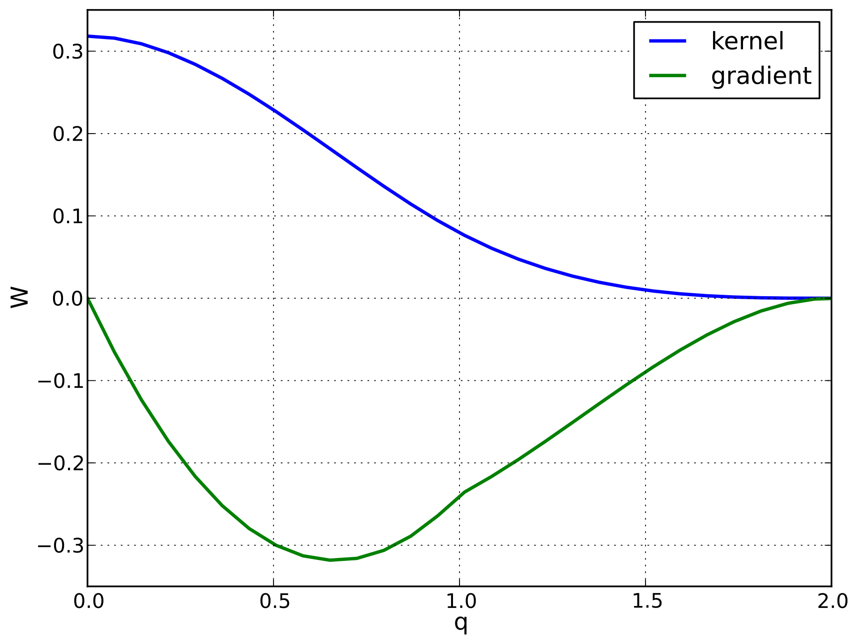

Figure 2.4: Cubic spline

Wendland

Another common kernel is the Wendland (1995) [30]. Unlike the cubic spline, this kernel is not a piecewise function, making it more computationally appealing for GPU application since no branching is possible. The Wendland kernel follows as,

$$W = \frac{C}{h^d}\left(1 - \frac{q}{2}\right)^4 (2q+1) \text{ for } 0 \le q \le 2 \tag{2.24}$$

where the constant $C = 7/4\pi$ in 2D and $21/16\pi$ in 3D. This kernel is used in the fluid momentum and continuity equations in this dissertation as the default choice. The Wendland kernel reduces particle clumping; the unphysical and undesirable grouping of particles [31].

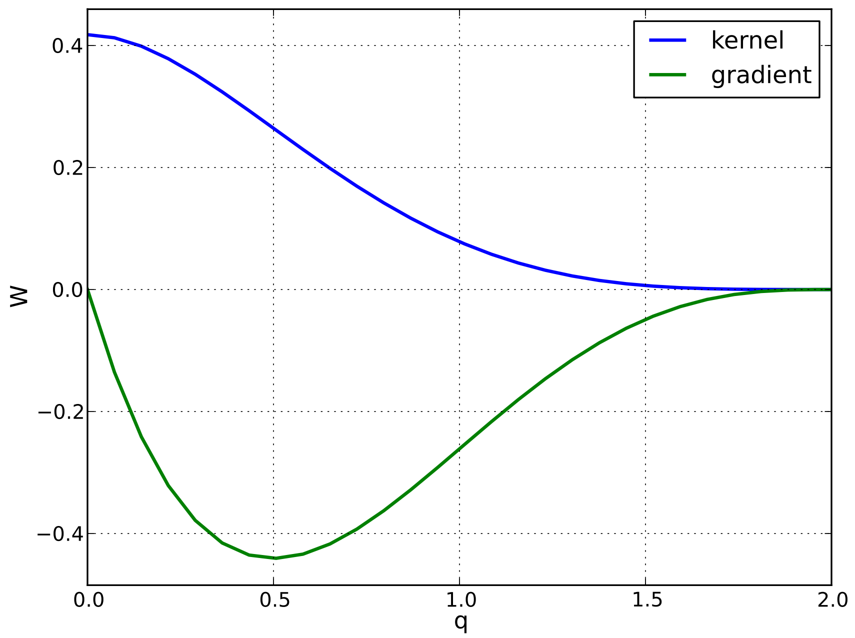

Figure 2.5: Wendland kernel

2.3 Fluid Mechanics in SPH

This section describes the governing fluid equation for SPH. The momentum and continuity equations of the Navier-Stokes equations for incompressible flow are introduced. First the commonly used “Standard form” in SPH for single phase flow is presented. The application of the Standard form is commonly used in free-surface flows.

The conservation of mass equation in the Eulerian frame is

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{2.25}$$

where $\mathbf{u}$ is the particle velocity and ρ is the density.

The conservation of momentum equation in the Eulerian frame is

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{2.26}$$

where p is the pressure, ν is the kinematic viscosity, and $\mathbf{f}$ is the body forces.

The material derivative relates the Eulerian and Lagrangian frames.

$$\frac{D(\cdot)}{Dt} = \frac{\partial (\cdot)}{\partial t} + \mathbf{u} \cdot \nabla (\cdot) \tag{2.27}$$

Rewriting the conservation of mass, equation (2.25), in Lagrangian view

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{u} = 0 \tag{2.28}$$

For particle methods the Lagrangian view of acceleration is written as:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \frac{D\mathbf{u}}{Dt} = \mathbf{a} \tag{2.29}$$

Then the momentum equation for a particle is :

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{2.30}$$

2.3.1 Standard Form

Continuity

Setting field A in equation (2.9) to ρ, the simplest density approximation becomes

$$\rho_i = \sum_j m_j W_{ij} \tag{2.31}$$

A quasi-compressible formulation is commonly used for incompressible flow in SPH. The variation in density is around 1%. This weak compressibility is calculated by integrating equation (2.28) instead of the direction summation. This alternative form is obtained from equation (2.28) where

$$\nabla \cdot \mathbf{u}_j = \frac{1}{\rho_i} \sum_j m_j (\mathbf{u}_i - \mathbf{u}_j) \cdot \nabla W_{ij} \tag{2.32}$$

Then,

$$\frac{d\rho_i}{dt} = \sum_j m_j (\mathbf{u}_i - \mathbf{u}_j) \cdot \nabla W_{ij} \tag{2.33}$$

For particles near a discontinuous boundary, such as a free-surface or the interface between different fluids, this form tends to maintain density. For example, in free-surface flow problems the surface may not have sufficient particles. Thus, equation (2.31) will result in the smoothing of the density at the surface. This smoothing will also occur at the interface between different fluids. However, integrating equation (2.33) ($d\rho/dt$) preserves the discontinuity, leading to a better density representation.

Momentum

The pressure and viscosity terms in the momentum equation are discretized in SPH. The pressure term in equation (2.30) can be written as

$$\frac{1}{\rho}\nabla p = \frac{1}{\rho_i}\sum_j \frac{m_j}{\rho_j} p_j \nabla W_{ij} \tag{2.34}$$

The above form has an asymmetric force and cannot be used. The symmetric form is derived by

$$\frac{1}{\rho}\nabla p \approx \nabla\left(\frac{p}{\rho}\right) + \frac{p}{\rho^2}\nabla\rho \tag{2.35}$$

Discretized in SPH, the first term becomes

$$\nabla\left(\frac{p_i}{\rho_i}\right) = \sum_j \frac{m_j p_j}{\rho_j^2}\nabla W_{ij} \tag{2.36}$$

The second term is

$$\nabla\rho_i = \sum_j m_j \nabla W_{ij} \tag{2.37}$$

Combining the first and second terms

$$\frac{1}{\rho_i}\nabla p_i = \sum_j \frac{m_j p_j}{\rho_j^2}\nabla W_{ij} + \frac{p_i}{\rho_i^2}\sum_j m_j \nabla W_{ij} \tag{2.38}$$

$$= \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij} \tag{2.39}$$

This form is known to conserve linear and angular momentum [32].

The viscous term $\nu\nabla^2\mathbf{u}$ of equation (2.30) can also be written into SPH form. This results in the second derivative of the kernel. In practice, this form is uncommon as it is known to be sensitive to error for low resolution (low order spline kernels) [33]. Many viscosity terms without the second derivative term have been proposed [34, 35, 27]. The artificial viscosity proposed by Monaghan [36] is popular for its simplicity.

$$\Pi_{ij} = \begin{cases} -\alpha\dfrac{\bar{c}_{ij}\mu_{ij}}{\bar{\rho}_{ij}} & \text{if } \mathbf{u}_{ij}\cdot\mathbf{r}_{ij} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

where $\mathbf{u}_{ij} = \mathbf{u}_i - \mathbf{u}_j$, $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, $\bar{c}_{ij} = (c_i + c_j)/2$, $\bar{\rho}_{ij} = (\rho_i + \rho_j)/2$, $\mu_{ij} = h\mathbf{u}_{ij}\cdot\mathbf{r}_{ij}/(r_{ij}^2 + \delta^2)$, generally $\delta^2 = 0.01h^2$ to avoid singularity and α is a simulation dependent parameter (generally, $\alpha = 7$).

The full SPH momentum equation is

$$\frac{d\mathbf{u}_i}{dt} = -\sum_j m_j\left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} + \Pi_{ij}\right)\nabla W_{ij} + \mathbf{g}_i \tag{2.41}$$

2.3.2 Multi-fluid

Continuity

Early theoretical developments in multi-phase and multi-component modeling using SPH were first introduced in 2007 [37, 38].

Recently, Monaghan (2013) proposed a simple approach [39]. The continuity equation is

$$\frac{d\rho_i}{dt} = \rho_i \sum_j \frac{m_j}{\rho_j} \left(\mathbf{v}_i - \mathbf{v}_j\right) \cdot \nabla_i W_{ij} \tag{2.42}$$

The continuity equation above is slightly different from equation (2.33). For single fluid cases both equations are identical. However, when large density ratios exist, then the above form is more accurate [40].

Momentum

The momentum equation is

$$\begin{aligned} \frac{d\mathbf{v}_i}{dt} &= -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} + R_{ij} + \Pi_{ij}\right) \nabla_i W_{ij} \\ &\quad + \sum_b \left[m_b \mathbf{r}_{ib} f\left(|\mathbf{r}_{ib}|\right) - m_b \Pi_{ib} \nabla_i W_{ib}\right] + \mathbf{g}_i \end{aligned} \tag{2.43}$$

The last summation term in the momentum equation is for the boundary conditions. The first term is the wall boundary to repel particles from passing through the wall. The second term is the viscous term between the wall and the fluid. This boundary summation term will be dropped and will be discussed in Section 2.6.1. This simplifies the equation to

$$\frac{d\mathbf{v}_i}{dt} = -\sum_j m_j \left(\frac{P_i + P_j}{\rho_i \rho_j} + R_{ij} + \Pi_{ij} \right) \nabla_i W_{ij} + \mathbf{g}_i \tag{2.44}$$

In the first summation, the first term arises from the pressure gradient in the Navier-Stokes equation, the second term R_{ij} is a repulsion force that acts between different particle types and the last term Π is the viscous term. The repulsion force R_{ij} is necessary to prevent an unphysical particle mixing at the interface and follows the approach of Grenier et al. [38]

$$R_{ij} = C \left| \frac{\rho_{0i} - \rho_{0j}}{\rho_{0i} + \rho_{0j}} \right| \left| \frac{P_i + P_j}{\rho_i \rho_j} \right| \tag{2.45}$$

where the constant C is $0.01 - 0.1$ and ρ_0 is the rest density of the particle. It is worth noting that this repulsion force is only activated between different fluids.

The viscosity term for a single fluid is [27]

$$\Pi = -\frac{8\bar{\nu}\mathbf{u} \cdot \mathbf{r}}{\bar{\rho}|\mathbf{r}|\bar{h}} \tag{2.46}$$

where the overbar denotes mean values, e.g. $\bar{\rho} = (\rho_i + \rho_j)/2$.

A replacement is made for fluid particles with different viscosities, following [41],

$$\frac{\bar{\nu}}{\bar{\rho}} \to \frac{2\nu_i \nu_j}{\nu_i \rho_i + \nu_j \rho_j} \tag{2.47}$$

The viscous term for multi-fluid SPH becomes,

$$\Pi_{ij} = -\frac{16\nu_i \nu_j}{(\nu_i \rho_i + \nu_j \rho_j)} \frac{\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{\bar{h}_{ij}\,|\mathbf{r}_{ij}|} \tag{2.48}$$

2.3.3 Equation of State

The pressure is generally explicitly modeled in SPH by directly relating it to density. The simulation of fluid is therefore weakly compressible. Alternatively, an implicit method can be used to model pressure by directly solving the Poisson equation every time-step. For Lagrangian method like SPH, this is costly since the system of equations must be assembled every time and solved.

In this work the equation of state used is

$$P = B\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] \tag{2.49}$$

where $\gamma = 7$, $B = c_0^2\rho/\gamma$, $\rho = 1000$ for water and $c_0^2 = c^2(\rho_0) = \left.\frac{\partial P}{\partial \rho}\right|_{\rho_0}$ is the speed of sound squared at the reference density. This explicit formulation may lead to large pressure variation for small changes in density. The error in pressure is proportional to error in density and depending on γ this error is amplified. This has lead to another relationship through linearization that has been shown to produce satisfactory results [42]

$$P = c^2(\rho - \rho_0) \tag{2.50}$$

If the actual speed of sound is used, the Courant–Friedrichs–Lewy (CFL) condition requires the time step to be prohibitively small. For example, taking the HFIR coolant channel to be approximately 1 mm and with 10 particles across the channel. The smoothing length $h \approx 10^{-4}$ m and the speed of sound for water is $c \approx 1500$ m/s, which makes the time step $\Delta t = C\min(h/c) \sim 10^{-4}/c = 10^{-7}$ [43]. In order to keep a reasonable time-step, the speed of sound is defined in practice to $c_0 \geqq 10u_{max}$. This artificial speed of sound is a practical requirement for using SPH. The speed of sound is not usually important to the accuracy of low Mach number simulations.

2.3.4 Stability

Due to the particle formulation of flow, particles may not flow in a ordered fashion. A reoccurring problem that requires careful consideration in SPH is particle clumping. Under certain conditions, particles form a group (i.e. clump) which increases the error and causes stability issues. Particle clumping is a non-physical instability that can spread and break the simulation if it is not controlled. Uncertainty and error analysis in SPH is an area requiring further work. In a stationary flow field Swegle et al. [44] did a one dimensional analysis and found the condition for instability as

$$W''\sigma > 0 \tag{2.51}$$

where σ is the stress. The instabilities arises both for compression and tension. However, in general, SPH kernels lead to an instability when the material is under tension and therefore this instability is referred to as tensile instability.

A 2D SPH turbulence thesis by Robinson [31] used the Wendland kernel. The Wendland kernel was found to be stable but the Cubic spline kernel caused instabilities. The stability criteria offered by Swegle [44] would suggest similar kernels should exhibit similar stability behavior. The work [31] concludes that it is unclear if clumping in flow is caused by tensile instability alone, suggesting other causes.

Methods to control tensile instability were offered by Randles et al. [45, 46] and Dyka et al. [47]. It was shown that tensile instability is prevented by the introduction of an artificial pressure repulsion [48],

$$f_{ij} = R\frac{W(r_{ij})}{W(\Delta x_0)} \tag{2.52}$$

where Δx_0 is the initial particle spacing and R is a factor that is a function of pressure.

$$R = \epsilon \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \quad \text{if} \quad p_i,\, p_j > 0 \tag{2.53}$$

where typically $\epsilon = 0.01$. This repulsion force is included in the momentum equation (last term) which is,

$$\frac{P_i + P_j}{\rho_i \rho_j} + \Pi_{ij} + R f_{ij} \tag{2.54}$$

2.4 Energy Transport

2.4.1 SPH

The energy transport equation is

$$\rho c_p \left(\frac{\partial T}{\partial t} + \nabla \cdot (\mathbf{u} T) \right) = \nabla k \cdot \nabla T \qquad \text{in } \Omega \tag{2.55}$$

where ρ is the density, c_p is the specific heat capacity, T is the temperature, $\mathbf{u}$ is the velocity of the flow, and k is the thermal conductivity. Since SPH is in the moving particle frame of reference, it simplifies to a diffusion equation

$$\rho c_p \frac{dT}{dt} = \nabla k \cdot \nabla T \tag{2.56}$$

Heat conduction in SPH is [41]

$$c_{p,i} \frac{dT_i}{dt} = \sum_j \frac{4 m_j}{\rho_i \rho_j} \frac{k_i k_j}{k_i + k_j} (\rho_i + \rho_j) T_{ij} \frac{\mathbf{r}_{ij} \cdot \nabla W_{ij}}{|\mathbf{r}_{ij}|} \tag{2.57}$$

where $|\mathbf{r}_{ij}| = |\mathbf{r}_i - \mathbf{r}_j|$ the distance between particle i and j. This model was coded and comparison to an analytical solution was made. For unknown reasons this model showed the expected behavior of diffusion but failed to produce acceptable results. For details see Appendix A.

2.4.2 MPS

Because of the results of heat conduction using SPH, a different method of solving the energy equation in a compatible frame work as the SPH for fluid flow is necessary. An alternative discretization method similar to SPH is the MPS (Moving Particle Semi-Implicit) [49]. The MPS method is used for flow problems similar to those found in SPH. Like SPH, MPS also interpolates by kernels. A notable difference between MPS and SPH is that an implicit pressure calculation is performed for MPS. Instead of calculating pressure using density as in SPH, MPS solves the Poisson equation. This approach has a clear advantage of ensuring incompressibility $\nabla \cdot u = 0$ and does not suffer from tensile instability. To address the heat conduction issue with SPH, MPS discretization can be applied to just the energy equation.

In [50, 51] the discretization for the Laplacian is presented but does not show the MPS conduction equation used nor discuss their algorithm. It appears the topic of heat transfer using MPS is not represented in the literature.

Here, the necessary aspects of MPS in the context of conduction are presented. Numerous kernel functions W have been proposed for MPS, one such kernel is (Koshizuka 1998) [52]

$$W(r,h) = \begin{cases} \dfrac{h}{r} - 1 & \text{if } 0 \leq r \leq h \\ 0 & \text{otherwise} \end{cases} \tag{2.58}$$

If particles become very close such that a division by zero takes place. This may lead to erroneous heat transfer. This problem is exacerbated by particle clumping caused by the fluid momentum equation.

In contrast, the kernel offered by Shakibaeinia and Jin (2010) [53],

$$W(r,h) = \begin{cases} (1 - \dfrac{r}{h})^3 & \text{if } 0 \leq \frac{r}{h} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.59}$$

does not have the singularity problem.

MPS kernels conform to similar kernel properties as do SPH kernels. For example, they have compact support and vanish beyond $h = 2$. Four MPS kernels are show in Figure 2.6.

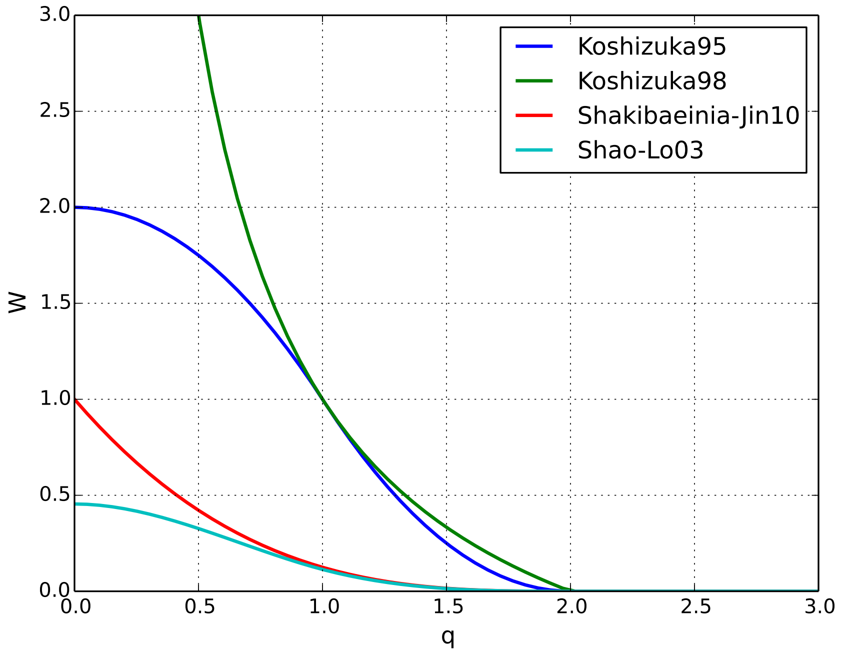

Figure 2.6: MPS kernels

The interaction of particles is local and is determined by the kernel. SPH defines density as a sum of locally weighted mass. MPS instead uses particle number density, defined as

$$n_i = \sum_{j \neq i} W(r_{ij}, r_e) \tag{2.60}$$

where $r_{ij} = |\mathbf{r}_j - \mathbf{r}_i|$ is the distance between particles i and j.

The MPS Laplacian discretization of field ϕ (e.g. conduction$= T$) is given by

$$\left\langle \nabla^2 \phi \right\rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} \left[(\phi_j - \phi_i) W(r_{ij}, r_e) \right] \tag{2.61}$$

where

$$\lambda = \frac{\sum_{j \neq i} r_{ij}^2 W(r_{ij}, r_e)}{\sum_{j \neq i} W(r_{ij}, r_e)}, \tag{2.62}$$

where d is the dimension of space and n^0 is the mean number density at $t = 0$. The above form is noted to be inaccurate in conduction problems. Over-estimation of conduction is reported [51], and the stated cause is the inconsistency of λ near the boundary. This problem seems to be similar to the problem encountered in conduction using SPH.

Modified MPS for Conduction

An additional problem in applying MPS for heat conduction exists. The above Laplacian discretization does not account for any material properties other than thermal diffusivity$= 1$. No discretization method in MPS is available except for the form used in the viscosity term of the Navier-Stokes equation, which is

$$\left\langle \mu \nabla^2 \mathbf{u} \right\rangle_i = \frac{2d}{\lambda n^0} \sum_{j \neq i} \left[\mu_{ij} (\mathbf{u}_j - \mathbf{u}_i) W(r_{ij}, r_e) \right] \tag{2.63}$$

However this form cannot be used since the discretization suffers from the same problem with the definition of λ near the boundaries mentioned earlier. Also, for differing particle properties, the interface condition is approximated by the arithmetic mean. However, the correct averaging is the harmonic mean. Here the discontinuous thermal conductivity result from SPH [41],

$$\frac{2k_i k_j}{k_i + k_j} \tag{2.64}$$

is applied to the discretization of the energy equation in MPS. Using the correct discretization, the final energy equation for multi-material property using MPS is

$$\langle \nabla k \cdot \nabla \phi \rangle_i = \frac{4d}{n^0} \sum_{j \neq i} \frac{k_i k_j}{k_i + k_j} \frac{\phi_j - \phi_i}{|\boldsymbol{r}_j - \boldsymbol{r}_i|^2} W_{ij} \tag{2.65}$$

Equation 2.65 is substituted for the SPH heat equation because both forms are in the particle frame of reference. Equation 2.65 is used for this work, and work partially borrowed from MPS literature.

2.5 Phase Change

The same conduction model can be used to solve for the heat transfer during phase change. The conduction solver was extended to incorporate phase change using the scheme described in the Section 2.4. The phase change is modeled by the enthalpy method shown in Algorithm 2.1. The rate of change of enthalpy is calculated based on the temperature of particles, which is then integrated in time. Then the temperature is updated from the enthalpy value. The temperature boundary condition is applied.

Algorithm 2.1 Phase change

$$\frac{dH}{dt} \leftarrow \nabla k \cdot \nabla T + Q$$

$$H^{n+1} = H^n + \Delta t \frac{dH}{dt}$$

$$T \leftarrow (1-f)\int_{T_{ref}}^{T} \rho c_s dT + f \int_{T_{ref}}^{T} \rho c_l dT + f\rho L = H$$

$$T \leftarrow \mathit{apply\ BC}$$

2.6 Boundary Conditions

2.6.1 Fluid

Wall

A fluid particle approaching the wall boundary experiences the pressure exerted by the wall due to the increase in local density. However, the deficiency of particles near the wall boundary weakens the necessary force to contain the particles inside the domain. This often results in the particles passing through the walls or becoming embedded in walls.

There are many methods for treating solid wall boundaries and this is an area of active research [54, 55, 56, 57]. A brief overview of the boundary condition between the wall and the fluid is described here. In particular, the focus will be on repelling particles from the wall so as to model no flow across the wall. Wall boundaries can be classified into

- Force
- Ghost Particles
- Dynamic Boundary

- Hybrid

There are mainly two models of force boundaries, Lennard-Jones type [54] and Repulsion Force [58]. The repulsion by force is very effective at preventing particles from penetrating the wall. In the Ghost particle method [43], the lack of particle interaction is accommodated by adding layers of stationary particles past the wall which exist to increase particle number density. In general, the wall boundary has two additional layers of virtual particles as shown in Figure 2.7.

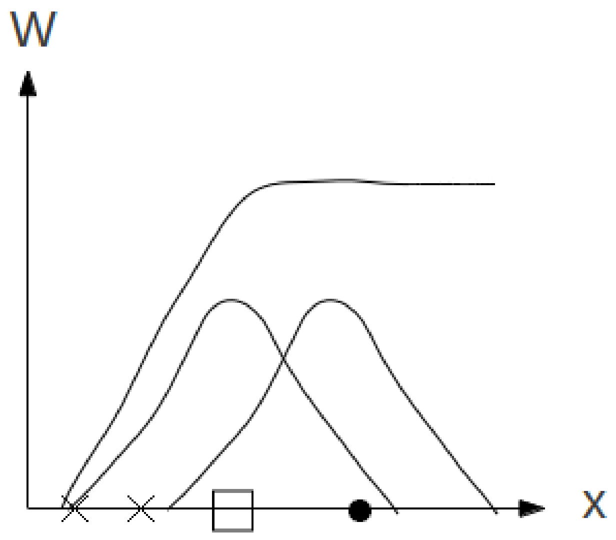

Figure 2.7: Kernels of virtual particles ($\times$), boundary wall($\Box$) and fluid particle ($\bullet$).

The Dynamic boundary condition imposes the governing fluid equations of momentum and continuity to walls [55]. Therefore the modeled boundary term in equation (2.43) is removed, resulting in equation (2.44). Finally, some combination of the above methods is a hybrid method [59]. The Dynamic boundary condition is implemented in the melt model for this dissertation.

Inflow/Outflow

Particles entering and exiting the computational domain must be explicitly tracked and controlled. In general, it is desirable to maintain some controlled mass flux (or velocity) at the inlet and conserve the governing fluid equations inside the domain. Prior applications of SPH model simple problems where this boundary is not necessary, therefore this area of research is new. Inflow and outflow are modeled for free-surface channel flow in the work by Shakibaeinia and Jin (2010) [53] and Federico et al. (2012) [60]. The basic idea is to have an inlet and outlet region. Particles are set to some regular structure and given some set velocity, density, and pressure condition in the inlet region. After it has moved some distance it is simulated as an internal fluid. The particles are purged and no longer part of the simulation as the fluid passes into some outlet region. The complexity arises in tracking how many particles of each type have moved out of the domain and implementing the appropriate calculation. The details of this algorithm are discussed in Section 3.5.4.

2.6.2 Heat

The Dirichlet condition or fixed boundary condition is directly applied to the particles by setting the value of boundary particles at every iteration. This boundary condition is used to set the inlet temperature. For discontinuities, such as the boundaries of the domains of the simulation, special attention is necessary to account for the deficiency of particles near the edge of domains [35]. The heat flux across different materials is handled without the need to explicitly determine the interface. Some regions may generate heat such as heat from fission. The source term is added to the particle in each time step in a manner similar to the Dirichlet condition.

2.7 Limitation

Apart from the previously discussed limitations of the model and the method, the current model does not take into account radiative heat transfer, turbulence and rigid-body motion of re-solidified melt material. Phase change is limited to melting and solidification due to limitations in allowable density ratio. High density ratios require modifying the interface boundary to handle non-physical mixing. This complication worsens for higher density ratios as in the case of gas against metal with the density ratio of steam over aluminum near 1/10000.

There are instabilities in the SPH method as discussed which can cause the simulation to fail. The most common instability is caused by particles clumping too close together and this is corrected by the addition of repulsion and/or using a modified kernel. In the context of this work, this instability is further complicated by the use of the dynamic boundary condition, the heat conduction, and phase change.

Chapter 3

Implementation

3.1 Ascent of GPU

The gaming industry's quest for realistic visual experience requires fast computationally rendering of graphics. This has driven rapid development of graphics hardware. The computational power of GPUs continues to rise faster than CPUs. A GPU offers high FLOPS and high bandwidth, opening up new possibilities in high performance computing (HPC). Traditionally, even a medium-sized HPC installation required a large investment. Because of the high cost, computing time on a HPC platform is out of reach for most users. On the other hand, GPUs are relatively cheap, at few hundred dollars each, and are generally found on every computer. The availability of the inexpensive GPUs, with huge computing power is opening HPC to the masses.

GPUs are distinctly different from CPUs, but both share similar components. Figure 3.1 shows a typical computer hosting a GPU device. The basic components common to both hardware are memory (orange), core (green), and control unit (yellow). The latest generation of GPUs typically have specifications as shown in Table 3.1 and Table 3.2.

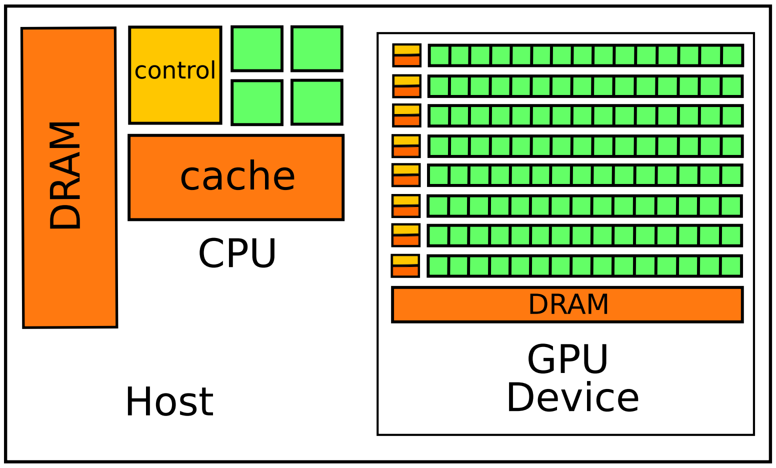

Figure 3.1: A typical CPU and GPU schematic

Table 3.1: NVIDIA GTX 680 specifications [61]

CUDA Cores	1536
Clock	1006 MHz
Bandwidth	192.2 GB/s
Memory	2/4 GB

Table 3.2: AMD Radeon 7970 specifications [62]

Streaming Processors	2048
Clock	925 MHz
Bandwidth	264 GB/s
Memory	3 GB

One of the clear differences from a CPU is that GPUs have a few hundred to thousands of "processor cores" compared to four or even eight cores on a CPU. A GPU core is relatively simple compared to a CPU core and they do not include branch prediction[1]. A CPU core is generally two times faster than a GPU, clocking at around 2 GHz or more. GPUs are designed to perform operations necessary for rendering images to be displayed on screen(s). The more demanding games require that a three-dimensional scene be transformed into

[1]Flow control instruction (if-else, switch, do, while) for threads of the same warp taking different execution paths. This divergence forces the serialization of the different paths.

pixels on a two-dimensional screen. To achieve reasonable refresh rates, GPUs must be fast. Such operations are repetitive and have independent data structures suited to a massively parallel computational hardware like the GPU.

The GPU is connected to its host by a bus called the Peripheral Component Interconnect Express (PCIe). Data transfer between the device GPU and host CPU is a high latency[2] and low bandwidth process[3]. The current, PCIe 3.0 standard has a bandwidth of 16GB/s. Recall, the internal memory bandwidth of a typical GPU is at least 10 times this (e.g. 192GB/s of GTX 680 Table 3.1). Hence, it is more efficient to minimized host-to-device data transfer by leaving data within the device. GPUs are suited for computation local to the device with low data transfer to the host.

The attractive performance of the GPU has led to attempts to perform computations other than graphics. Some of the early GPU computations were linear algebra, image processing, and SPH [63]. The specialized architecture of early GPU require expert knowledge and use a low level programming language. Many high level languages for GPU programing are now developed to ease coding for modern GPU.

3.2 CUDA

3.2.1 Introduction to CUDA

In 2007 NVIDIA released Compute Unified Device Architecture (CUDA). The accessibility of CUDA made general purpose GPU computing widespread. CUDA is a platform made of software and hardware. It is an application programming interface (API) for programing NVIDIA GPUs. The CUDA toolkit provides a high-level C-like language and a low-level

[2] The duration of time between the messaging of the instruction and execution of the instruction.

[3] Typical host-to-device data transfer is around 6 GB/s, whereas device-to-device is over 150 GB/s (as of this writing).

API for general programming. CUDA is the framework for the implementation of the model described in this work.

3.2.2 CUDA Framework

The basic unit of parallelism of CUDA is a C function called the kernel. The word kernel used in this chapter refers to the CUDA kernel and not the SPH kernel W, unless otherwise stated. Each kernel function is executed as a thread N times in parallel. In contrast, such a C function is executed once on a CPU or the number of CPU cores available. The CUDA kernels are massively parallel, often executed thousands to millions of times.

Multiple threads make a block(s) which may have up to three dimensions. Block(s) then create a grid, also with up to three dimensions. The size of blocks and grids are set by the programmer within the limits of the hardware's architecture.

Concurrency[4] between kernels is guaranteed for a group of 32 threads called a warp. A Warp is a hardware implementation and is not necessary for writing CUDA kernels, but is an important aspect of performance. For example, if the $1536 + 1$ threads are executed on the GTX 680 (Table 3.1), 1536 threads will first execute then followed by a final batch to compute the last thread. Processing the last thread with a separate batch negatively affects the computational performance.

Additional consideration is the conditional statements in a kernel, which may cause branch divergence. As an example, let there be N threads, where $N = N_A + N_B$. If all threads have the same condition then no divergence occurs. However, if at least one thread is a different condition, then all threads N_A are computed, then followed by the next condition. This branching makes an otherwise parallel execution into a serial operation. Since 32

[4]Simultaneous execution of instruction.

threads group together into a warp, a single thread branching from the rest will make the rest wait for this single thread to complete.

The GPU device has several kinds of memory, they are:

- global: main memory of a device (2/4 GB), which is accessible by all threads, but access is very slow.
- texture: cached Read-Only memory, which may be beneficial for certain uses.
- registers: very fast memory that is unaccessible to the programmer and is optimized by the compiler.
- constant: cached Read-Only memory limited to 64 KB. It can be as fast as registers if all threads access the same address.
- shared: memory that is accessible only by the kernels on the same streaming multiprocessor (SM). Shared memory has faster access than global memory.

For the device to compute data, it must be transfered by the host CPU to the global memory. This process is controlled by the host and the device can only access its own memory. A typical CUDA program may be structured like Figure 3.2.

1. The host CPU allocates memory on the GPU which is then transfered by the CPU to the GPU.
2. The CPU calls the kernel to be executed on the GPU. The data is partitioned into grid, blocks, and threads.
3. Once the GPU starts running, the CPU is free to do other work.
4. When the kernel has completed work, the result is transfered from the GPU to the CPU.

5. All allocated memory in the GPU is then freed.

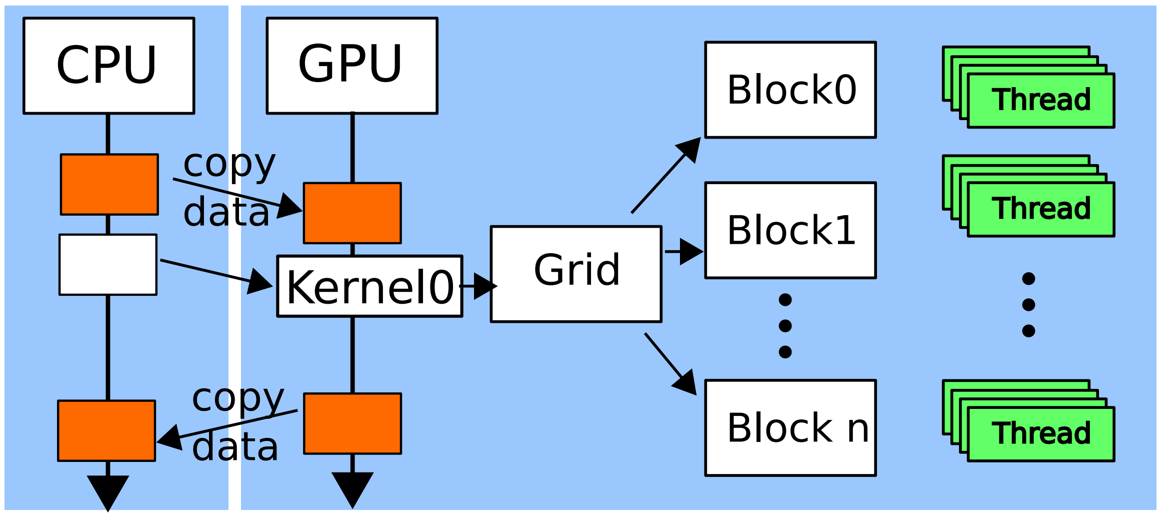

Figure 3.2: CUDA kernel execution model

Computational problems requiring complex algorithms rely on libraries for data primitives such as sorting, counting, and reduction. Such functions are commonly used and as such are available in many libraries. However, for specialized hardwares like the GPU, there are only a few libraries available.

CUDA Thrust [64] is used in this work. It is a high-level library for parallel algorithms for both CPUs and GPUs based on CUDA. The template library is like the Standard Template Library (STL) of C++. Thrust allows rapid development of complex codes that are both portable and concise. However, this high level interface to low level CUDA can cause slower execution time and higher memory usage.

3.3 SPH on GPU

SPH is computationally costly compared to mesh-based methods such as FEM for most standard fluid flow evaluations. This is due to the many calculation of interactions (i.e.

sampling) surrounding each particle. The computational cost is addressed by using a high performance parallel computing environment using Message Passing Interface (MPI)[5]. Such resources are limited, partly due to high capital cost, which has limited the use of SPH to small academic problems. The identification of GPUs as computing platforms for SPH increased interest in the method.

One of the first SPH works to exploit GPU computing used OpenGL [63]. OpenGL was not designed for generic programming but for graphics output. The release of CUDA made GPU programming more accessible. SPH on GPUs developed with CUDA showed performance gains of up to two orders of magnitude faster than CPU code [65].

In fluid dynamics, some areas of GPU SPH application are coastal wave–structure interaction [66], lava flow [67], and avalanche flow [68], with focus on real-time visual simulation. More recently, work on SPH for free-surface flow was extended from a single GPU to multiple GPUs [69]. The major SPH code at present is DualSPHysics [70, 71, 72] which is collaborative work of several university groups released under open source licenses. It allows a choice of running on CPUs using MPI or GPUs. Recently, DualSPHysics application showing a billion particles splashing onto an off-shore oil rig was published [73]. As of the writing of this work, the only SPH code with development activity appears to be DualSPHysics.

3.4 Model Algorithm

The basic SPH algorithm is an iteration with three components as described in Figure 3.3. The three components are a neighbor search, interaction calculation, and integration. All three components have subcomponents. All particle properties such as position, velocity, and temperature are restructured in the neighbor search section. Then pressure is

[5]Standard for parallel computing in distributed memory systems.

calculated based on density. With pressure known, the force acting on the particle is calculated. Heat transfer is then calculated. This step is absent for most SPH codes, which only simulate fluid flow. The final component is integrating the position and velocity using the force. The particles are prescribed by the set boundary parameters. The data is then updated for the next time step and the iteration repeats until termination.

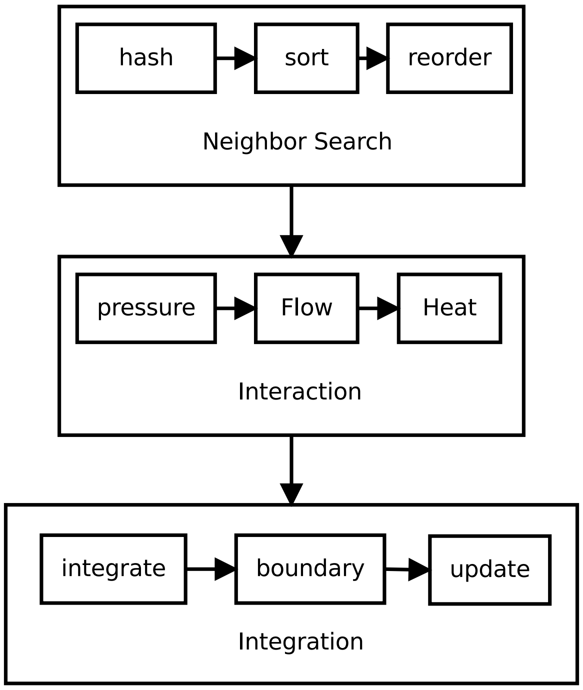

Figure 3.3: Time step

The interaction calculation of the SPH method is simple. The algorithm for the SPH method involves iterating on the three components. In contrast to the apparently simple SPH method, the neighbor search algorithm is a complex part of the implementation. A practical SPH code requires a fast and efficient neighbor search algorithm. Due to the specialized architecture of the GPUs, careful consideration must be given to neighbor search algorithms for SPH. A small number of SPH source codes are available publically. The only code with source code and documentation are the serial, parallel and GPU codes by the group SPHysics. Another SPH algorithm documented for GPUs is offered by Krog

[68]. The rest of this chapter will describe in detail the SPH algorithm for GPUs.

3.5 Model in CUDA

The fuel melting model developed in CUDA is shown in Algorithm 3.1. The three major components broadly described in Section 3.4 are expanded in the iteration section. First, the inflow boundary condition is executed. Due to the constraint of no global memory resizing and the parallelism of the algorithm, this section involves more than prescribing particles and their properties at the boundary. The data in memory of particles for inlet/outlet boundaries are reordered since they are handled differently than the fluid. The reordering makes this section more complex and the details will be shown in Section 3.5.4. Second, particles are reordered and neighboring particles are marked. Then pressure is explicitly calculated by the state equation based on density ratio. The interaction for each and every particle with its surrounding N particles is calculated. It is worth noting that the set of neighboring N particles does not include the ith particle itself since the evaluation of the distance $r_{ij} = |r_i - r_j|$ where $i = j$ is zero will corrupt the computation since r_{ij} occurs as a denominator in the SPH equations. The governing equations are then integrated for flow and heat. The change of phase is computed before integrating particle position to identify particle types. Solid particles will stay fixed in the original position, where as a fluid particle will flow. Solid particles that have melted are fluid particles and will flow. Then boundary conditions and particles are checked to ascertain if they are within the boundary. The results are occasionally copied back to host and written out as a file for that time step. Data transfer between host and device is limited since the device will wait on the transfer to complete before the computation continues. Finally, the allocated resources in the device and host are freed.

Algorithm 3.1 Detailed Fuel Melt Algorithm

```
1. initialization (check input, read simulation parameter, and material properties)
2. construct problem (position, velocity, type, temperature, ... etc.)
3. allocate memory and transfer data from host to device (GPU)
4. iterate
    (a) inflow BC, see Section 3.5.4
    (b) neighbor search (hash, sort, reorder), see Section 3.5.1
    (c) pressure $p = equation\ of\ state(\rho)$
    (d) flow $dv_i = \sum_{j \in N}$, see Section 3.5.2
        where $N = \{j \in \forall \text{ neighbor}, i \neq j\}$
    (e) energy $dh = \sum_{j \in N}$, see Section 3.5.2
    (f) integrate energy
    (g) calculate phase change
    (h) integrate flow
    (i) apply boundary conditions, check
   transfer results from device (GPU) to host (occasionally)
   Output results from host
5. clean up memory on both host and device
```

Data analysis and visualization is post-processed using the ParaView software [74]. ParaView is an open source scientific visualization software with an interactive graphical user interface. ParaView reads many data formats, many of which are CFD related. ParaView is built on Visualization Toolkit (VTK) for data processing and graphics rendering. Therefore it naturally follows to adopt a data file format compatible within this framework. VTK has many file formats for different data structures categorized into structured, unstructured, serial, and parallel for Extensible Markup Language (XML) formats and also legacy formats [75].

VTK's API is not used to write data; instead, the data is written out by the code directly. This code was written to reduce the library dependency necessary for execution. For this reason the data format uses the legacy VTK file format for unstructured data for its relatively simple file structure. Large simulation results can create large files and if written out in ASCII, it can be an order of magnitude larger than binary.

3.5.1 Neighbor Search

The number of neighboring particles is finite since the kernel W has compact support[6]. Figure 3.4 shows particles in a domain with the circle of radius r representing the compact support of the particle at the center, which will be called particle i. Looking at Figure 3.4, it is obvious which particles are inside the radius r of particle i. However, it is not immediately clear how to efficiently determine these particles given the list of coordinates for all the particles. The problem of determining local neighbors is a common problem in many fields and is referred to as nearest neighbor search. There are many neighbor search algorithms used for different applications. There are a few common algorithms used in SPH.

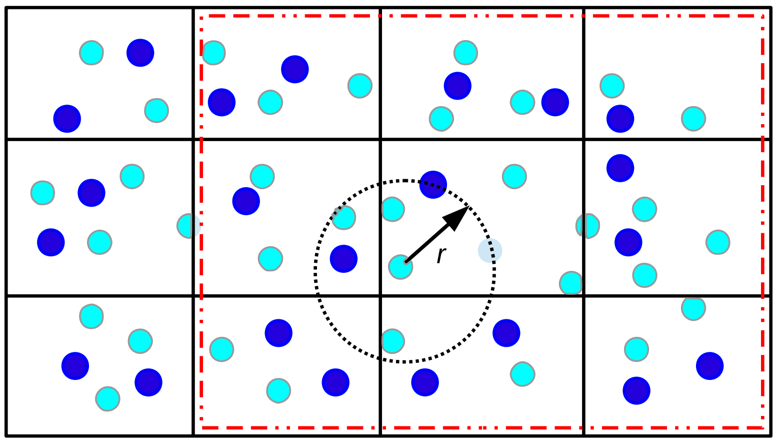

Figure 3.4: Neighbor search in 2D

[6]See equation (2.5)

One method is to check, given the positions of particle i and particle j , whether particle j is

$$\begin{cases} \text{inside} & \text{if } r_{ij} \leqq r_e \\ \text{outside} & \text{if } r_{ij} > r_e \end{cases} \tag{3.1}$$

where the distance between particles is $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$. This naive algorithm requires that every particle checks every other particle. The computational complexity of this approach is of order $\mathcal{O}(n^2)$ where n is the number of particles. For a large number of particles, this all-pair search algorithm would be computationally very expensive.

There are a few algorithms to better address this neighbor search problem. One method is to use a linked list[7] [76]. For example, Figure 3.5 shows three linked lists. A simple linked list is a pair of data and reference "pointing" to the next pair. For a neighbor list, this data structure can hold the index to the array with particle data. The reference points to the next index value of the neighbor particle. This list continues until no neighboring particles exist. Each particle will have its own linked list. In this approach, the linked list must be constructed and updated as particles move, changing the neighboring particles. Consequently, the changing number of neighboring particles means the length of the list also changes.

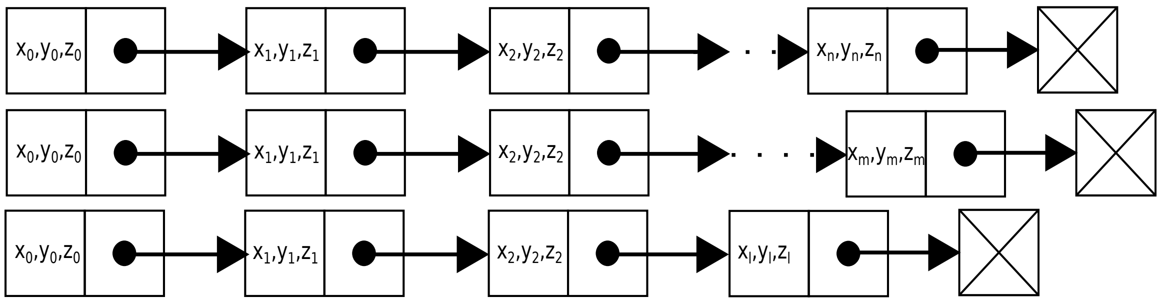

Figure 3.5: Singly linked list

[7] A sequence of elements with a data structure containing the data and the address of the next element.

Instead of uniformly partitioning the domain into equal cells as in Figure 3.4 or with linked lists, an adaptive hierarchical tree is another approach that has been used to identify neighboring particles [77]. Figure 3.6 (left) shows a domain of nine particles divided into quadrants of different levels. Figure 3.6 (right) shows the tree structure of representing the hierarchy. The first level divides the domain into two particles four and five, and two quadrants each with more levels. In the top left quadrant, particle i is surrounded by particles two, six, seven, and eight. The list of possible neighbor particles is obtained by going up the tree. In the case of this example for particle i, these particles are $8, (2, 6, 7), (4, 5)$. Hence, particles 1 and 3 cannot be neighbors.

The hierarchical nature of a tree search is suited for variable smoothing length [43]. The different levels of tree allow different smoothing lengths to be used. In the tree search algorithm described above the domain was recursively split; repeatedly subdividing the divisions. This algorithm adapts the computation to regions where particle concentration is high, improving the computational efficiency. A tree search is computationally efficient for a large number of particles where the algorithm is of order $\mathcal{O}(n \log n)$ [77].

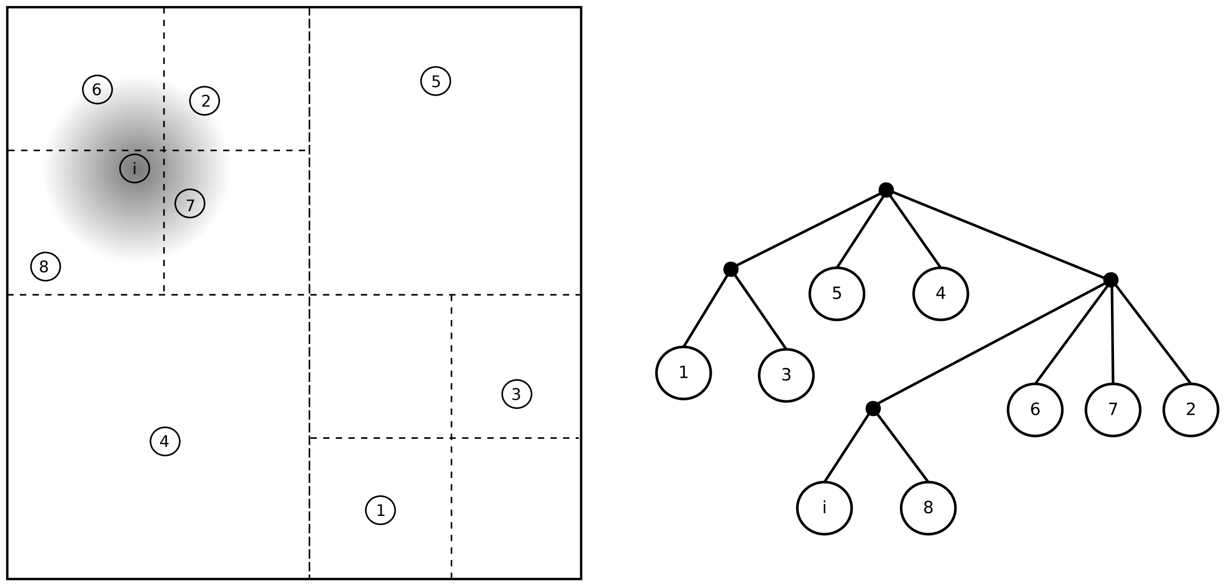

Figure 3.6: Tree Search

The two algorithms, linked list and tree search algorithms, work well in CPU platforms. However, without significant development the use of these algorithms is not practical for GPU architecture. In the case of a linked list, the changing number of particles means the list size also changes. This is problematic since it corresponds to resizing the global memory on the GPU device, which only the host CPU can perform. It is a time consuming process for the GPU to resize its memory.

In the case of a tree search, the same issue of memory resizing occurs. A Tree branches into many levels with each level having its own branch. As discussed earlier, on GPUs divergence by a conditional if-else statement is expensive. A tree search will have many branches that require the evaluation of this conditional expression.

There is a more simple approach, which is the use of hash functions[8] to map particles. A hash algorithm [78] is used in this work. Recall, Figure 3.4 shows two different particles inside a domain composed of 12 cells in 2D. The cell with the particle encircled by a dotted ring has eight neighboring cells marked by the red line. The domain is divided into equally spaced cells. Then the position of any particle within the domain is uniquely identified to a cell by

$$id_{x,y,z} = \left\lfloor \frac{x - x_0}{cellsize_x}, \frac{y - y_0}{cellsize_y}, \frac{z - z_0}{cellsize_z} \right\rfloor \tag{3.2}$$

where x, y, z are the coordinate values of the particle and the subscript 0 designates the reference point. For a 2D case, e.g. $z = 0$, the dimension is not used. It is clear that all particles in the same cell will have the same *id*. To assign a unique value to each particle a hash function is used. A Hash function may use large prime numbers. The hash function that is used in this work is

[8]An algorithm for mapping data(s) to constant data. For example, in SPH, given the coordinates (x, y, z) a direct relationship in the form, $u \leftarrow f(x, y, z)$ can be determined.

$$id_z \cdot cellsize_x \cdot cellsize_y + id_y \cdot cellsize_x + id_x \tag{3.3}$$

The neighbor search algorithm for this work is Algorithm 3.2.

Algorithm 3.2 Neighbor Search

```
1. Hash by equation (3.2) and equation (3.3)
2. sort index by hash value using radix sort
3. reorder data by index
```

For sorting, radix sort[9] from the CUDA Thrust library is used. Any sorting algorithm can be used. Radix sort is chosen because it is fast, having a worst case performance of $\mathcal{O}(kn)$, where k is some constant and n is the number of items. The hash value is sorted, which is paired to an index. Using this index all data structure (e.g. temperature, velocity, density...) is reordered. Consequently, the data structure is consecutive in the memory address. Now the data structure in memory is localized to the actual position of particles in the problem. Finding all the neighboring particles is simply a matter of iterating through all the cells and all the particles inside each cell.

3.5.2 Particle Interaction

All interaction calculations share the same basic Algorithm 3.3, shown below. The algorithm starts by fetching particle i's position and any material property data. Then the algorithm iterates through all cells surrounding the cell of the ith particle. In each cell, any particle inside the cell is checked to determine if the distance from this particle to particle i is less than a distance r_e. If the particle is inside the smoothing length, then the interaction

[9]An sorting algorithm that groups by least/most significant digit, repeating this step for the next significant digit. For details see `http://en.wikipedia.org/wiki/Radix_sort`

term is calculated. This operation is done for all particles in this cell and all particles inside all the other neighboring cells.

In CUDA, each thread computes the interaction summation on particle i due to all its neighboring particles j that are within the support radius. From an optimization point of view, the symmetry of the interaction between two particles can be exploited. However, in this work the symmetry is not adopted because doing so increases algorithmic complexity for GPU platforms. Additionally, the read-write operation to the global memory address on a GPU is a few hundred clock cycles, whereas arithmetic operations to compute the interaction term for a particle is at least an order of magnitude lower.

Algorithm 3.3 Interaction per thread

```
Input: particle $i$ and its data
  • For all cells between $[(x-1, x+1), (y-1, y+1), (z-1, z+1)]$
      – For all particles in this cell
          1. get data, e.g. position of particle $j$, $r_j$
          2. calculate distance $r_{ij}$
          3. if $r_{ij} \leq r_e$, calculate contribution
Output: sum of contribution
```

3.5.3 Time stepping

The governing partial differential equations are reduced to ordinary differential equations in SPH. Various integration methods have been proposed. This work uses the Verlet integration scheme [79]. The derivative of velocity by the central difference method is

$$F^n = \frac{v^{n+1} - v^{n-1}}{2\Delta t} \tag{3.4}$$

where F is the acceleration of the particle. Then, the future velocity is

$$v^{n+1} = v^{n-1} + 2\Delta t F^n \tag{3.5}$$

To obtain the position of the particle, the central difference method is used again. The second derivative of position, which is the acceleration, is

$$\frac{d^2 r}{dt^2} = \frac{r^{n+1} + r^{n-1} - 2r^n}{\Delta t^2}, \tag{3.6}$$

which rewritten in terms of the future position is

$$r^{n+1} = 2r^n - r^{n-1} + \Delta t^2 F \tag{3.7}$$

The Taylor series expansion of r^{n-1} is

$$r^{n-1} = r^n - \Delta t v^n + \frac{\Delta t^2}{2} F \tag{3.8}$$

Substituting equation (3.8) into equation (3.7), the position of the particle is updated by

$$r^{n+1} = r^n + \Delta t v^n + \frac{1}{2} \Delta t^2 F^n \tag{3.9}$$

Both equation (3.10) and equation (3.11) are obtained in a similar manner as the velocity. The density of the particle effects pressure, which in turn effects the particle flow. The density is updated by

$$\rho^{n+1} = \rho^{n-1} + 2\Delta t D^n \tag{3.10}$$

where $D = \dot{\rho}$ is the rate of change of density.

$$h^{n+1} = h^{n-1} + 2\Delta t H^n \tag{3.11}$$

where $H = \dot{h}$ is the rate of change of enthalpy.

To prohibit the integration from diverging due to the uncoupled equations, it is suggested [71] that every n ($n \approx 30$) time steps, that the integration be

$$v^{n+1} = v^{n-1} + \Delta t F^n \tag{3.12}$$

$$\rho^{n+1} = \rho^{n-1} + \Delta t D^n \tag{3.13}$$

$$h^{n+1} = h^{n-1} + \Delta t H^n \tag{3.14}$$

The equations (3.12), (3.13), and (3.14) are optional in the code. In general, enabling this option results in longer stable simulations.

3.5.4 Boundary

Dynamic Boundary

The Dynamic boundary condition is used to maintain separation between the fluid and the wall particles. Algorithm 3.4 describes the method. Particle separation occurs due the change in density, which then effects pressure and ultimately influences the position of the particle through the implementation of the Navier Stokes momentum balance in SPH. For example, a particle approaching a wall will experience a repulsive force due to the increase in apparent density as the distance between the particles diminishes. This is because the SPH kernel has more weight with diminishing distance. The increase in density from the reference density corresponds to an increase in pressure. This pressure acts on both particles, exerting a repulsive force between the particles, thus maintaining particle and wall separation. Unlike fluid particles, wall particles are not moved, forcing only the fluid particles to update position.

Algorithm 3.4 Dynamic Boundary

$$P_{all} \leftarrow \rho$$

$$\frac{dv}{dt}_{fluid} \leftarrow \nabla P_{all} + \Pi_{fluid} + \mathbf{g}$$

$$\frac{d\rho}{dt}_{all} \leftarrow all$$

$$update : \begin{cases} \mathbf{v}, \mathbf{r} & if\ fluid \\ unchanged\, \mathbf{v}, \mathbf{r} & if\ boundary \\ \rho & \forall particles \end{cases}$$

Inflow and Outflow

Particles flow in from the top and flow out the bottom in Figure 3.7. The different colored sections represent the inlet, normal region, outlet, fixed walls, and particles not part of the simulation. In this example, particles in each section share the same identification tag except those in the “normal” region, which can contain material that depends on the particles. Particles crossing into a threshold are considered to have exited the outlet and are no longer available to interact with other particles. The particles at the bottom of Figure 3.7 are fixed, these particles exert pressure on to the particles flowing out, ensuring that the outgoing particles do not free fall under gravity.

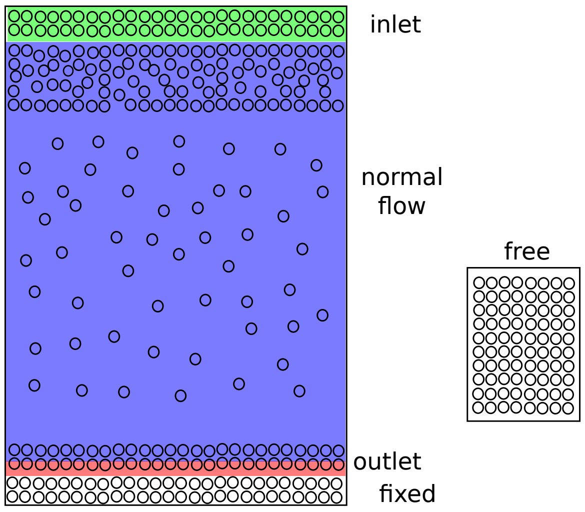

Figure 3.7: Inflow and outflow boundary

Particles that have flowed out are buffered in memory until they are reassigned as inflow particles. Let N be the total number of particles in a domain of which f is the number of free particles not presently part of the simulation and n is the number of particles at the inlet. The number of particles is such that $N \gg f \geq n$.

For a single thread running on a CPU this problem is straightforward, looping through an array to set inlet particles until n particles are set. In contrast, this approach is very inefficient for GPU environment. Here a new method for inlet boundary condition is presented. Algorithm 3.5 is the scheme for applying inlet boundary condition for GPUs in parallel without global memory resizing.

The basic idea is to remove particles at the outlet and move them to the inlet with the proper properties set. Before new inlet particles can be created two conditions must be met, they are

- number of particles that have exited the outlet is greater than or equal to the number of particles required for the inlet
- time has passed $t_{newInflow} = d_{inlet}/U_{inlet} \geq elapsed\ time$

The first condition ensures that the number of free particles at least matches the number of particles required to create new inlet particles. The second condition ensures that the new inlet particles being created are not placed too close to existing particles.

Algorithm 3.5 Inflow

```
1. save a copy of particle type $pt \leftarrow pt^0$
2. create sequence $s = \{0, 1, 2, ..., N-1\}$
3. stable sort by key (radix sort) the particle type $pt$ with sequence $s$
4. fill array of size $N$ with tag, $index \leftarrow tag$
5. alter $index = \{0, 1, 2, ..., n_{inlet}, tag, ... tag_{N-1}\}$, where $n$ is the number of inlet particles
6. stable sort by sequence $s$ (key) with $index$
7. replace position, velocity, temperature, enthalpy, density for inlet particle data using
   $index$
8. copy back particle type $pt^0 \leftarrow pt$
```

First, it is necessary to preserve the original particle type since the ordering will be lost in the sorting. Then a sequence having the size of the total number of particles N is created. This array is the ordering of the array in memory and will be used to restore to the original ordering. Next, the particle types are paired with the sequence s. This forms a key-value pair, where the key is the particle type and the value is the sequence. This pair is sorted by particle type (key) and the sequence s is also reordered to match the original pairing. The parallel sorting function, the function for creating the sequence, and the function for filling an array is from the CUDA Thrust library. For this example, free particles f have pt value less than normal particles. The sorted arrays pt' are now ascending

$$pt' = \{pt_0 < pt_1 < \cdots < pt_{N-1}\} \tag{3.15}$$

The free particles now are at the beginning of the array. The sorted sequence s' corresponds to pt' such that

$$\{(pt_0, s_0), (pt_1, s_1), (pt_2, s_2),\ \dots\ , (pt_{N-1}, s_{N-1})\} \tag{3.16}$$

An array index of size N is created all with the same value of tag. The first n values are assigned a new sequential value. For instance, this array will be

$$index = \{0, 1, 2,\ \dots\ , n_{inlet},\ tag, tag,\ \dots\ , tag_{N-1}\} \tag{3.17}$$

To obtain the particles which are inlet particles a final sort is done on s paired with $index$. Since s was ordering sequentially, the new sorted array s^{final} is reordered back to its original sequence along with $index$. Consequently, $index$ with values other than tag are the particles set to be inlet particles. Finally, using $index$, all data for the identified free particles are set to inlet data using the tag. Lastly, the original copy of the particle type is restored.

Chapter 4

Model Test Cases

In this chapter the implemented model is tested. First the flow model is investigated, comparing it to the analytical solution of Poiseuille flow. Secondly, the heat transfer model is applied to a thermal diffusion problem and the numerical solution is compared to the exact solution. SPH solutions are compared for different particle spacings. Lastly, a phase change problem is described and solved by the phase change model implemented in SPH. The numerical solution is also compared with the analytical solution.

4.1 Flow

An incompressible fluid flows between two infinitively long parallel plates as represented in Figure 4.1. Initially the fluid is at rest and at time $t > 0$, experiences a body force F. The walls are distanced L apart and stationary.

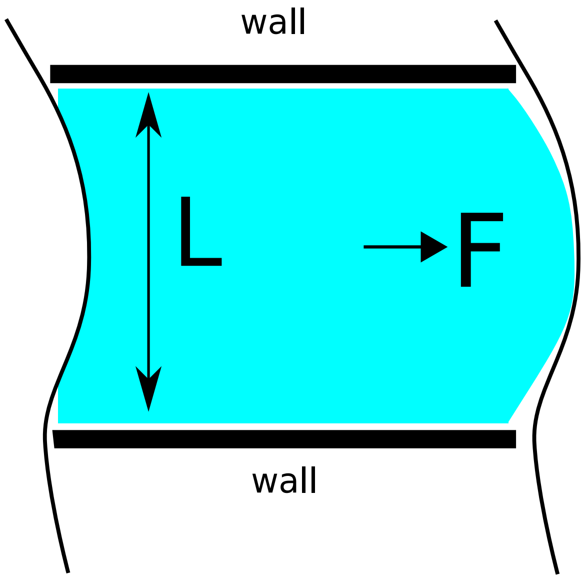

Figure 4.1: Poiseuille flow problem.

The exact solution to this unsteady Poiseuille flow is [80]

$$\begin{aligned} u_x(y,t) &= \frac{F}{2\nu}y(y-L) \\ &+ \sum_{n=0}^{\infty}\frac{4FL^2}{\nu\pi^3(2n+1)^3} sin\left(\pi\frac{y}{L}(2n+1)\right) exp\left(-\nu\left(\pi\frac{2n+1}{L}\right)^2 t\right) \end{aligned} \tag{4.1}$$

where F is a constant acceleration, L is the channel thickness, and ν is the kinematic viscosity. All the particles are subject to the acceleration F. Here, u_x is the horizontal velocity, parallel to the wall, as a function of the distance perpendicular to the wall. The parameters for this test are shown in Table 4.1. Based on the parameters the asymptotic flow is laminar with a Reynolds number $Re = u_0L/\nu = 45.125$.

Table 4.1: Flow parameters

L [m]	0.19
F [m/s^2]	9.8 (1G)
ν [m^2/s]	0.008
particle spacing [m]	0.02
time step [s]	2×10^{-5}

The x(horizontal) component of the velocity field is shown in Figure 4.2 for $t = 0.2$. There are about 40 fluid particles in a staggered arrangement between the walls. The field is constructed by Delaunay triangulation based on the values of the particles. The wall particles are in dark blue and the fluid velocity gradually increases towards the center.

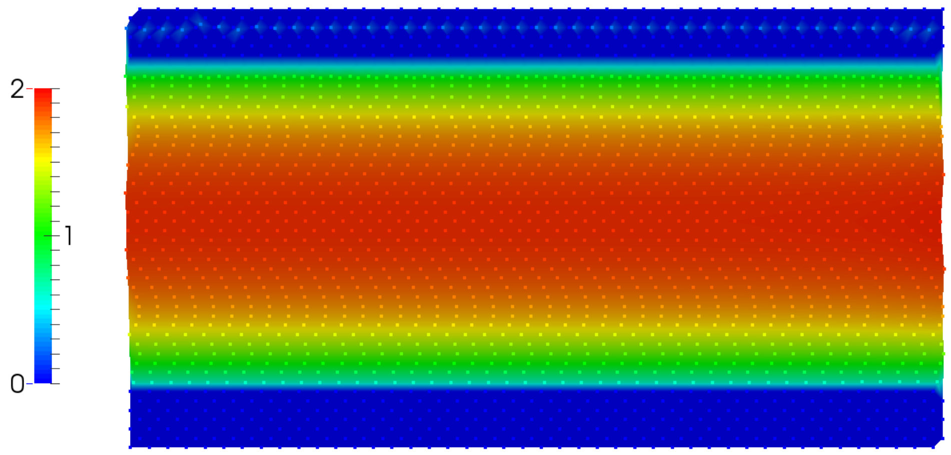

Figure 4.2: Velocity Field at $t = 0.2$.

Figure 4.3 shows the x(horizontal) component of the velocity as a function of y(vertical) position. These velocity profiles are extracted from the same simulation as Figure 4.2. The circles represent the model results and the solid lines are exact solutions of equation (4.1). Because the velocity profile was interpolated from the Delaunay triangulation of velocity, the circles do not exactly represent the 0.02 particle spacing. The results are color by time for $t = 0.01$, 0.05, 0.1, and 0.2.

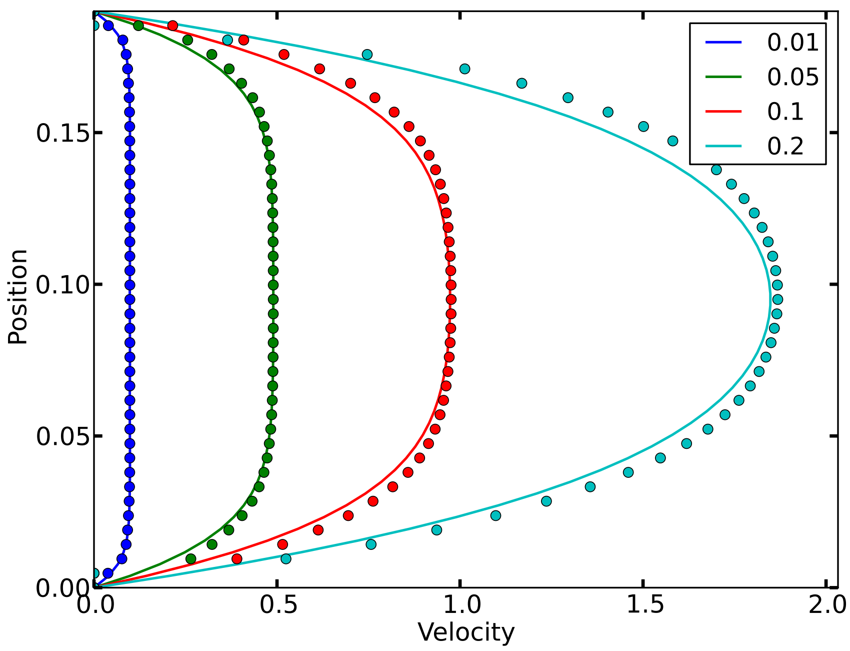

Figure 4.3: Velocity profile of the Poiseuille flow

4.2 Heat Transfer

Conductive heat transfer under the assumption of constant properties is described by the governing equation

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + Q \tag{4.2}$$

where T is the temperature, α is the thermal diffusivity, and Q is the source term. For the purpose of this comparison to an analytical solution $Q = 0$. Given an initial condition

$$T(t=0) = sin(a\pi x)sin(b\pi y)sin(c\pi z) \tag{4.3}$$

with the boundary condition along the surface $\partial\Omega$ as

$$T = 0, \quad \boldsymbol{x} \text{ on } \partial\Omega \tag{4.4}$$

Then the exact solution is

$$T = e^{-\lambda t} sin(a\pi x)sin(b\pi y)sin(c\pi z) \tag{4.5}$$

where

$$\lambda = \alpha\pi^2(a^2 + b^2 + c^2) \tag{4.6}$$

The numerical solver by design was developed for three dimension, but is capable of solving lower dimensions. To test the full capability, a three dimensional problem is solved. The Table 4.2 shows all the parameters used for the test problem. It is a unit cube with particle spacing of 0.05 along all axes.

Table 4.2: Conduction Parameters

sides: a,b,c [m]	1
thermal diffusivity [m^2/s]	0.038198983
particle spacing [m]	0.05
time step [s]	5×10^{-5}

The results are shown in Figure 4.4 and Figure 4.5. The Figure 4.4 shows a unit cube showing the particles at the center section and a plane cut along the the x-y plane where $z = 0.5$ at time $t = 1.24985$. The spacing of the major and minor tick marks along the axes are automatically set by ParaView and do not correspond to the particle spacing of the simulation parameter. The plane surface is created based by Delaunay triangulation using the particle result to help visualize the solution field. The Figure 4.5 shows the solver results at $x = z = 0.5$, $y \in (0, 1)$ and the exact solution. The solver results are the values of the particles (i.e. not the interpolated values from the Delaunay triangulation).

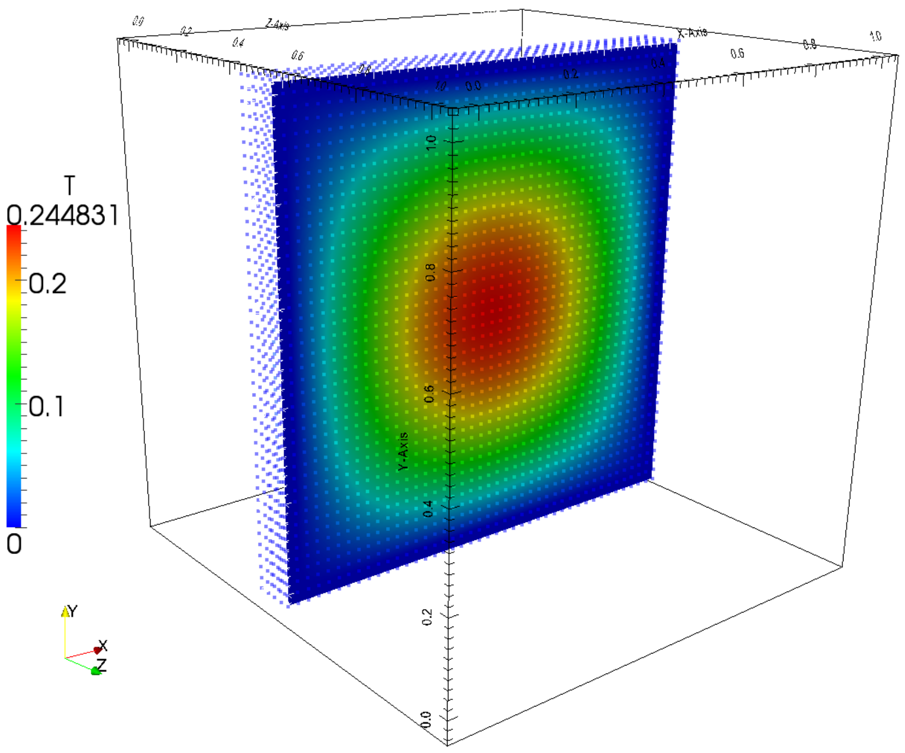

Figure 4.4: Internal slice of 3D unit cube.

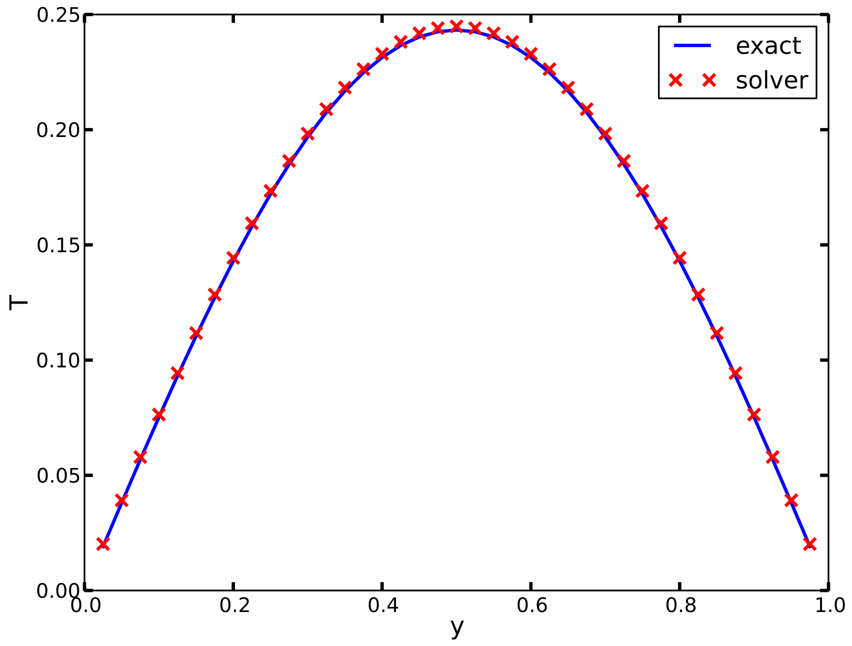

Figure 4.5: Comparison of exact and simulation result along y-axis.

Tests on the effect of particle spacing were done using all the same conditions as described except each having a different particle spacing. The results of these tests are all consistent with Figure 4.4 and Figure 4.5. These results can be seen in Appendix B.

The error of various particle spacing is shown in Figure 4.6. The same problem described above is solved. The L_2 error norm is used as a measure of the error, where the error is defined as

$$\|error\| = \sqrt{\sum_{k}^{N} e_k^2} \tag{4.7}$$

where error $e = exact - approximate$. The Figure 4.6 shows monotonically improving accuracy as resolution increases. For the coarsest particle spacing, the error is higher than the trend due to the effect of the boundary condition for particle methods.

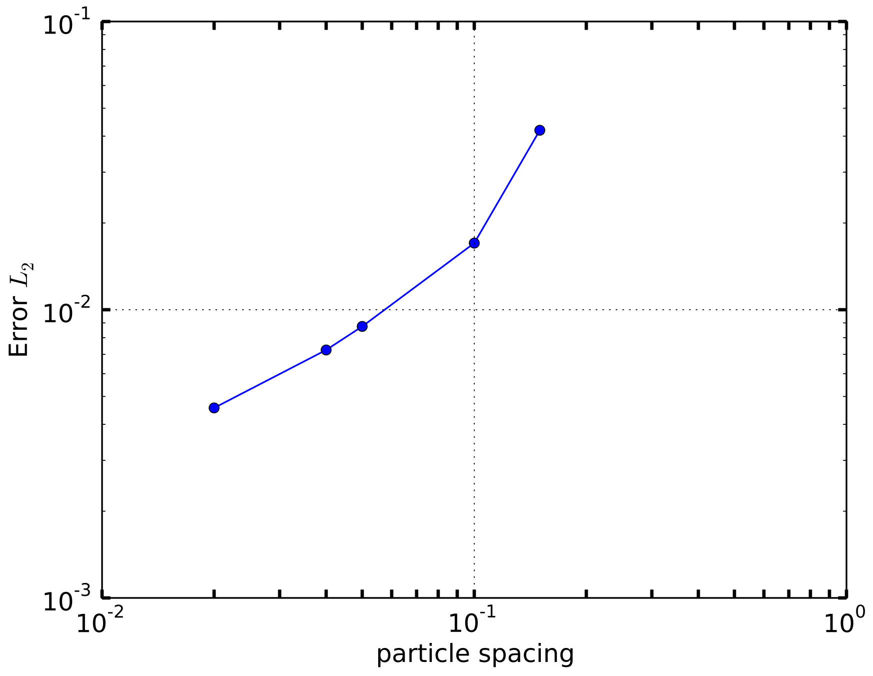

Figure 4.6: Error of conduction model.

4.3 Phase Change

Analytical solutions to phase change problems are few and limited to simple conditions. Figure 4.7 shows a phase change problem in an infinite domain with homogeneous material with no temperature dependent properties. This Stefan problem is a one dimensional phase change problem with the medium initially at T_0. The medium changes phase at T_m for $t > 0$.

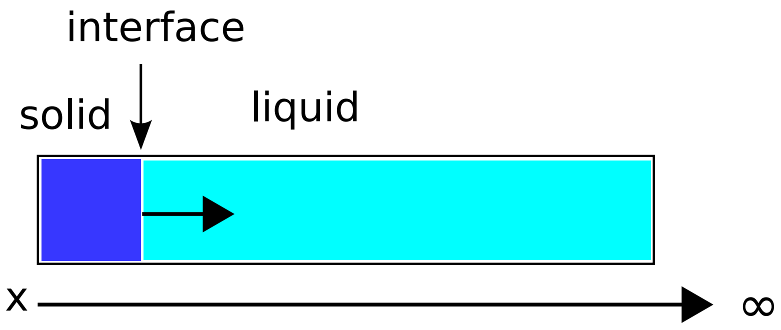

Figure 4.7: The freezing of liquid in the Stefan problem.

Initially the entire domain is liquid at a temperature $T > T_m$. At time $t > 0$ the temperature at the origin is set to $T(t > 0) = T_a$, where $T_a < T_m$. As time progresses, the liquid starts to freeze and grows, turning more liquid into solid. The heat transfer in the Stefan problem is only by conduction and assumes the interface is sharp. The interface condition is

$$k_1 \left(\frac{dT}{dx}\right)_1 - k_2 \left(\frac{dT}{dx}\right)_2 = \rho L \frac{dX}{dt} \tag{4.8}$$

where k is the thermal conductivity, T temperature, ρ density, L latent heat and dX/dt is the speed of the interface. This interface condition is also known as the Stefan condition.

The exact solution is [81]

$$T(x) = \begin{cases} T_B + \dfrac{T_m - T_B}{erf(\lambda)} erf(\eta) & \text{solid} \\ T_0 + \dfrac{T_m - T_0}{erfc(\lambda)} erfc(\eta) & \text{liquid} \end{cases} \tag{4.9}$$

where

$$\eta = x\sqrt{\rho c_p / 4kt} \tag{4.10}$$

The interface position is,

$$x = 2\lambda\sqrt{kt/\rho c_p} \tag{4.11}$$

The parameter λ is given by

$$T_m - T_B = erf(\lambda)e^{\lambda^2}\left(\frac{\lambda L\sqrt{\pi}}{C} + \frac{(T_0 - T_m)e^{-\lambda^2}}{erfc(\lambda)}\right) \tag{4.12}$$

Instead of solving the nonlinear equation for λ, variables T_0, T_m, λ, L, C are specified and solved for T_B instead. The boundary conditions for the numerical scheme are fixed with $T(x=0,t) = T_B$and $T(x=1,t) = T_0$ for the duration of the simulation. The other variables are $T_0 = 1.2,\ T_m = 1,\ \lambda = 0.5,\ L = 1, C = 1$ which makes $T_B = 0.1906$.

Table 4.3: Phase change parameters

dimension [m]	1.0x1.0
thermal diffusivity [m^2/s]	1.0
particle spacing [m]	0.04
time step [s]	2×10^{-5}

The analytical solution to the Stefan problem is one dimensional. This solution is for a medium of infinite length. The numerical test is done on a unit length with the right side having $T_0 = 1.2$ for a small finite time such that position of the interface is far from the right boundary.

In Figure 4.8, the solution to the Stefan problem is tested in two dimensions. The points on Figure 4.8 are the values of the particles and the background field was constructed to show the values in between particles at time 0.04. The interpolated background field was produced using a Delaunay triangulation based on the unit square of 25 by 25 particles. The color along the vertical y-axis are the same and only vary along the horizontal x-axis. The solution demonstrates the proper functioning of the solver.

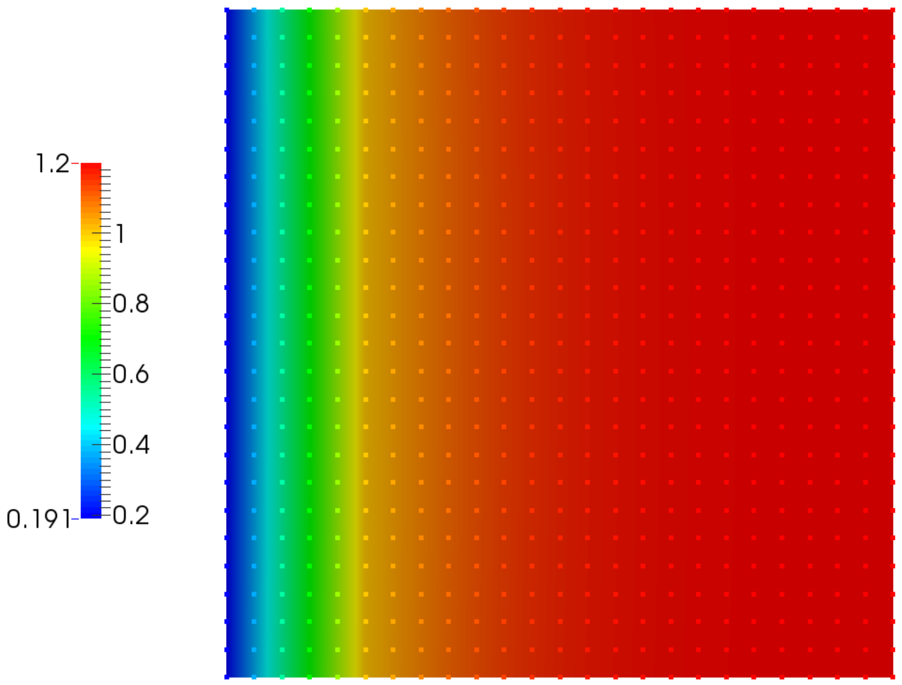

Figure 4.8: Exact and numerical solution to Stefan problem.

The Figure 4.9 shows the exact and the numerical solution along the center (i.e. $y = 0.5$) of the solution shown in Figure 4.8. The lines are exact solutions and the cross x is the fixed particle. The empty ◯ represents the exact location of the solid-liquid interface at $x \sim 0.2$.

The interface is particularly challenging as it is a abrupt transition point requiring more nodes to accurately resolve. This solver shows no noticeable deviation from the exact solution, including at the interface.

As a part of the comparison other particle spacings were tested for the same conditions described. The results of the other resolutions are shown in Appendix C. All results are consistent with Figure 4.8 and Figure 4.9. The error response to the resolution is presented in Figure 4.10, following the same procedure as presented in the heat transfer comparisons. The L_2 error used in Figure 4.10 was presented in equation (4.7).

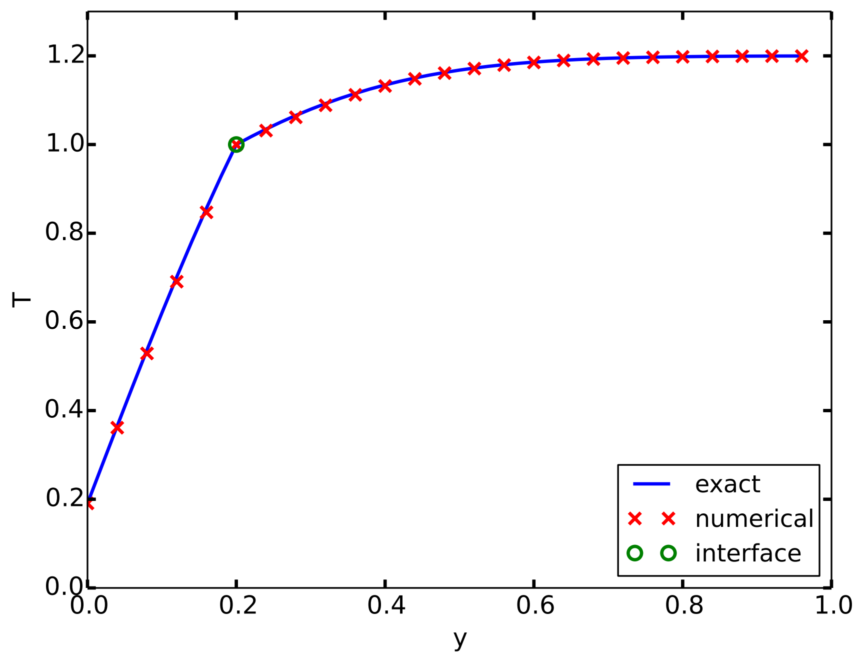

Figure 4.9: Exact and numerical solution to Stefan problem along the center (y=0.5).

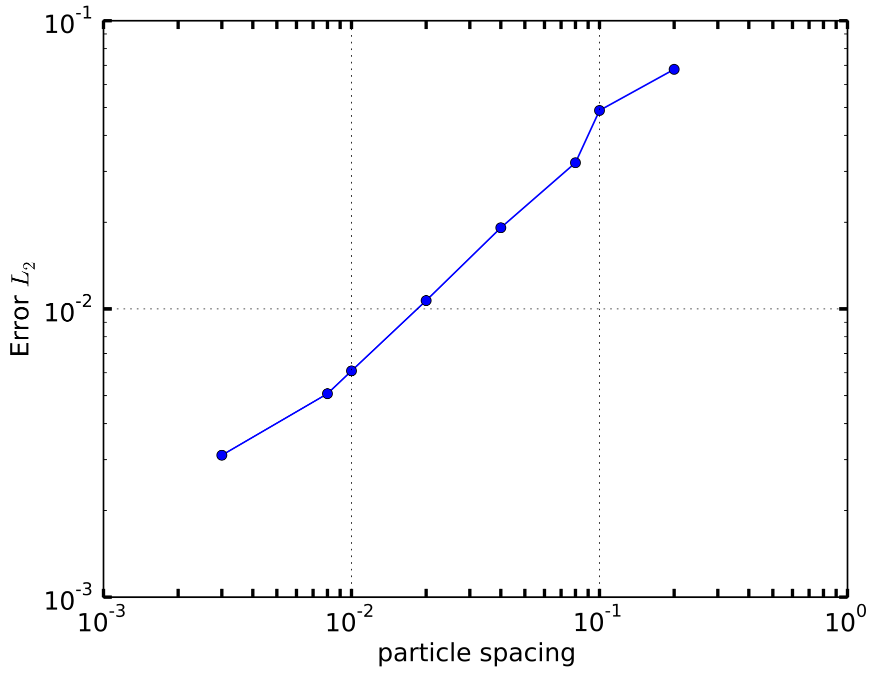

Figure 4.10: Phase Change Error

Chapter 5

Fuel Plate Melting

5.1 Model Scaling

HFIR fuel plate has a thickness of length of 0.050 in. (1.27 mm) with a length of 20 in. (50.8 cm) of active fuel and an arc length (width) of 3.3005 in. (83.8 mm) for the inner fuel plate and 2.944 in. (74.8 mm) for the outer fuel plate. The smallest scale is the thickness which determines the minimum practical particle spacing for the simulation. For a problem with large difference in length scale, the smallest length imposes a large cost to the simulation. For example, placing 20 particles across the combined thickness of a fuel plate and coolant channel will span 0.1 inches. At 20 particles per 0.1 in., there will be 4000 particles along the length of the active fuel, and 1780 along the width. In total there are 142.4 million particles in the domain, and this is for the coarsest particle spacing likely to return accurate results. An additional consideration is the dependence of the time step on particle spacing. The time step for this example is of order 10^{-7}. Increasing the number of particles further decreases the time step. These effects compound to make a direct SPH simulation approach to the HFIR fuel geometry not practical with the available computing resource.

The problem of scale is not unique to particle methods or even to computational methods in general. Similar issues arise when designing large scale aircraft or commercial nuclear power plants. To study the system response of such a structure and components scaled down experiments are performed. These studies allow for modeling based on scaling and similarity.

Scaling is applied to the HFIR fuel melting simulation to reduce the computing time while preserving the relative importance of phenomena of interest. The short computational time of the scale problem comes with the benefit of running multiple cases with different parameters to study their effects. This work adopts a scaling following Zuber et al. [82] where the scaling is based on time τ. The time ratio is

$$\Pi_i = \frac{\tau_{cv}}{\tau_i} = \omega \tau_{cv} \tag{5.1}$$

where τ_{cv} is the system response time in the control volume and ω is the frequency of the process. The main interest of this study is the heat up and subsequent melt migration. Using the scaling, the characteristic time ratio to preserve is

$$\Pi = \frac{q''A}{\rho H Q} \tag{5.2}$$

where q'' is the heat flux, A the heat transfer area, Q the volumetric flow rate, and ρH the enthalpy per unit volume. Relating the model and the actual values is

$$\left[\frac{q''A}{\rho H V}\tau\right]_m = \left[\frac{q''A}{\rho H V}\tau\right]_a \tag{5.3}$$

In preserving time and using the same material, e.g. aluminum, the above equation (5.3) reduces to

$$\left[\frac{q''A}{V}\right]_m = \left[\frac{q''A}{V}\right]_a \tag{5.4}$$

$$= \frac{Power}{volume} \tag{5.5}$$

Equation 5.4 is the power density. All the terms are known for the HFIR core, which is $q''/width = 9.594 \times 10^9$ W/m^3 [83], where width is taken to be the minimum channel width of 40 mils (0.001016 m). Since modeling is done in two dimensions the length (radial) of the fuel plate of 5.5 in. (0.1397 m) is taken into account. This then becomes 1.34 MW/L for the two dimensional case. If the power density is taken to be 6×10^9W/m^3 [84], then the power density for a 85 MW power is 0.711 MW/L.

In Figure 5.2, the relative power density distribution used in HFIR is shown [85]. This is a radial cross section view of the inner and outer parts of the HFIR core. The original data has 27 by 15 data points respectively along the vertical axis and radially. A quadratic smoothing is applied in this figure to improve the visual quality. This work uses the inner side of the outer section as the profile for the two dimensional simulation model.

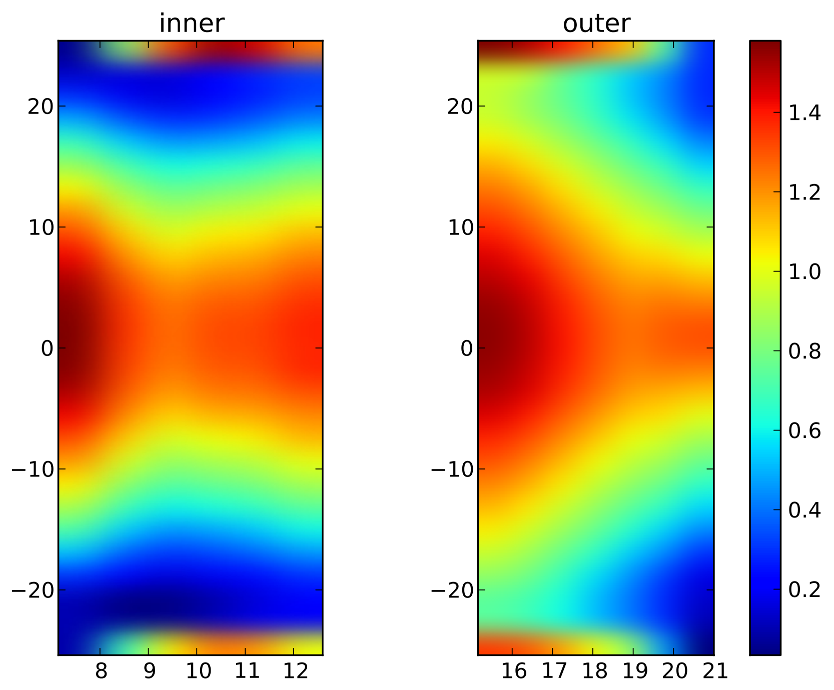

Figure 5.1: Relative Power density distribution in the HFIR fuel annuli.

The relative power density at 15.15 cm radially from center of core will be used in this model. To improve the number of data points for use in the model we seek a continuous function for interpolation of the relative power density. A polynomial fit of 16 order is created using a least-square fit based on the 27 data points. The polynomial function is

$$\begin{aligned} f &= 8590814.17324x^{16} - 66659465.4541x^{15} + 235678601.952x^{14} - 503012634.021x^{13} \\ &+ 723896851.063x^{12} - 743217280.46x^{11} + 561816951.651x^{10} - 318190348.258x^{9} \\ &+ 136104072.138x^{8} - 43997519.643x^{7} + 10683755.3645x^{6} - 1923308.64945x^{5} \\ &+ 251149.846545x^{4} - 22982.0402251x^{3} + 1389.40687206x^{2} - 46.8307670106x^{1} \\ &+ 1.34198723057 \end{aligned} \tag{5.6}$$

The power density profile and the original data are shown in Figure 5.2. The vertical axis has been normalized to a height of one. Different polynomial order were tried and the current 16 order fit was found as a practical choice for good fit. This polynomial power density is applied to the fuel matrix.

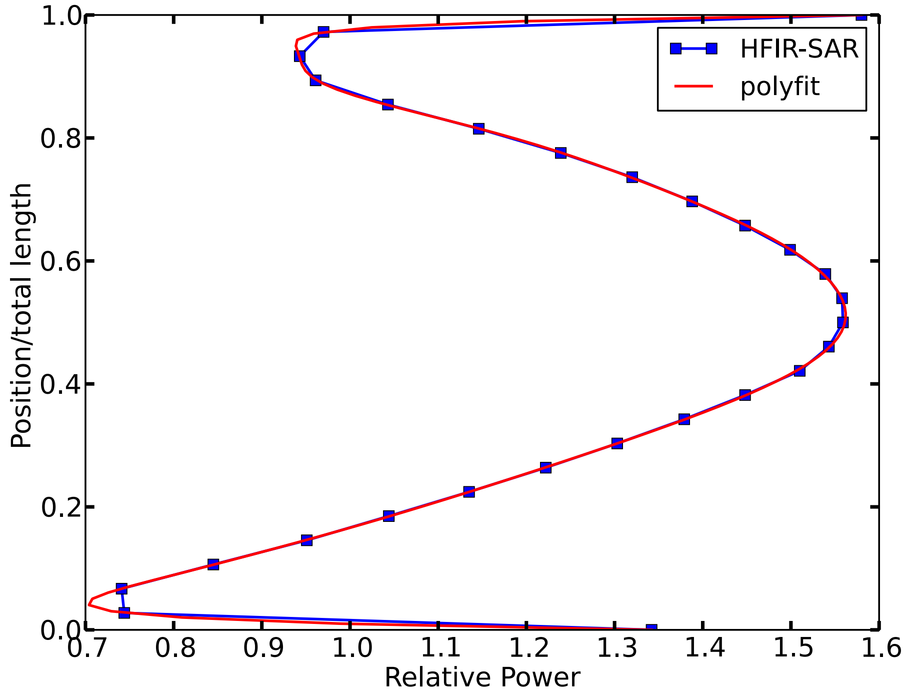

Figure 5.2: Model relative rower density distribution.

The material properties used in this work are shown in Table 5.1 [86, 87]. In Table 5.1, the row "phase change" is the temperature where the phase changes into another. The liquid water boils at 373 K, the solidus temperature of solid Aluminum is 819 K, and the liquidus temperature is 924 K.

Some of the properties where unavailable and these were replaced by the closest material state value. The aluminum properties with asterisk * are the properties of Al 6061-O, the actual cladding material used in the fuel. The aluminum properties without the asterisk * are those of pure aluminum. All of molten aluminum properties are that of pure aluminum. Heat capacity for liquid aluminum could not be found and the heat capacity of solid aluminum is used.

Table 5.1: Material properties.

	Al		Coolant
	solid	liquid	liquid
density [kg/m^3]	2700*	2357	1000
phase change [K]	819*	924*	373
viscosity [kg/m-s]		0.012	0.001
thermal conductivity [W/m-K]	180*	100	0.58
heat capacity [J/kg-K]	902*	902	4200
latent heat [J/kg]	321000		334000

The dimensional size and the different particle placements are shown in Figure 5.3. The right image is the particle representation of the left side. In Figure 5.3 (right), initially liquid water surrounds the solid aluminum at the center. No molten aluminum exists.

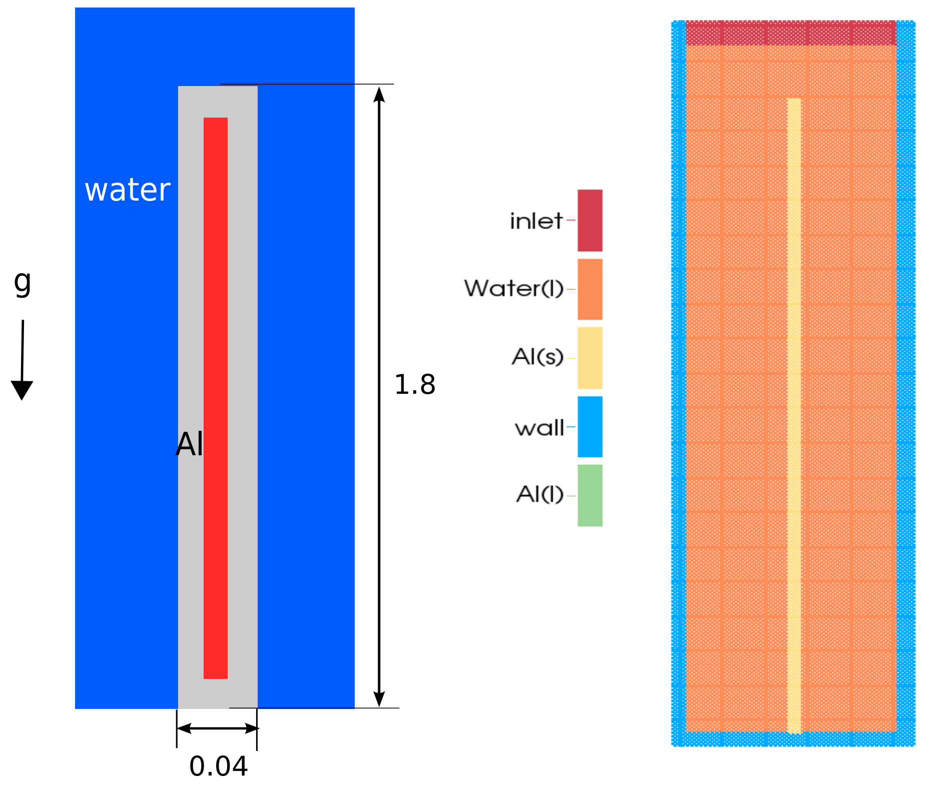

Figure 5.3: Model problem. Left figure is not to scale.

The dimensions of the model used in the simulation are in Table 5.2. Just like the actual HFIR fuel matrix, the fuel matrix in this model is slightly biased to the left. The initial number of particles by type is shown in Table 5.3, which corresponds to Figure 5.3 (right). The reserve particles are not shown in Figure 5.3.

Table 5.2: Dimensions [m].

	height	thickness
fuel matrix	1.2	0.02
fuel	1.8	0.04
coolant channel	2.0	0.5

Table 5.3: Initial number of particles.

Particle Type	Number of Particles
Inlet	672
Water (l)	17505
Al (s)	1074
Wall	3448
Al (l)	0
reserve	700

5.2 Simulation Cases

In the two dimensional computational domain, the particles are staggered with spacing of 0.02 m, which corresponds to over 23000 particles. The time step $dt = 2E - 5$ s for up to eight seconds, which takes slightly more than half an hour (using GTX 680) writing out 400 solution steps.

Various factors affect the melting phenomena. For example, viscous melts flow more gradually and will remain more local to the initial position. The power and initial margin to the melting temperature affects the time to melting. Conditions that could be encountered during an actual flow blockage for an aluminum plate fuel research reactor are considered here.

Simulations were set to study the effect of power density, channel flow after blockage, and initial fuel plate temperature. Six cases with varying conditions for the channel flow velocity, power density, and initial coolant temperature were run as shown in Table 5.4. The initial fuel plate temperature is set at 100 K above the coolant temperature. The last two cases with an asterisk * did not properly complete the simulation. The cause of the failure appears to be particle clumping.

Table 5.4: Study case conditions.

Case	Channel Flow [m/s]	Power density [MW/L]	Coolant Temperature [K]
1	0.1	0.71	330
2	0.1	0.71	400
3	1.0	0.71	400
4	0.1	1.34	330
5*	0.1	1.34	400
6*	1.0	0.71	330

In the following sections the results of the runs are presented. The figures use the same four times for display, top left: 2 sec, top right: 4 sec, bottom left: 6 sec, bottom right: 8 sec. for the first four cases. Videos of the simulations are attached for a better understanding of the melt sequence.

5.2.1 Case 1: low power, slow flow, low temperature

This first case is characterized by the low power density and slow flow, nearly resembling a melting in a static channel with melt migrating due to the density difference and gravity. Figure 5.4 shows the temperatures at four times during the simulation. The fuel uneventfully heats up for up to t=4 seconds. Figure 5.5, shows the particle types for the Case 1 simulation. The inner portion of the fuel has started melting at time 4 seconds. Parts of the molten fuel melts and flows downward at t=6 seconds. For much of the simulation, most of the molten material has a temperature between 1000-1400 Kelvin. With little flow to mix and distort the molten material, the center of the melt heats up. By the end of the run half of the center section of the fuel plate has melted away as seen in in Figure 5.5 in the 8 second frame.

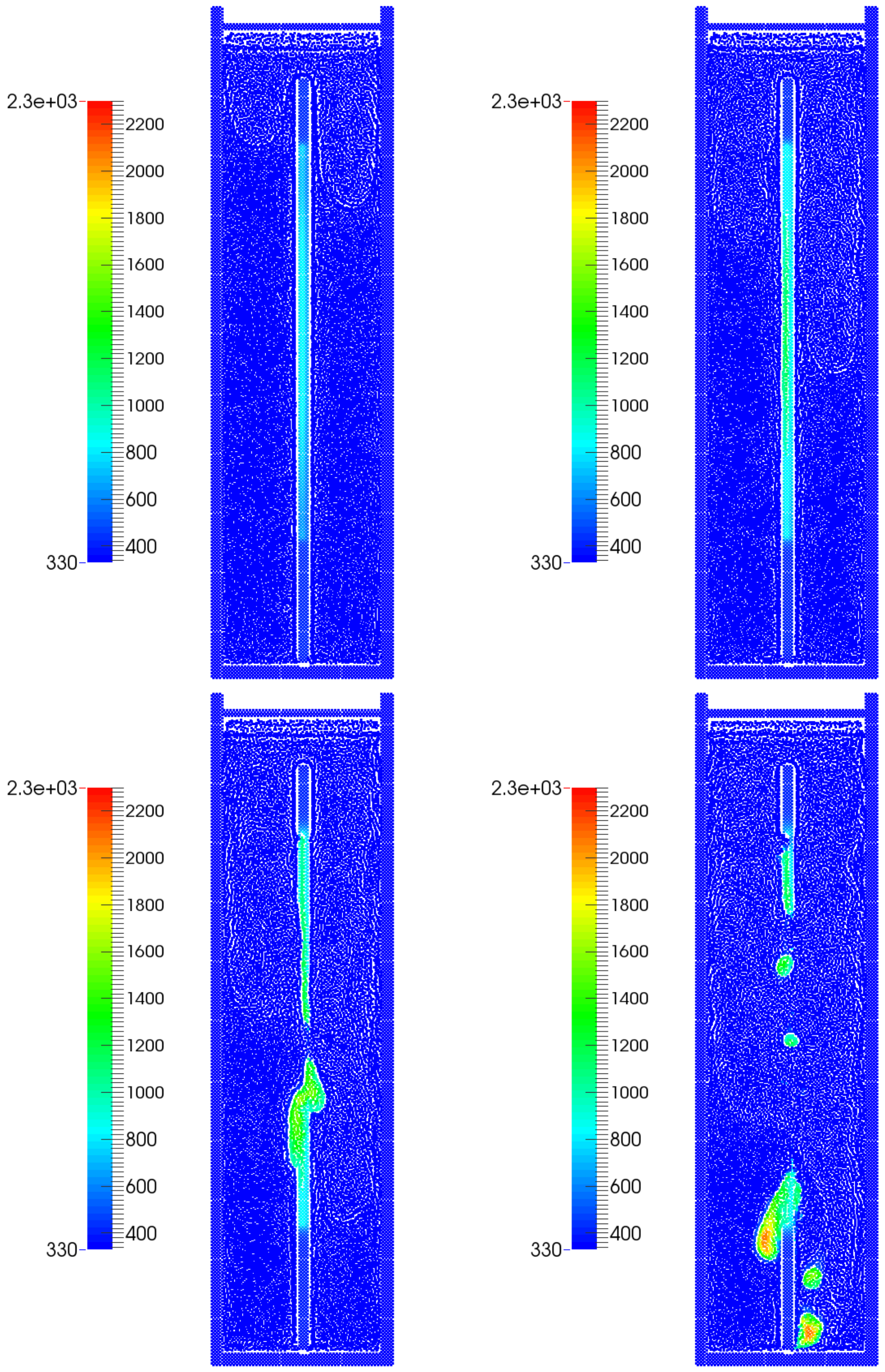

Figure 5.4: Case 1 temperature [K].

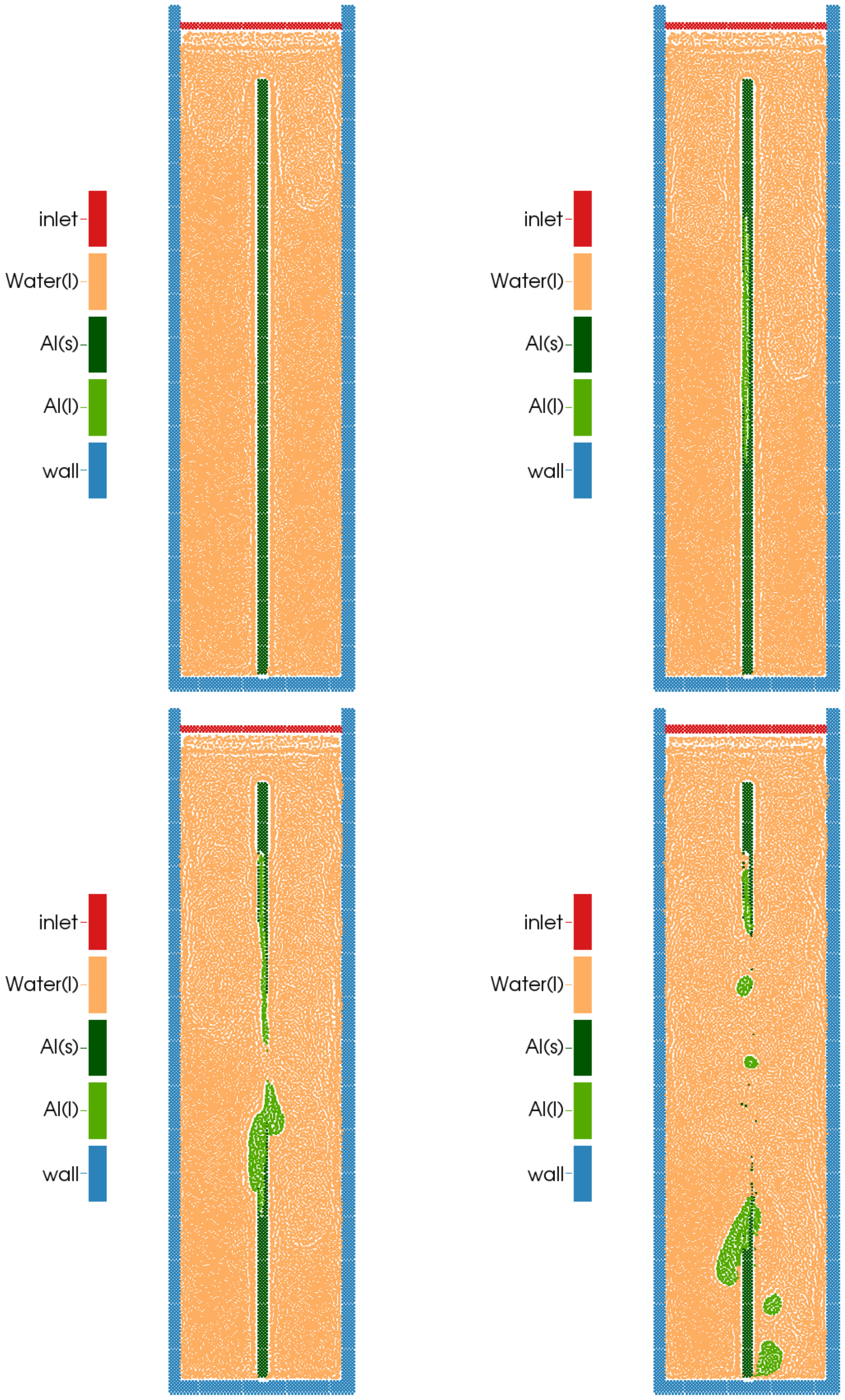

Figure 5.5: Case 1 particle types.

5.2.2 Case 2: low power, slow flow, high temperature

The initial starting temperature of the simulation is elevated 70°C for the second case, with all other parameters consistent with Case 1. Figure 5.6 and Figure 5.7 show the temperature and particle types, respectively. Case 2 events are similar to those of Case 1. The higher initial temperature causes quicker melting.

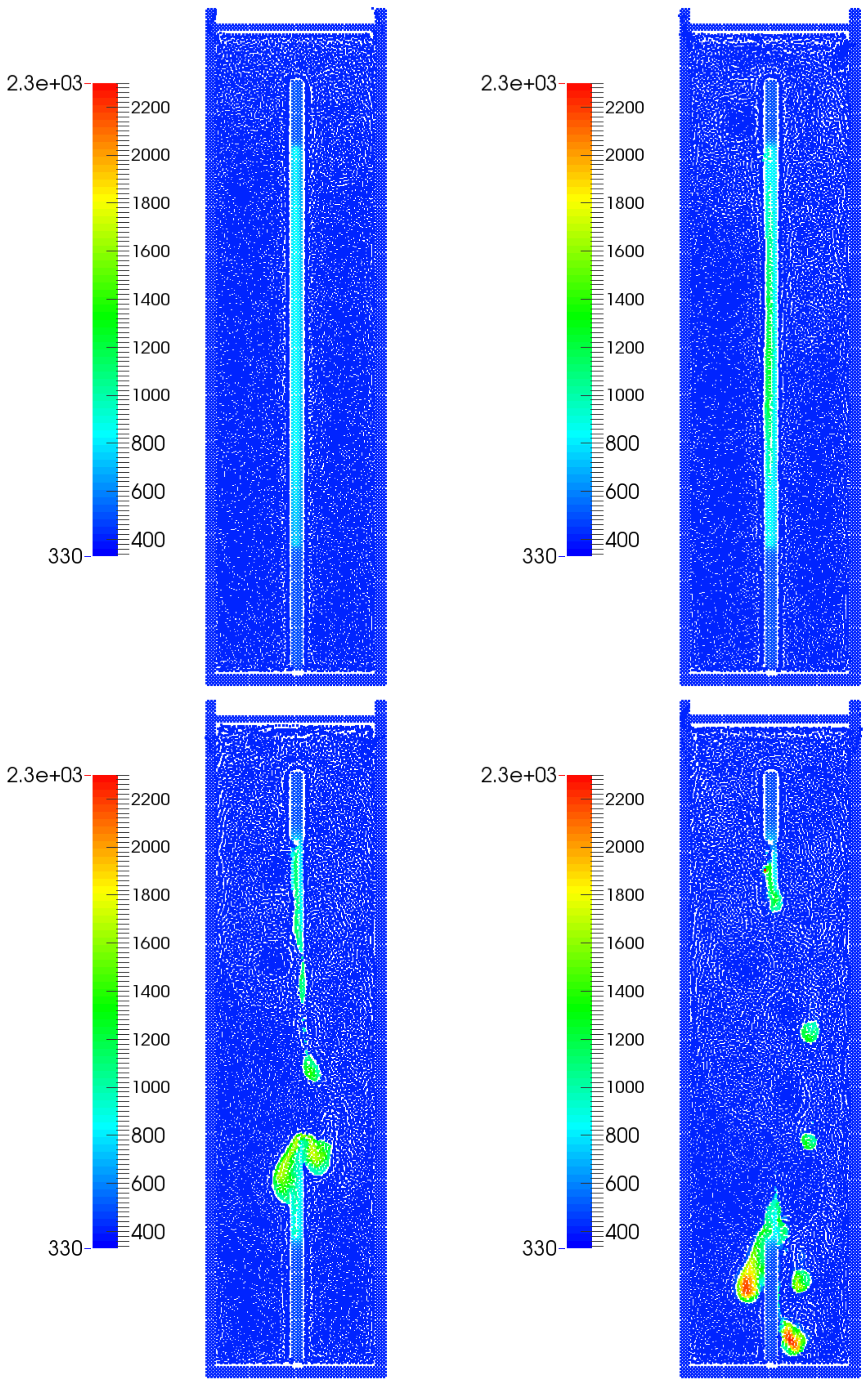

Figure 5.6: Case 2 temperature [K].

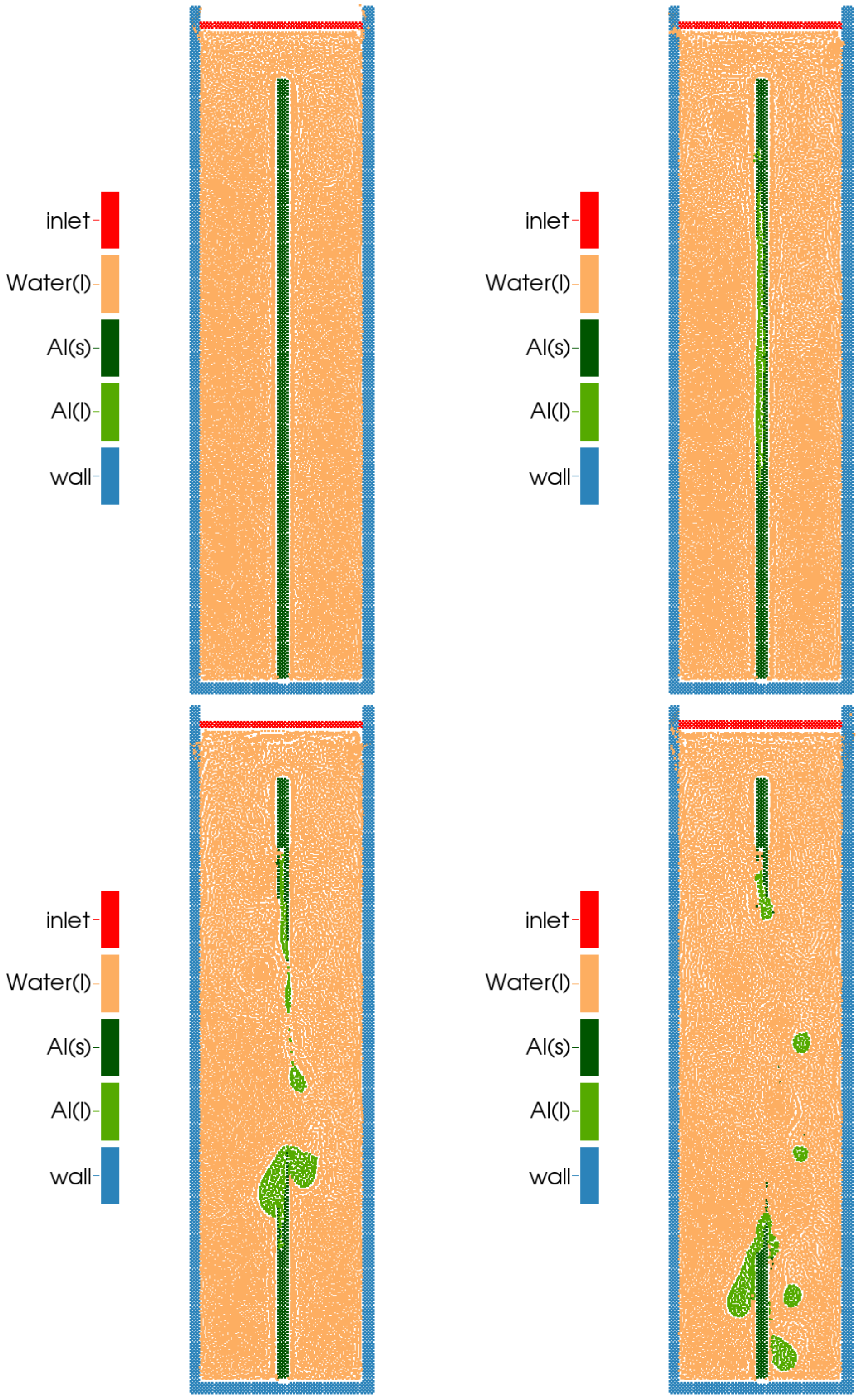

Figure 5.7: Case 2 particle types.

5.2.3 Case 3: low power, fast flow, high temperature

Case 3 is the same as Case 2 except the coolant velocity is elevated from 0.1 to 1.0 m/s. Figure 5.8 and Figure 5.9 show the temperature and particle types, respectively. The number of separate molten aluminum particles are fewer for this case, and the molten aluminum temperatures are lower. The first molten material leaves the channel more quickly, convecting with the flow, leaving little time to heat up in the channel and melt adjacent material. Also the clad to coolant heat transfer is improved, allowing the clad to avoid melting in the lower power density regions. Some of the cladding material, which has no internal heat generation, remains relatively unaffected as seen in Figure 5.9 at time equal to 8 seconds.

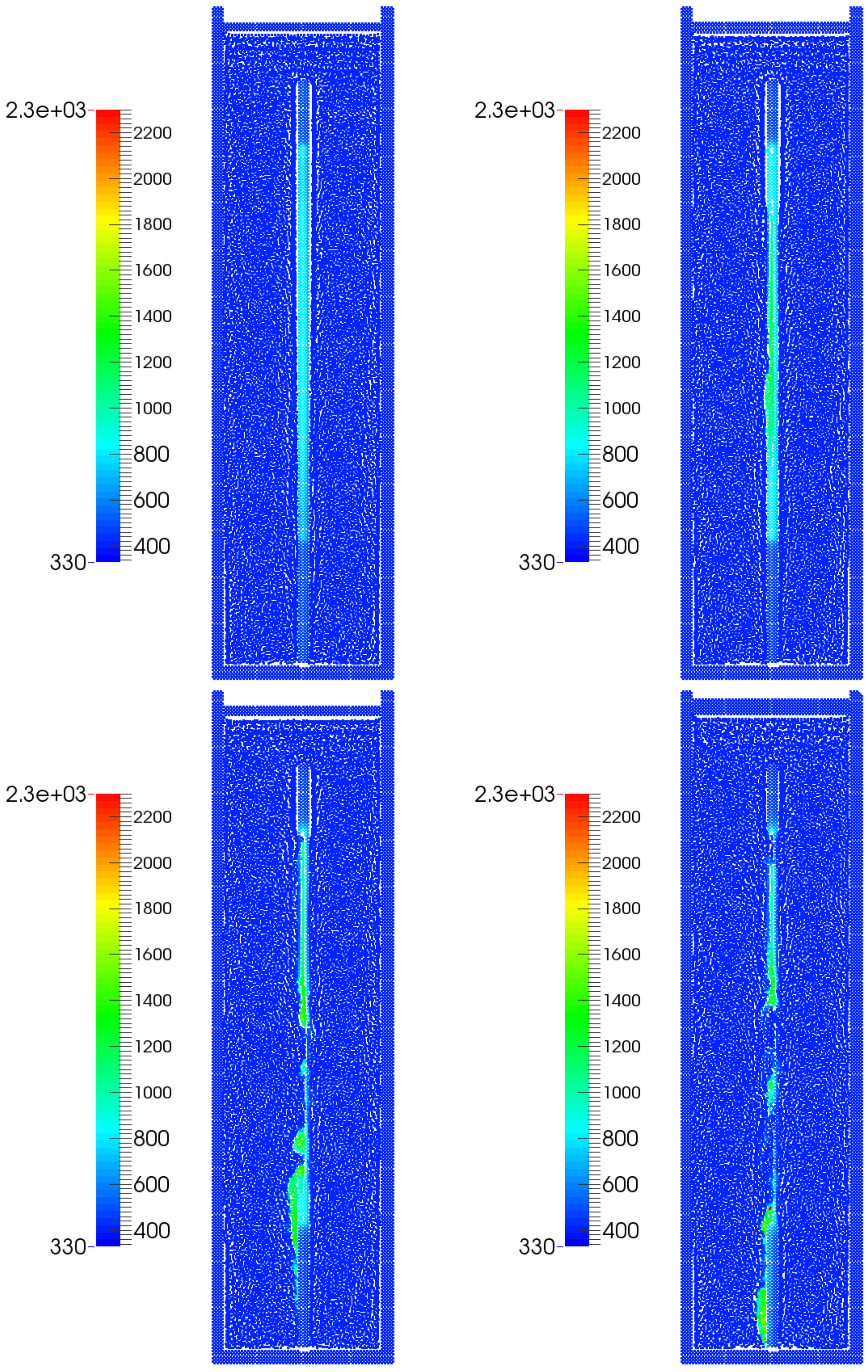

Figure 5.8: Case 3 temperature [K].

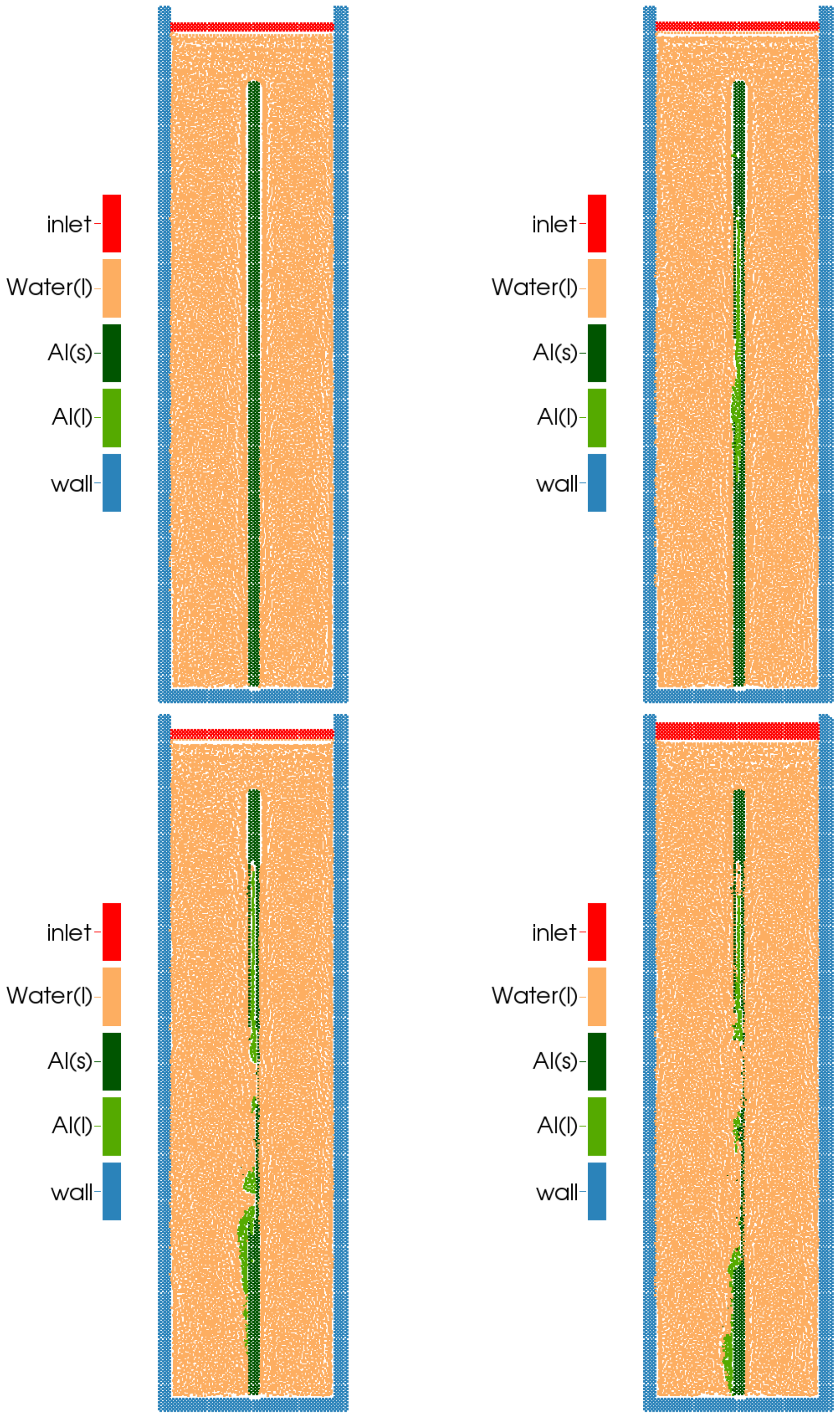

Figure 5.9: Case 3 particle types.

5.2.4 Case 4: high power, slow flow, low temperature

Case 4 has the highest power density with slow channel flow. Figure 5.10 shows the rapid heating of the fuel compared to all the previous cases. By t=6, parts of the melt have reach temperature exceeding 2300 Kelvin. This hot melt flows out but also heats the bottom portion of the still intact aluminum as seen in Figure 5.10 (t=8). At t=8 seconds, the entire section containing fuel matrix is gone.

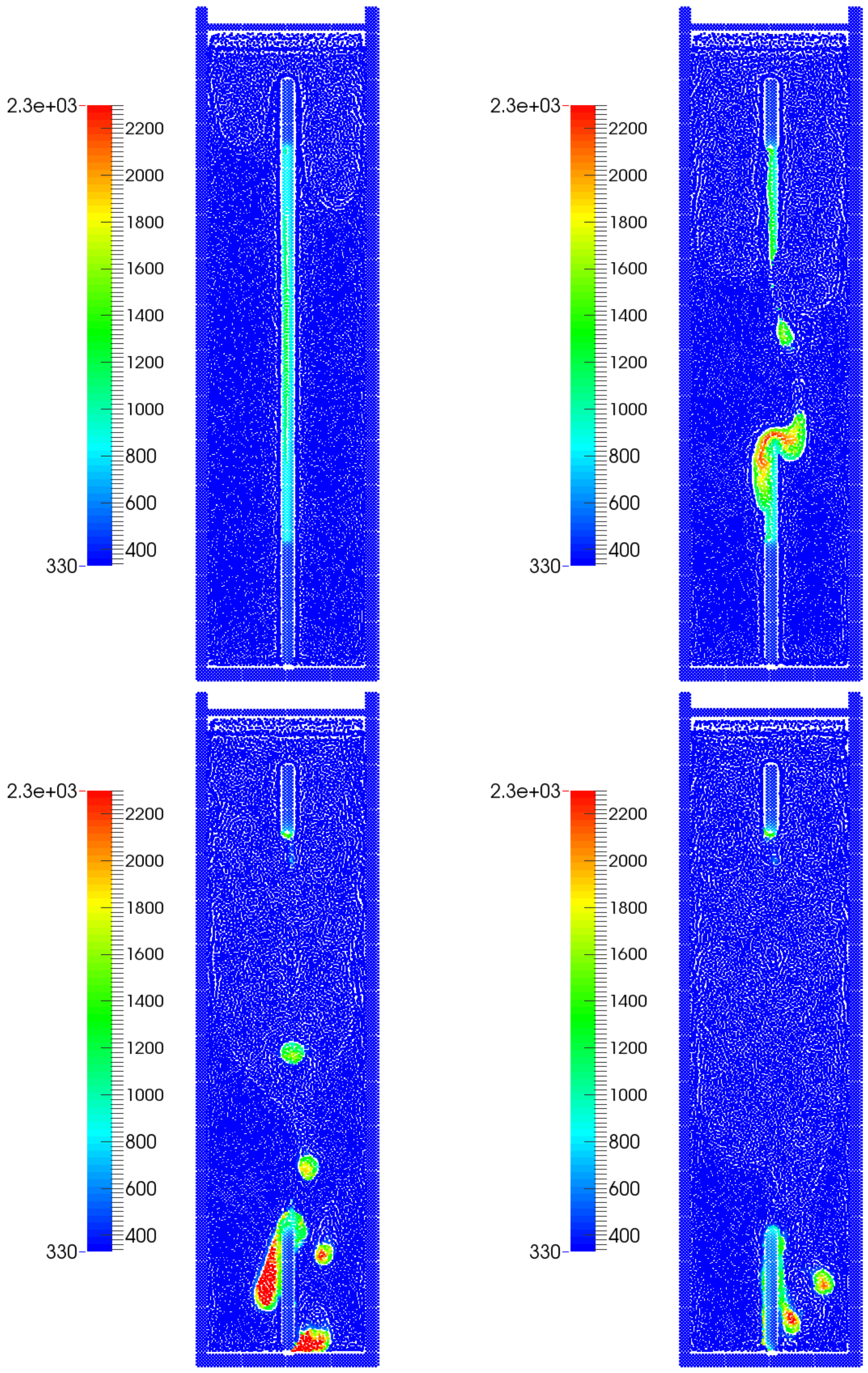

Figure 5.10: Case 4 temperature [K].

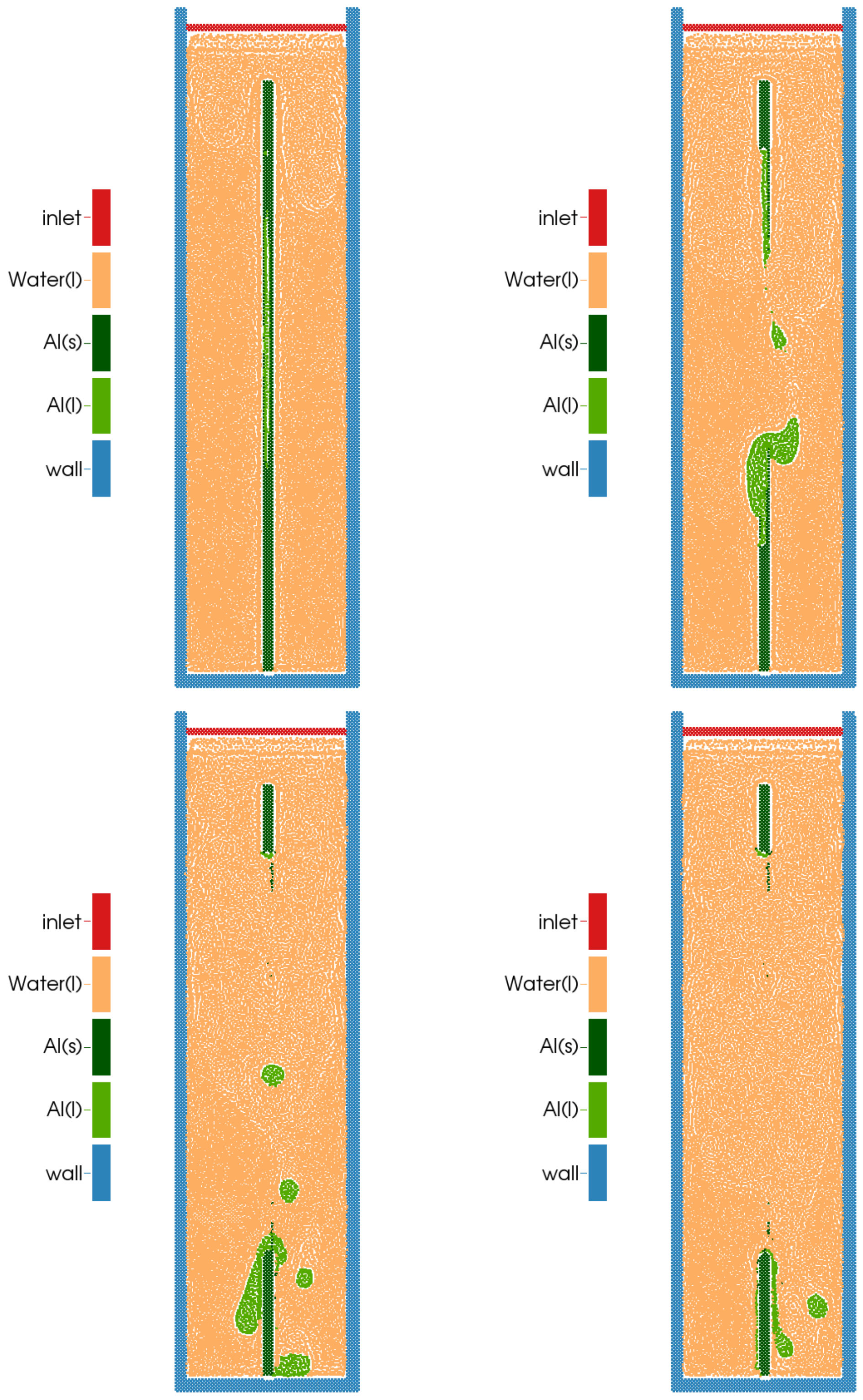

Figure 5.11: Case 4 particle types.

5.2.5 Case 5: high power, slow flow, high temperature

Case 5 has the same conditions as Case 4 for power and flow, but the coolant and fuel start 70 degrees hotter. The times for the figures shown below are t=2 (top left), 4 (top right), and 5 (bottom) seconds. The temperature field (Figure 5.12) of Case 5 follows Case 4 (Figure 5.10) for t=2 and 4 seconds. The simulation for Case 5 fails at t=5 seconds. The cause of failure appears to be due to some particles near the hottest region clumping together.

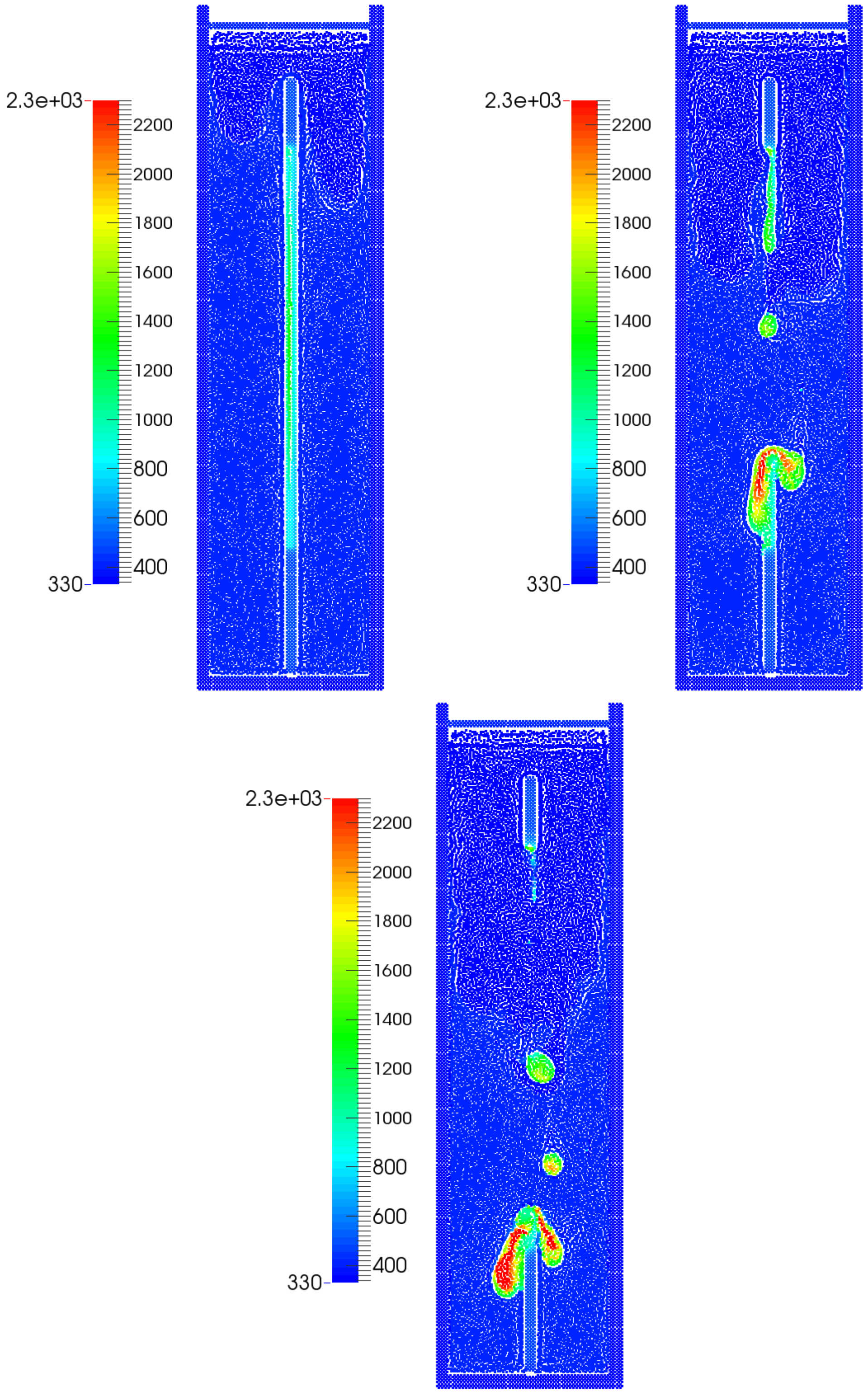

Figure 5.12: Case 5 temperature [K].

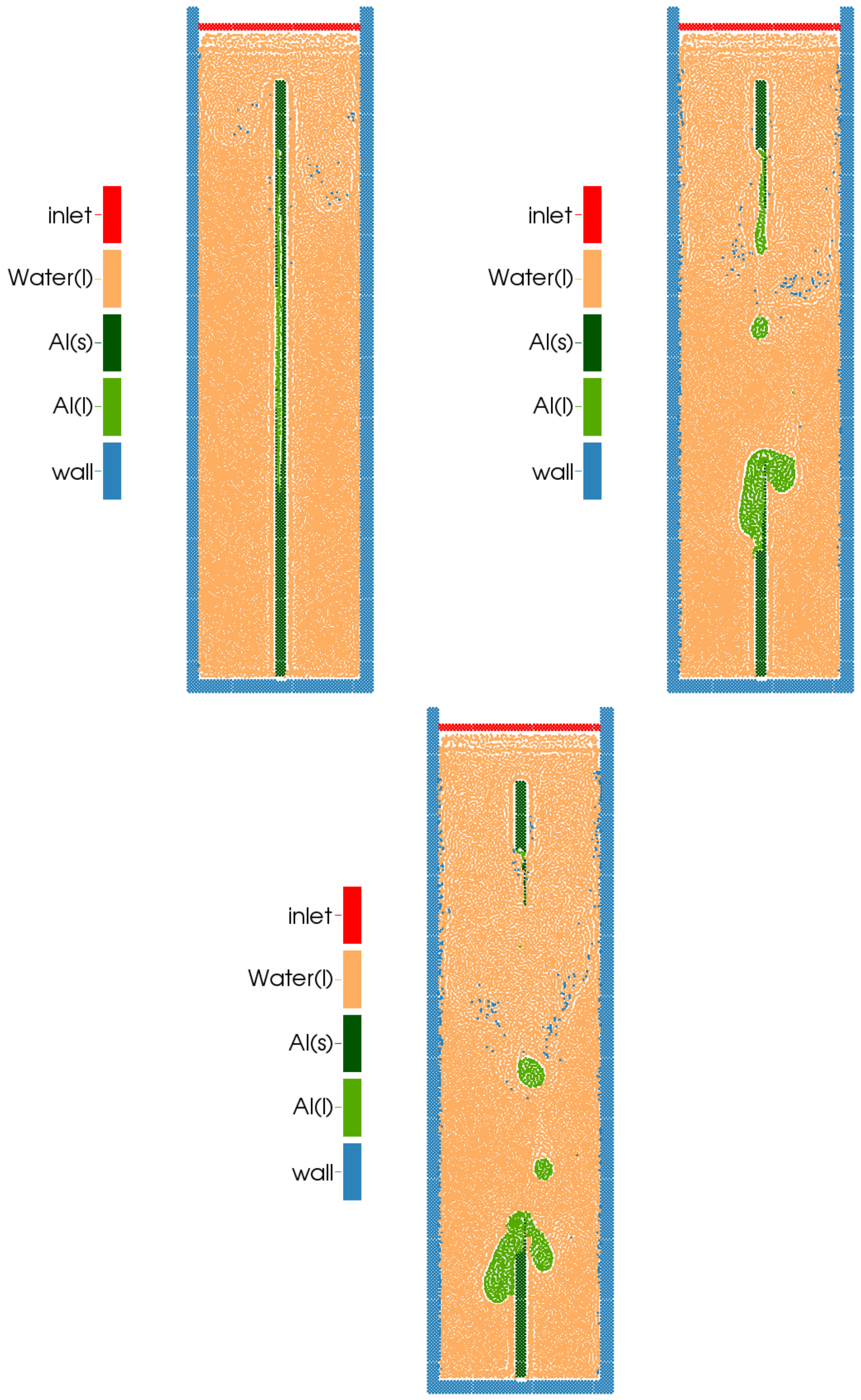

Figure 5.13: Case 5 particle types.

5.2.6 Case 6: low power, fast flow, low temperature

The final case has a low power density, fast channel flow, and low initial temperature. The times for the figures shown below are t=2(top left), 4(top right), 6(bottom left), and 7.2(bottom right) seconds. Case 6 shows larger melting (Figure 5.15) than Case 3 (Figure 5.9). The center section of the fuel is mostly melted away with some cladding. Case 3 has significant fuel and cladding remaining at time equal 6 seconds. The simulation fails at 7.2 seconds. The region where this occurs is at the melt location with the highest temperature. This hot spot does not have a temperature gradient indicating an abnormal value.

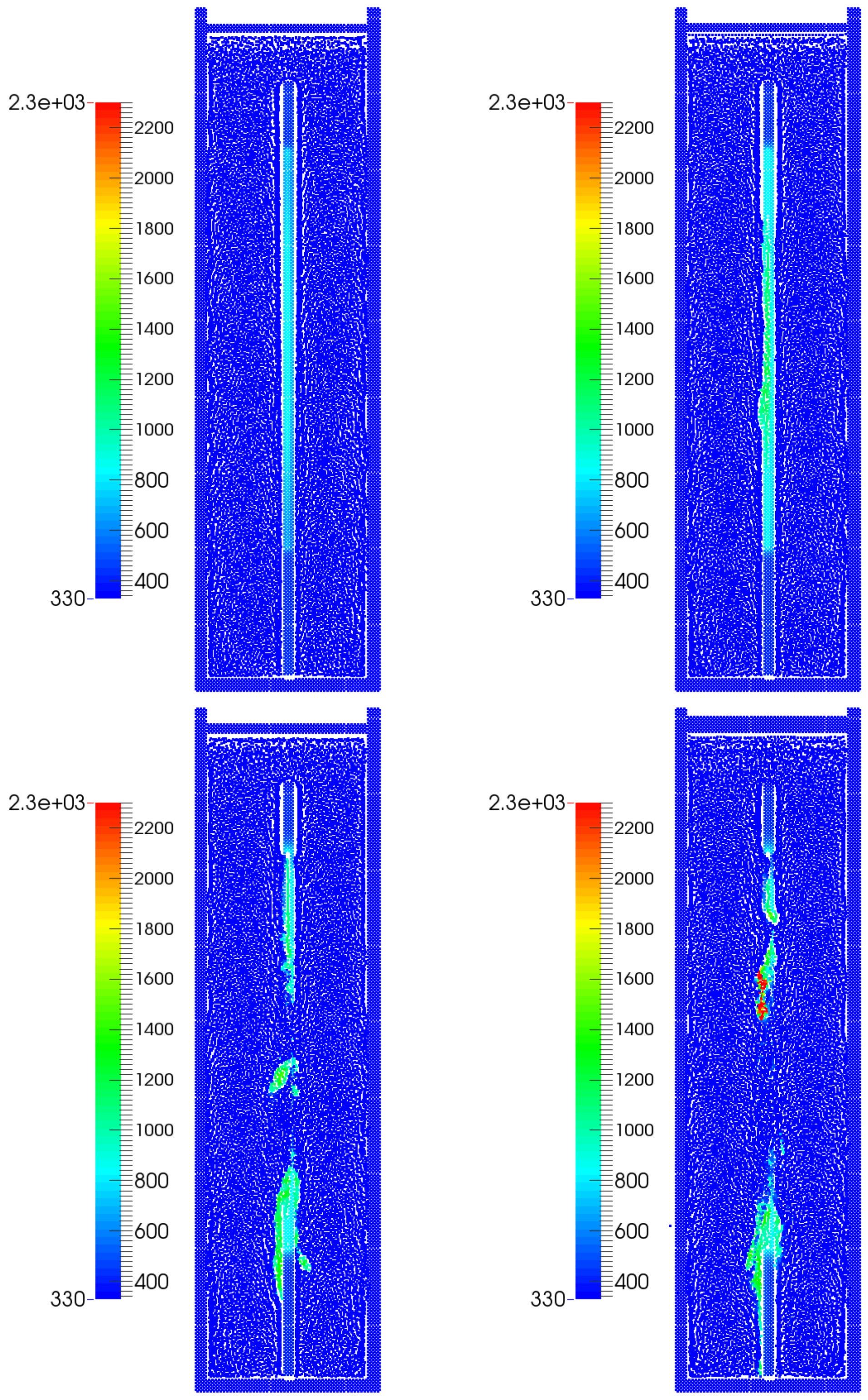

Figure 5.14: Case 6 temperature [K].

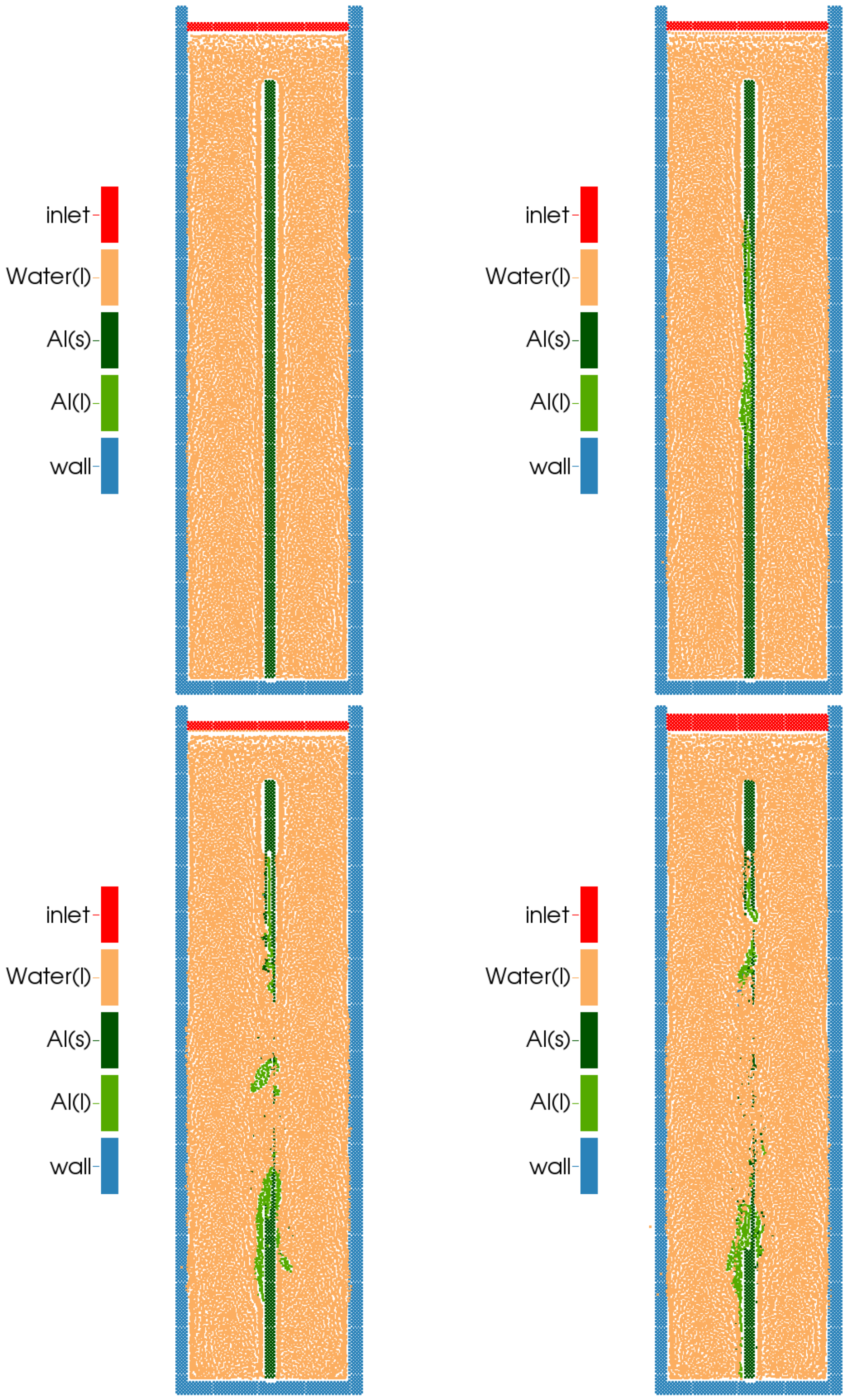

Figure 5.15: Case 6 particle types.

5.3 Temperature profiles and Melt fraction

The first case is low power and low flow. This is reflected by the gradual increase in temperature of the aluminum in Figure 5.16. Once the melting temperature of the aluminum is reached at 2.7 seconds, the first occurrence of liquid aluminum is apparent, with an average temperature of 1000 K. Since the initial molten aluminum may also have the power term, the melt continues to heat up. The average temperature of the solid falls as the particles continue to transition phase and the molten fuel becomes disconnected from the solid fuel plate.

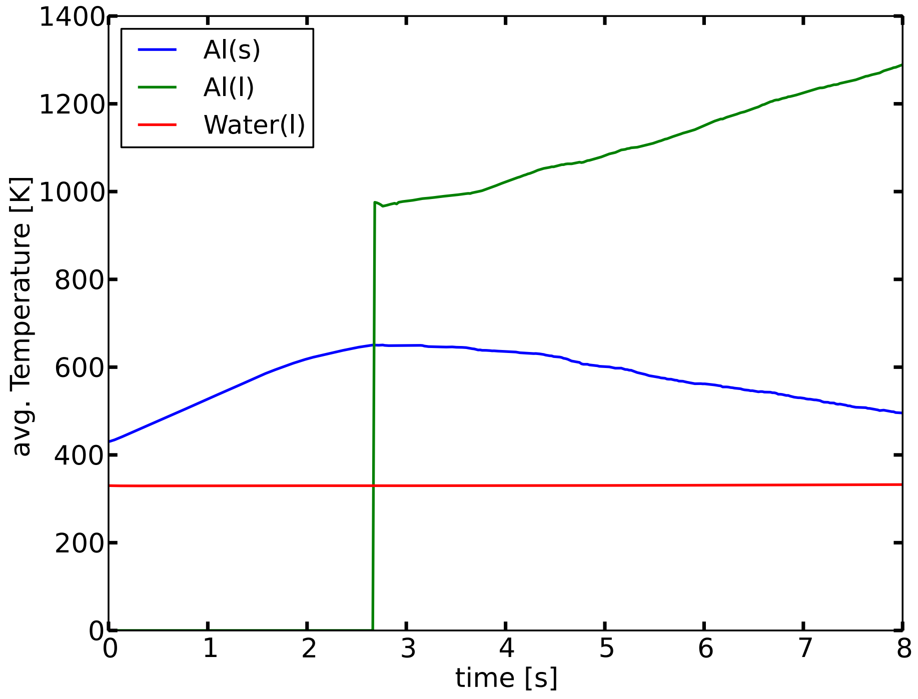

Figure 5.16: Case 1: Average temperature by material.

Figure 5.17 shows Case 2 where the initial temperature is elevated 70°C from that in Case 1. The occurrence of the first melt is evident at around 2.3 seconds, roughly 0.4 seconds sooner than Case 1.

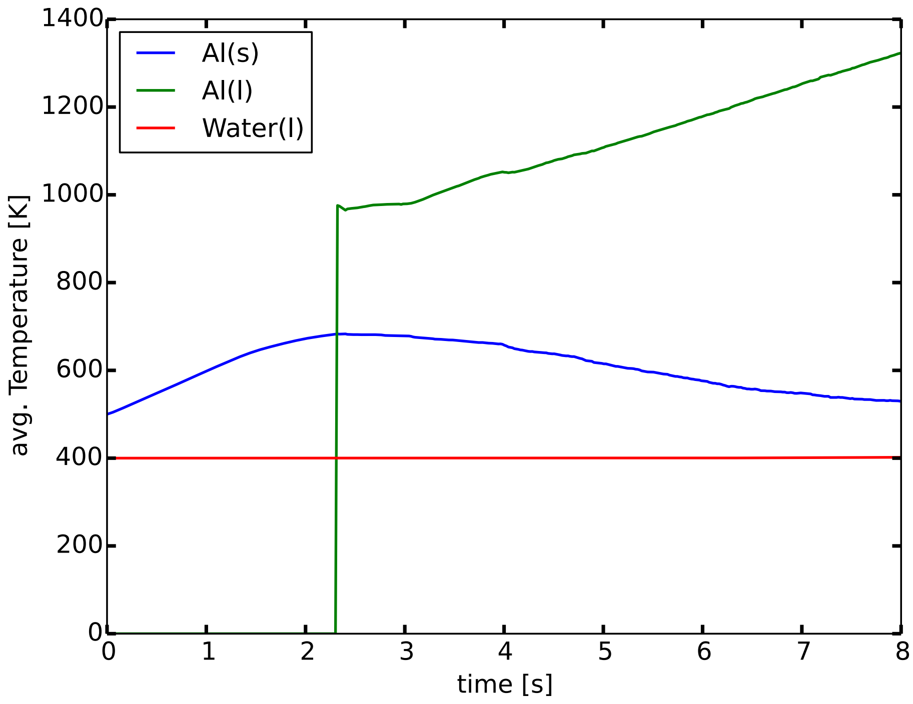

Figure 5.17: Case 2: Average temperature by material.

Figure 5.18 shows the results of Case 3 which is identical to Case 2 except the coolant flow is elevated from 0.1 to 1.0 m/s. Case 3, having the same initial temperature and power density as Case 2, melts aluminum at identical times to Case 2 (2.3 sec). However, because the channel flow is faster, the rate of temperature increase for melted fuel is not as high as Case 1 or 2. Also, since molten material leaves the domain much more quickly than the slow flow conditions of Case 1 and Case 2, the average temperature of the melt does not continue to rise.

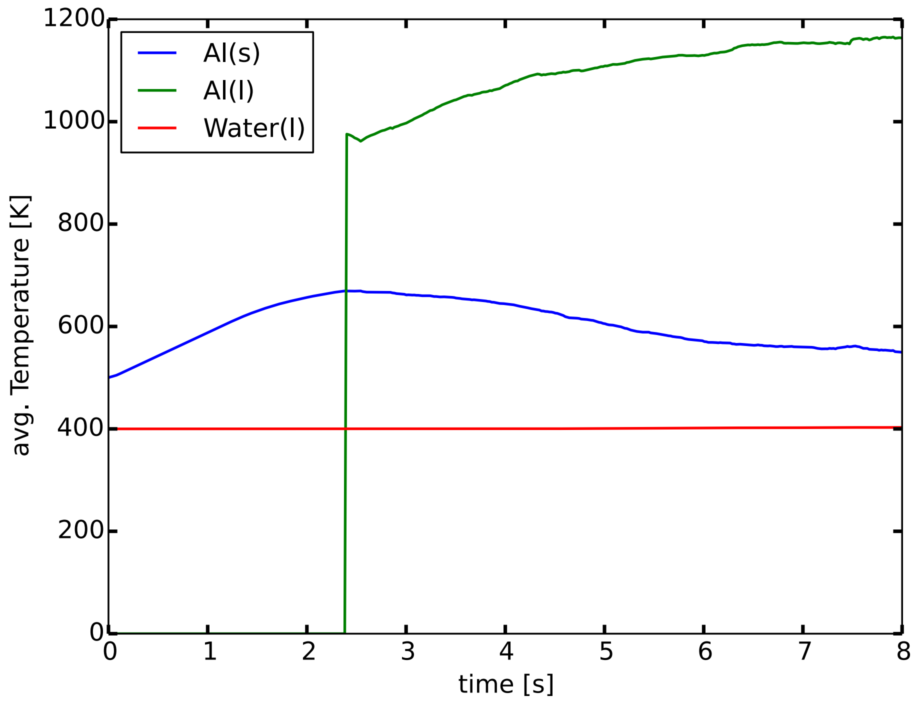

Figure 5.18: Case 3: Average temperature by material.

Case 4 shown in Figure 5.19, has nearly double the power density of Cases 1, 2, 3, and 6. The first melt is seen at 1.2 seconds into the simulation. The high power density drives the rapid increase in the molten aluminum temperature, reaching up to 1800 Kelvin. The molten fuel temperature falls in steps after 5 seconds as parts of the hot initial melt leave the computational domain in groups.

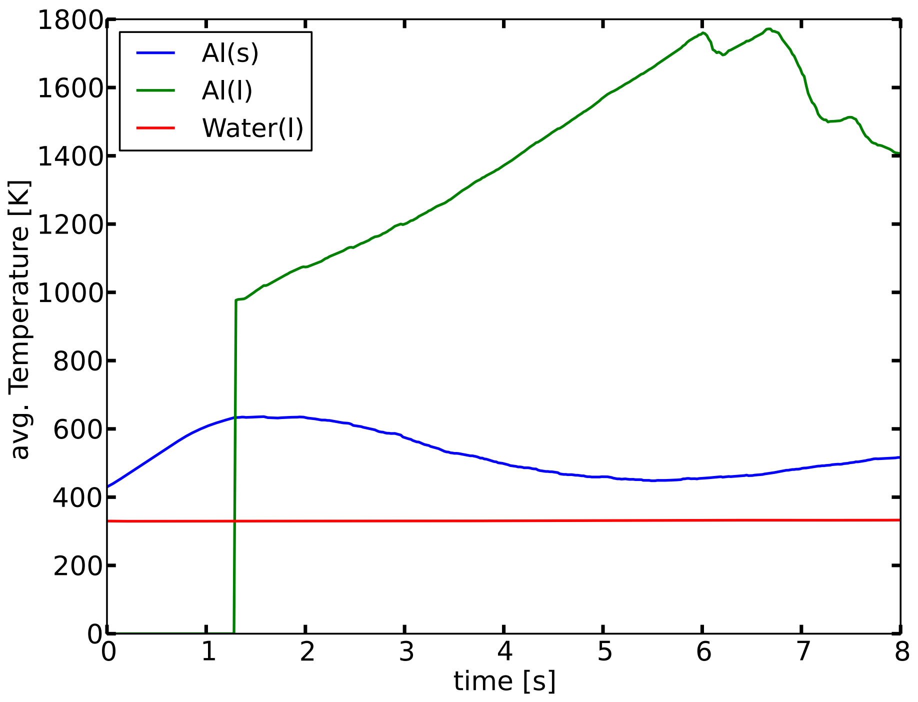

Figure 5.19: Case 4: Average temperature by material.

The Case 5 shown in Figure 5.20 show a similar trend to Case 4 (Figure 5.19). Due to the higher initial temperature of Case 5, the first appearance of molten aluminum occurs 0.2 seconds sooner than Case 4. However, Case 5 fails at 5 seconds into the simulation.

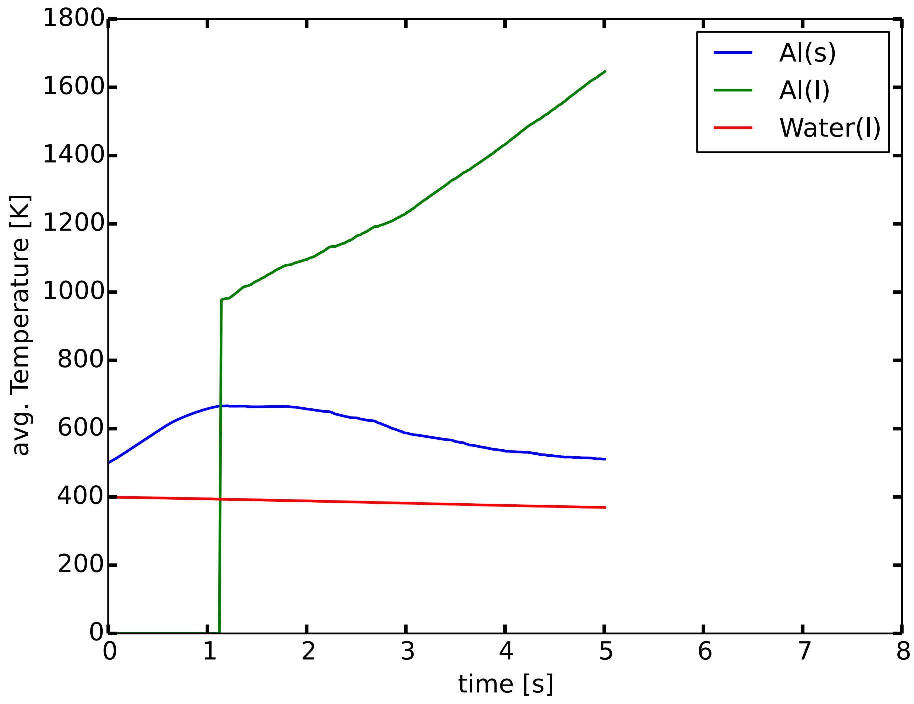

Figure 5.20: Case 5: Average temperature by material.

Case 6 shown in Figure 5.21 Slightly after t=6 seconds a spike in the molten aluminum temperature indicates that at least one particle has an abnormal temperature. This develops into the hot spot seen in Figure 5.14 at t=7.2 seconds.

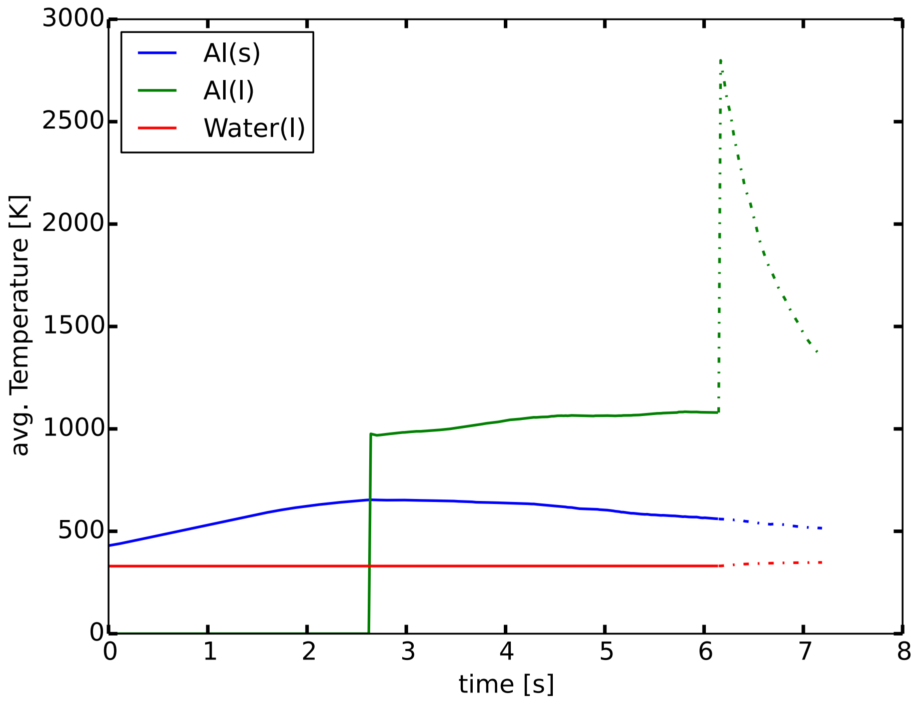

Figure 5.21: Case 6: Average temperature by material.

The simulation captures the complex behavior of molten fuel and molten material migration. Due to the lack of a model for the forced convective heat transfer between the fuel wall and the coolant water, the average coolant temperature profile in all cases above does not heat up. The initial heat transfer coefficient between the surface of the fuel plate and the coolant is the same for all cases. Note also that the initial temperature difference is set to 100 K for all cases. Table 5.5 shows the approximate heat transfer coefficient at the surface near the center of the fuel plate. It is calculated for all six cases right before the onset of melting. To obtained these values, the temperature field is constructed by Delaunay triangulation, then the gradient of temperature is calculated, followed by interpolating the points to obtain the gradient normal to the fuel plate surface. The calculation of the gradient is approximate. Case 3 and 6 have higher channel flow and Case

4 and 5 have higher power density, both conditions contributing to a higher heat transfer coefficient than the first two cases.

Table 5.5: Heat transfer coefficient between fuel plate and coolant before first melt.

Case	heat transfer coefficient[W/m^{2}K]
1	2768
2	2735
3	3426
4	3015
5*	3181
6*	3238

Figure 5.22 shows the melt fraction of all the cases as a function of time. The melt fraction is defined as

$$melt\% = \frac{Al(l)_t}{Al(s)_{t=0}} \tag{5.7}$$

The fraction is calculated based on the number of particles in each phase. Figure 5.22 shows the melt fraction for all the cases. Case 4 has an early development of melting and continues melting until more than half the material has melted. Case 6 also follows the same trend as Case 4. The melt fraction of Case 4 peaks at around 5 seconds and decreases due to the movement of the melt out of the computational domain. The factor differentiating the second and the third case is the channel flow contributing to the migration of the molten aluminum to melt more material down stream. The slower flow in Case 1 and Case 2 allow more time for molten aluminum to interact with the solid aluminum. In contrast, the faster flow of Case 3 and Case 6 remove molten aluminum from the channel. The two major factors contributing to high melt fraction are high power density and the duration of molten fuel interacting with the solid fuel.

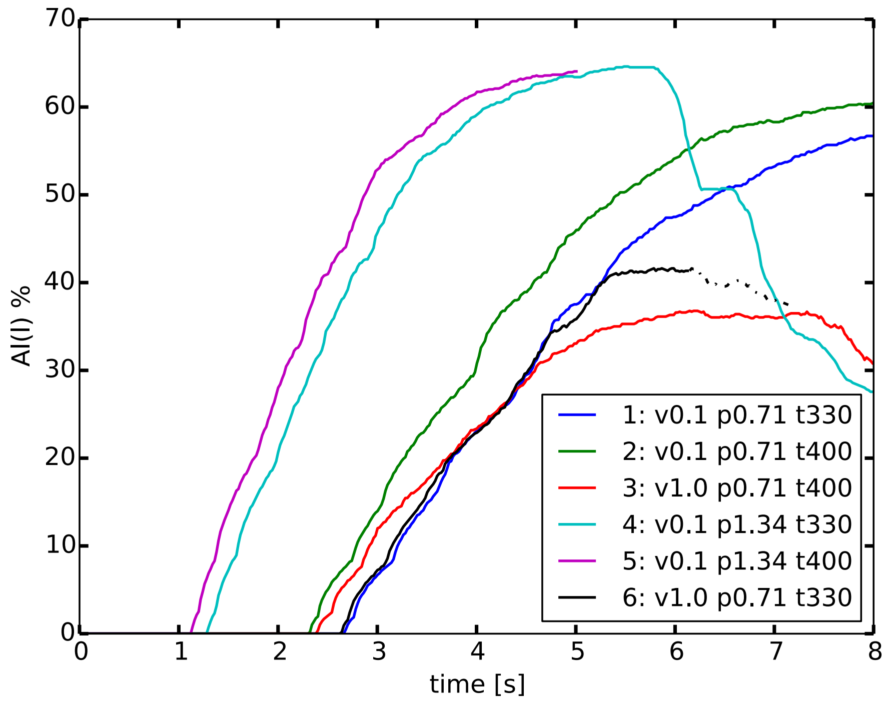

Figure 5.22: Melt fraction vs time of the different cases. The legend shows the parameters v(velocity), p(power), and t(temperature) of Table 5.4 for each case.

The fuel melt behavior of Case 3, with high coolant velocity, low power density, and higher initial temperature, and Case 6, with low coolant velocity, low power density, and low initial temperature are close in melt fraction. The Case 6 melt fraction is initially higher perhaps because less material is swept out of the computational domain due to the onset of particle clumping before the failure. Case 3 and Case 6 have nearly converged at the time of failure for the Case 6 run.

Chapter 6

Conclusions

The objective of this research was to develop simulation tools suited to assessment of fuel melting following inlet coolant blockage in the HFIR and other MTR-type reactors. Available computational methods for modeling fuel melting in coolant were reviewed, along with the history of fuel melting events in MTR reactor designs. Particle based methods were selected for the fuel simulation and are presented in this work. This fuel melt progression model is composed of a multi-fluid SPH formulation combined with a heat transfer and phase change model. The model was implemented in a graphics processing unit (GPU) using CUDA. An overview of the code structure is shown in Appendix D. A user guide will be available describing how to create, simulate, and post-process custom simulations.

The new model is tested against analytical solutions for flow, conduction and 1-D melt progression and shows good agreement. The new model is applied to a scaled MTR fuel melt progression. Four different cases are presented with varying power density, initial fuel temperature and coolant flow velocity. Elevated power density accelerates melt progression. Increased coolant flow preserves some cladding integrity and rapidly sweeps melted fuel out of the computational domain.

The current work shows the particle method is useful in modeling the complex geometric evolution of fuel melting in coolant. This work is the first implementation of SPH with three material components and two phases. The GPU implementation of SPH is completely custom to the MTR fuel melt application.

In summary, this work presented the following:

- flow blockage is modeled mechanistically through fuel melt for MTR reactor fuel.
- multicomponent SPH model is developed and implemented on GPU. Multifluid SPH works have been theoretical and no algorithms are discussed. Most prior SPH implementations are single phase, or involve one free surface.
- multicomponent Smoothed Particle Hydrodynamics is combined with heat transfer and phase change. No prior treatment of this multiphysics case in SPH exists.
- inflow/outflow computational "domain" boundary created in SPH.

This work will benefit from improved modeling of convective heat transfer between fuel and coolant, and the addition of radiative heat transfer models for high temperature conditions. The scaled fuel melting used in this work can be moved to a more direct unscaled simulation with additional computational resources. Further work into better modeling the heat transfer between the different interfaces after melting is also needed. The swelling of the fuel plates due to the release and migration of fission gases with increasing temperature is another physical phenomenon not currently modeled that may significantly influence simulation outcomes.

Though it was not a focus of this work, future work could include optimization of the code to improve run times. The use of a distributed GPU structure with MPI could allow escalation of the simulation size and increase in the level of detail in the model. SPH method is established, in certain applications, but it is not as mature as traditional mesh-based computational fluid dynamics tools. Particle based methods require further understanding of stability limits, and higher level programming tools to improve accessibility of the method to engineers not expert in GPU programming. However, the SPH method is compatible with parallel computing architectures and promises to rapidly gain in utility as these computing architectures are more commonly used.

References

[1] Research Reactor Division. 4.4 thermal and Hydraulic Design. Technical Report ORNL/SAR-2344 rev.8, ORNL, 2011.

[2] Oak Ridge National Laboratory. Reactor Technical Parameters: Overview. http://neutrons.ornl.gov/facilities/HFIR/techparameters.shtml.

[3] Oak Ridge National Laboratory. HFIR fuel element dimensions. http://neutrons.ornl.gov/facilities/HFIR/reactorassembly.shtml.

[4] Research Reactor Division. Executive Summary. In *Probabilistic Risk Assessment*. 2004. ORNL/RRD/INT-36/Rev.2.

[5] D. H. Johnson. Appendix B. Probabilistic Risk Assessment of Flow Blockage Events on the HFIR. In *Probabilistic Risk Assessment*. 2004. ORNL/RRD/INT-36/Rev.2.

[6] Oak Ridge National Laboratory. High Flux Isotope Reactor Fuel Assembly Photo, January 2013. http://en.wikipedia.org/wiki/File:High_Flux_Isotope_Reactor_Fuel_Assembly_Photo.jpg.

[7] Qing Lu, Suizheng Qiu, and G.H. Su. Flow blockage analysis of a channel in a typical material test reactor core. *Nuclear Engineering and Design*, 239(1):45 – 50, 2009.

[8] Amgad Salama and Salah El-Din El-Morshedy. CFD simulation of flow blockage through a coolant channel of a typical material testing reactor core. *Annals of Nuclear Energy*, 41(0):26 – 39, 2012.

[9] Martina Adorni, Anis Bousbia-Salah, Tewfik Hamidouche, Beniamino Di Maro, Franco Pierro, and Francesco D'Auria. Analysis of partial and total flow blockage of a single fuel assembly of an MTR research reactor core. *Annals of Nuclear Energy*, 32(15):1679 – 1692, 2005.

[10] Amgad Salama. CFD investigation of flow inversion in typical MTR research reactor undergoing thermal-hydraulic transients. *Annals of Nuclear Energy*, 38(7):1578 – 1592, 2011.

[11] Amgad Salama and Salah El-Din El-Morshedy. CFD simulation of the IAEA 10 MW generic MTR reactor under loss of flow transient. *Annals of Nuclear Energy*, 38(2-3):564 – 577, 2011.

[12] Amgad Salama and Salah El-Din El-Morshedy. CFD analysis of flow blockage in MTR coolant channel under loss-of-flow transient: Hot channel scenario. *Progress in Nuclear Energy*, 55(0):78 – 92, 2012.

[13] Jerry A. Crabtree. The Effect of Inlet Blockage Configuration on Flow Behavior in Rectangular Channels. Master's thesis, University of Tennessee, Aug 1997.

[14] A. Leenaers, F. Joppen, and S. Van den Berghe. Microstructural analysis of MTR fuel plates damaged by a coolant flow blockage. *Journal of Nuclear Materials*, 394(1):87 – 94, 2009.

[15] F. Joppen. Review of the accident source terms for aluminide fuel: Application to the BR2 reactor. In *Proc. 9th International Topical Meeting on Research Reactor Fuel Management,(Budapest, Hungary, April 2005)*. SCK-CEN, Belgium, 2005.

[16] J. R Kirkpatrick. Estimate of Propagation of Melting in HFIR Fuel Plates Initiated by Blockage of Two Adjacent Channels. Technical Report HFIR-CTD-91-JRK-15-2, ORNL, 1991.

[17] F. T. Binford, T. E. Cole, and E. N Cramer. The High Flux Isotope Reactor Accident Analysis. Technical Report ORNL-3573, ORNL, 1967.

[18] T. M. Sims and W. H. Tabon. Report on Fuel-Plate Melting at the Oak Ridge Research Reactor July 1. 1963. Technical Report ORNL/TM-627, ORNL, 1964.

[19] H. Nakamura. Review of Fuel Cooling Channel Flow Blockage for HFIR and Related Aluminum Clad Plate Fuel Reactor Designs. Technical Report ORNL/TM-2012/291, ORNL, 2013.

[20] Georges Berthoud. Vapor Explosions. *Annual Review of Fluid Mechanics*, 32(1):573–611, 2000.

[21] Research Reactor Division. 15.3.3 Decrease in Primary System Flow Rate. Technical Report ORNL/SAR-2344 rev.8, ORNL, 2011.

[22] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, 1977.

[23] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375, 1977.

[24] M. B. Liu and G. R. Liu. Smoothed Particle Hydrodynamics SPH: an Overview and Recent Developments. *Archives of Computational Methods in Engineering*, 17:25–76, 2010.

[25] K. Iwasaki, H. Uchida, Y. Dobashi, and T. Nishita. Fast particle-based visual simulation of ice melting. *Computer Graphics Forum*, 29(7):2215–2223, 2010.

[26] J. J. Monaghan. Extrapolating B splines for interpolation. *Journal of Computational Physics*, 60(2):253–262, 1985.

[27] J. J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703–1759, 2005.

[28] J. P. Morris. *Analysis of Smoothed Particle Hydrodynamics with Applications.* PhD thesis, Monash University, 1996.

[29] J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. *Astronomy and Astrophysics*, 149:135–143, August 1985.

[30] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, 1995.

[31] Martin Robinson. *Turbulence and Viscous Mixing using Smoothed Particle Hydrodynamics.* PhD thesis, Monash University, 2009.

[32] J. J. Monaghan. An introduction to SPH. *Computer Physics Communications*, 48(1):89–96, 1988.

[33] Joseph P. Morris, Patrick J. Fox, and Yi Zhu. Modeling low reynolds number incompressible flows using SPH. *Journal of Computational Physics*, 136(1):214–226, 1997.

[34] J. J. Monaghan and R. A. Gingold. Shock simulation by the particle method SPH. *Journal of Computational Physics*, 52(2):374–389, 1983.

[35] Paul W. Cleary. Modelling confined multi-material heat and mass flows using SPH. *Applied Mathematical Modelling*, 22(12):981–993, 1998.

[36] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574, 1992.

[37] X. Y. Hu and N. A. Adams. An incompressible multi-phase SPH method. *Journal of Computational Physics*, 227(1):264–278, 2007.

[38] N. Grenier, M. Antuono, A. Colagrossi, D. Le Touzé, and B. Alessandrini. An Hamiltonian interface SPH formulation for multi-fluid and free surface flows. *Journal of Computational Physics*, 228(22):8380–8393, 2009.

[39] J. J. Monaghan and Ashkan Rafiee. A simple SPH algorithm for multi-fluid flow with high density ratios. *International Journal for Numerical Methods in Fluids*, 71(5):537–561, 2013.

[40] Andrea Colagrossi and Maurizio Landrini. Numerical simulation of interfacial flows by smoothed particle hydrodynamics. *Journal of Computational Physics*, 191(2):448–475, November 2003.

[41] Paul W. Cleary and Joseph J. Monaghan. Conduction Modelling using Smoothed Particle Hydrodynamics. *Journal of Computational Physics*, 148(1):227–264, 1999.

[42] M. Antuono, A. Colagrossi, S. Marrone, and D. Molteni. Free-surface flows solved by means of SPH schemes with numerical diffusive terms. *Computer Physics Communications*, 181:532–549, March 2010.

[43] G. R. Liu and B. Liu. *Smoothed Particle Hydrodynamics: A Meshfree Particle Method.* World Scientific, 2003.

[44] J. W. Swegle, D. L. Hicks, and S. W. Attaway. Smoothed particle hydrodynamics stability analysis. *Journal of Computational Physics*, 116(1):123–134, January 1995.

[45] P. Randles. Smoothed Particle Hydrodynamics: Some recent improvements and applications. *Computer Methods in Applied Mechanics and Engineering*, 139:375–408, December 1996.

[46] P. W. Randles and L. D. Libersky. Normalized sph with stress points. *International Journal for Numerical Methods in Engineering*, 48(10):1445–1462, 2000.

[47] C. T. Dyka, P. W. Randles, and R. P. Ingel. Stress points for tension instability in sph. *International Journal for Numerical Methods in Engineering*, 40(13):2325–2341, 1997.

[48] J. J. Monaghan and A. Kos. Scott russell's wave generator. *Physics of Fluids*, 12(3):622–630, 2000.

[49] S. Koshizuka and Y. Oka. Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nuclear science and engineering*, 123(3):421–434, 1996.

[50] Seiichi Koshizuka, Hirokazu Ikeda, and Yoshiaki Oka. Numerical analysis of fragmentation mechanisms in vapor explosions. *Nuclear Engineering and Design*, 189(1-3):423–433, 1999.

[51] Shuai Zhang, Koji Morita, Kenji Fukuda, and Noriyuki Shirakawa. An improved MPS method for numerical simulations of convective heat transfer problems. *International Journal for Numerical Methods in Fluids*, 51(1):31–47, 2006.

[52] Seiichi Koshizuka, Atsushi Nobe, and Yoshiaki Oka. Numerical analysis of breaking waves using the moving particle semi-implicit method. *International Journal for Numerical Methods in Fluids*, 26(7):751–769, 1998.

[53] Ahmad Shakibaeinia and Yee-Chung Jin. A weakly compressible MPS method for modeling of open-boundary free-surface flow. *International Journal for Numerical Methods in Fluids*, 63(10):1208–1232, 2010.

[54] J.J. Monaghan. Simulating free surface flows with SPH. *Journal of Computational Physics*, 110(2):399–406, 1994.

[55] A. J. C. Crespo, M. Gomez-Gesteira, and R. A. Dalrymple. Boundary Conditions Generated by Dynamic Particles in SPH Methods. *Computers, Materials, & Continua*, 5(3):173–184, 2007.

[56] R. Vacondio, B. Rogers, P. Stansby, and P. Mignosa. SPH Modeling of Shallow Flow with Open Boundaries for Practical Flood Simulation. *Journal of Hydraulic Engineering*, 138(6):530–541, 2012.

[57] S. Adami, X.Y. Hu, and N.A. Adams. A generalized wall boundary condition for smoothed particle hydrodynamics. *Journal of Computational Physics*, 231(21):7057 – 7075, 2012.

[58] J. Monaghan and A. Kos. Solitary waves on a cretan beach. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 125(3):145–155, 1999.

[59] Angela Ferrari, Michael Dumbser, Eleuterio F. Toro, and Aronne Armanini. A new 3D parallel SPH scheme for free surface flows. *Computers & Fluids*, 38(6):1203–1217, 2009.

[60] I. Federico, S. Marrone, A. Colagrossi, F. Aristodemo, and M. Antuono. Simulating 2D open-channel flows through an SPH model. *European Journal of Mechanics - B/Fluids*, 34(0):35–46, 2012.

[61] NVIDIA. Geforce GTX 680 Specifications, May 2012. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-680/specifications.

[62] AMD. AMD Radeon HD 7970 Graphics, Sep 2013. http://www.amd.com/us/products/desktop/graphics/7000/7970/Pages/radeon-7970.aspx#3.

[63] T. Harada, S. Koshizuka, and Y. Kawaguchi. Smoothed particle hydrodynamics on GPUs. In *Computer Graphics International*, pages 63–70, 2007.

[64] Thrust Parallel Algorithms Library, Nov 2012. http://thrust.github.io/.

[65] Alexis Hérault, Giuseppe Bilotta, and Robert A. Dalrymple. SPH on GPU with CUDA. *Journal of Hydraulic Research*, 48(sup1):74–79, 2010.

[66] A. Barreiro, A.J.C. Crespo, J.M. Domínguez, and M. Gómez-Gesteira. Smoothed Particle Hydrodynamics for coastal engineering problems. *Computers & Structures*, 120(0):96–106, 2013.

[67] Alexis Hérault, Giuseppe Bilotta, Annamaria Vicari, Eugenio Rustico, and Ciro Del Negro. Numerical simulation of lava flow using a GPU SPH model. *Annals of Geophysics*, 54(5), 2011.

[68] Øystein Krog. GPU-based Real-Time Snow Avalanche Simulations. Master's thesis, Norwegian University of Science and Technology, Jun 2010.

[69] E. Rustico, G. Bilotta, A. Hérault, C. Del Negro, and G. Gallo. Smoothed Particle Hydrodynamics simulations on multi-GPU systems. In *20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*.

[70] Alejandro C. Crespo, Jose M. Domínguez, Anxo Barreiro, Moncho Gómez-Gesteira, and Benedict D. Rogers. GPUs, a New Tool of Acceleration in CFD: Efficiency and Reliability on Smoothed Particle Hydrodynamics Methods. *PLoS ONE*, 6(6):e20685, 06 2011.

[71] M. Gómez-Gesteira, B.D. Rogers, A.J.C. Crespo, R.A. Dalrymple, M. Narayanaswamy, and J.M. Domínguez. SPHysics – development of a free-surface fluid solver – Part 1: Theory and formulations. *Computers & Geosciences*, 48(0):289–299, 2012.

[72] M. Gómez-Gesteira, A.J.C. Crespo, B.D. Rogers, R.A. Dalrymple, J.M. Domínguez, and A. Barreiro. SPHysics – development of a free-surface fluid solver – Part 2: Efficiency and test cases. *Computers & Geosciences*, 48(0):300–307, 2012.

[73] J.M. Domínguez, A.J.C. Crespo, D. Valdez-Balderas, B.D. Rogers, and M. Gómez-Gesteira. New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. *Computer Physics Communications*, 184(8):1848–1860, 2013.

[74] Kitware. Paraview - open source scientific visualization, Jan 2013. http://www.paraview.org/.

[75] Kitware. File Formats for VTK Version 4.2, January 2013. http://www.vtk.org/VTK/img/file-formats.pdf.

[76] J. M. Domínguez, A. J. C. Crespo, M. Gómez-Gesteira, and J. C. Marongiu. Neighbour lists in smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 67(12):2026–2042, 2011.

[77] L. Hernquist and N. Katz. TREESPH - A unification of SPH with the hierarchical tree method. *Astrophysical Journal Supplement Series*, 70:419–446, June 1989.

[78] Simon Green. Particle Simulation using CUDA, Jul 2012. http://docs.nvidia.com/cuda/samples/5_Simulations/particles/doc/particles.pdf.

[79] Loup Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159:98–103, Jul 1967.

[80] Joseph P. Morris, Patrick J. Fox, and Yi Zhu. Modeling Low Reynolds Number Incompressible Flows Using SPH. *Journal of Computational Physics*, 136(1):214 – 226, 1997.

[81] Joseph J. Monaghan, Herbert E. Huppert, and M. Grae Worster. Solidification using smoothed particle hydrodynamics. *Journal of Computational Physics*, 206(2):684–705, July 2005.

[82] Novak Zuber, Gary E. Wilson, Mamoru Ishii, Wolfgang Wulff, B. E. Boyack, A. E. Dukler, P. Griffith, J. M. Healzer, R. E. Henry, J. R. Lehner, S. Levy, F. J. Moody, M. Pilch, B. R. Sehgal, B.W Spencer, T.G Theofanous, and J. Valente. An integrated structure and scaling methodology for severe accident technical issue resolution: Development of methodology. *Nuclear Engineering and Design*, 186(1-2):1–21, 1998.

[83] Research Reactor Division. Table 4.4-3 heat transfer data for HFIR 85-mw operation. Technical Report ORNL/SAR-2344 rev.8, ORNL, 2011.

[84] V. Khane, P. K. Jain, and J. D. Freels. Thermal Safety Assessment of LEU Conversion of ORNL's High Flux Isotope Reactor. In *ANS Winter Meeting*, 2012.

[85] Research Reactor Division. Table 4.4-9 HFIR relative power-density distribution. Technical Report ORNL/SAR-2344 rev.8, ORNL, 2011.

[86] John E. Hatch. *Aluminum Properties and Physical Metallurgy.* American Society for Metals, 1984.

[87] MatWeb. Aluminum 6061-O, Feb 2012. http://www.matweb.com/search/datasheet.aspx?MatGUID=626ec8cdca604f1994be4fc2bc6f7f63/.

Appendix

Appendix A

SPH Heat Conduction

The heat conduction equation is

$$\rho c_p \frac{\partial T}{\partial t} = \nabla k \cdot \nabla T \qquad \text{in } \Omega$$

Model of heat conduction based on SPH [41] is

$$c_{p,i} \frac{dT_i}{dt} = \sum_j \frac{4m_j}{\rho_i \rho_j} \frac{k_i k_j}{k_i + k_j} T_{ij} \frac{\mathbf{r}_{ij} \cdot \nabla W}{|\mathbf{r}_{ij}|} + Q_i$$

This conduction model was coded in CUDA for the GPU. This code only contained conduction and no fluid motion is considered. The purpose of this approach was to maintain simple and direct development of conduction in the GPU environment. This lead to a coding time of a few days and after some debugging the solution produced the proper behavior of diffusion on a square domain (though the code is 3D). An example solution of this is shown in the left figure below. The code worked but the comparison against an exact solution showed the results were wrong as shown in the right figure. For a problem with isothermal boundary conditions $T(t)\big|_{\partial\Omega} = 0$ and initial condition $T = sin(\pi x) sin(\pi y) sin(\pi z)$ the exact solution is

$$T = e^{-3\pi^2 \alpha t} sin(\pi x) sin(\pi y) sin(\pi z)$$

The magnitude of diffusion was often over/under estimated, depending on changes to the variables. This problem was costing time such that changes were made to reduced the code to a 2D version but this did not fix the situation. Furthermore, normalization constants for the Wendland kernel were recalculated for all dimensions but no error was found between the code and calculated constants. Then the kernel function was changed from Wendland to cubic spline (a commonly used kernel) but this did not resolve the problem. None of the efforts changed the result nor showed signs of improvement.

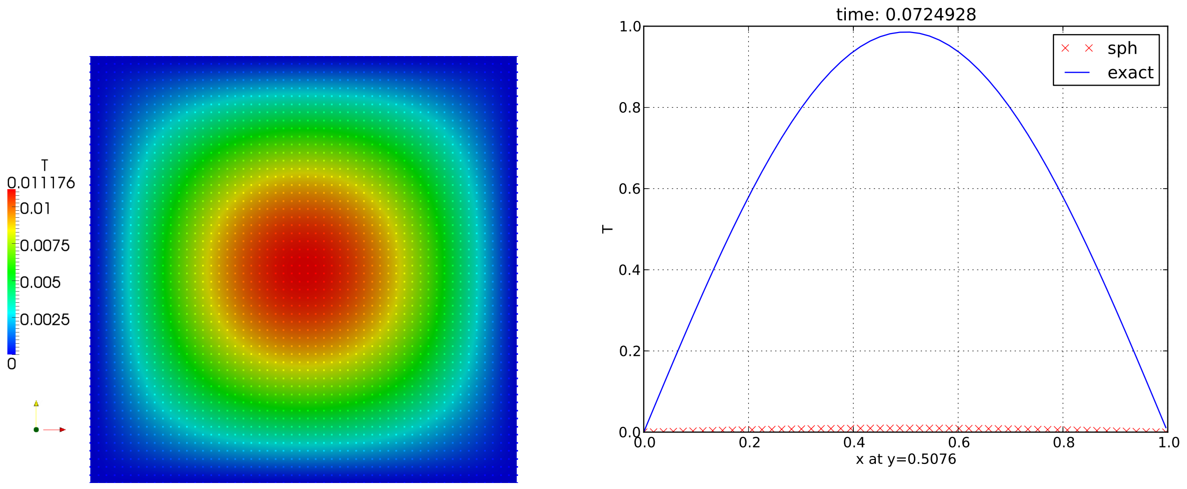

Figure A.1: 2D SPH conduction solution of isothermal boundary

Appendix B

SPH Heat Conduction Error

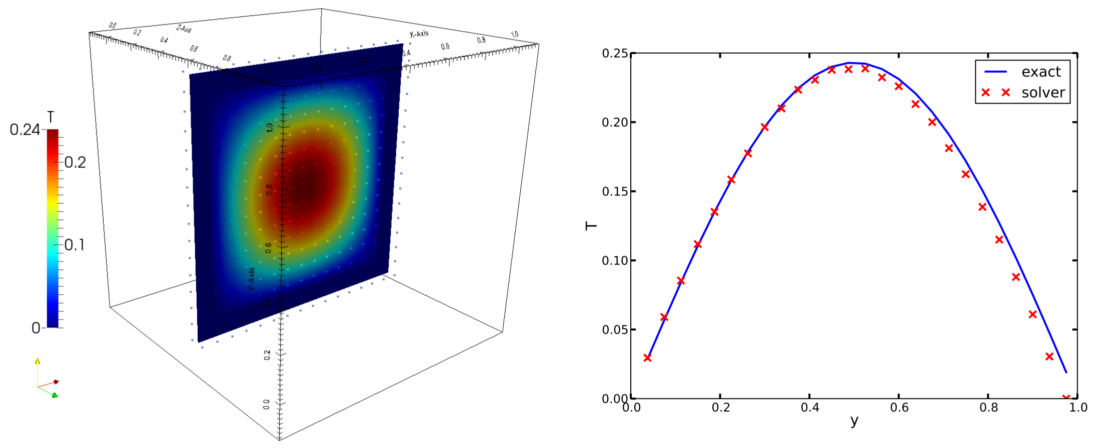

Figure B.1: Error 0.15

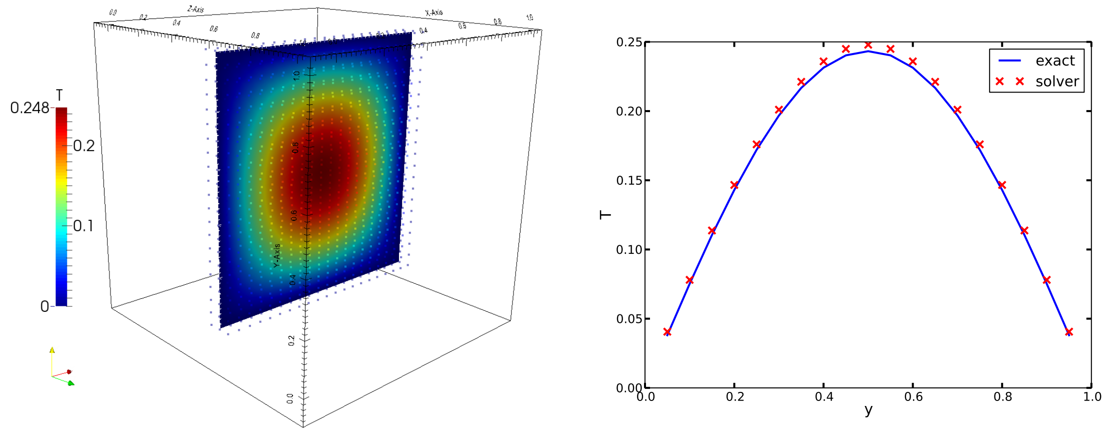

Figure B.2: Error 0.1

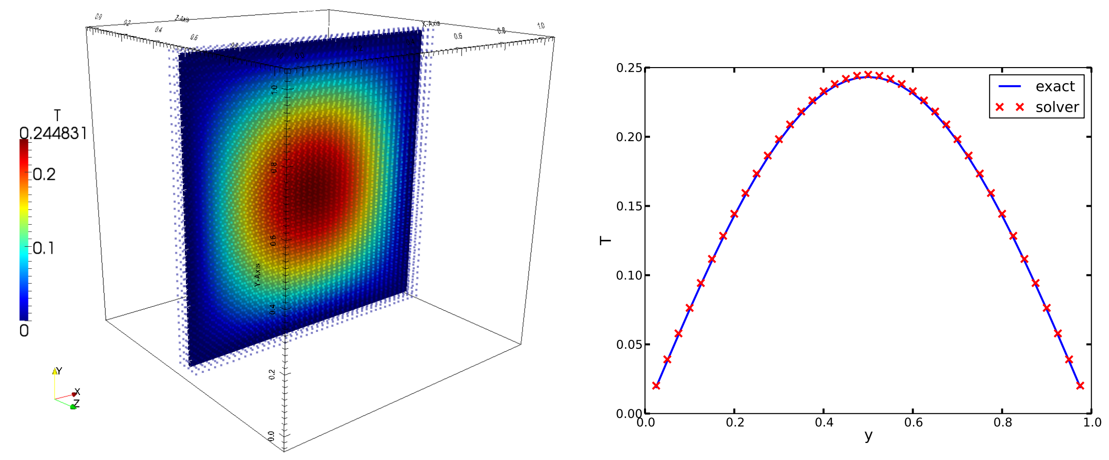

Figure B.3: Error 0.05

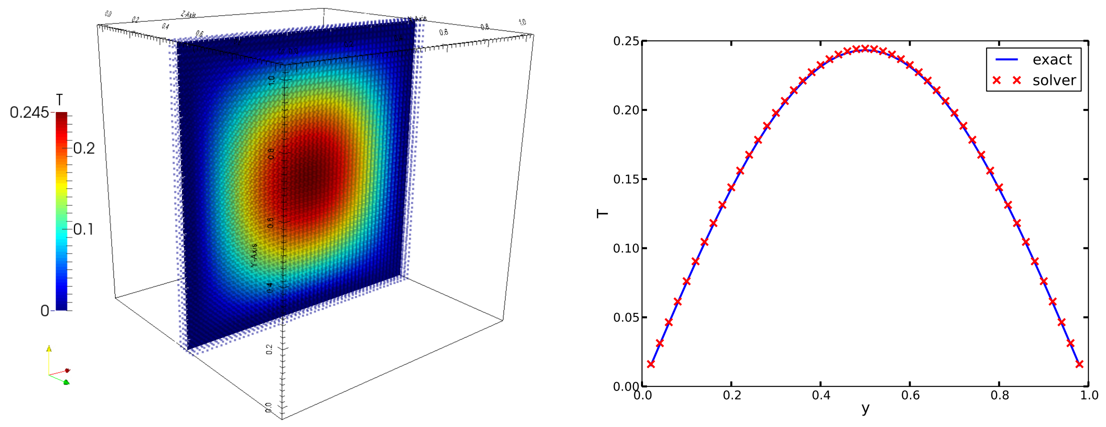

Figure B.4: Error 0.04

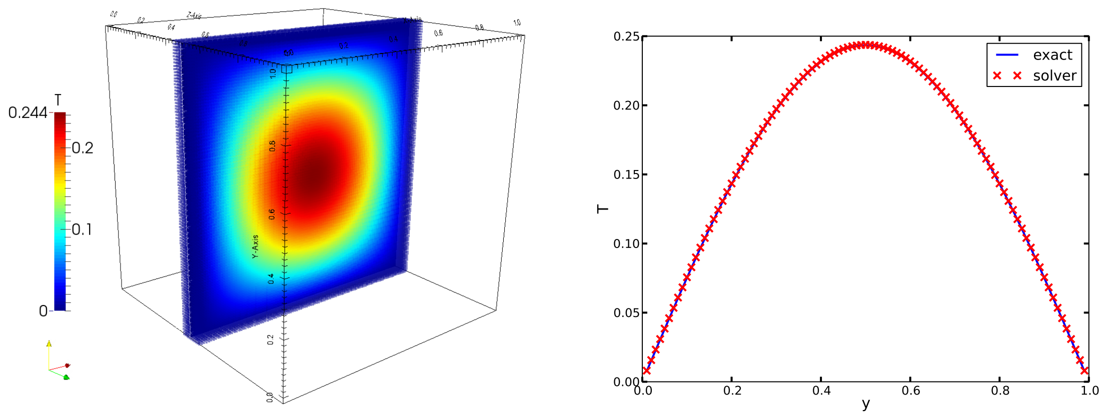

Figure B.5: Error 0.02

Appendix C

SPH Phase Change Error

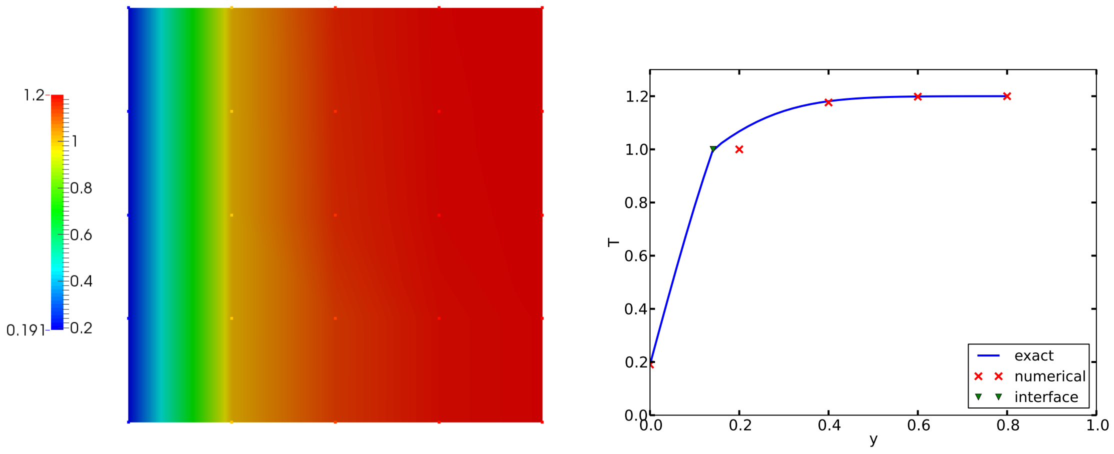

Figure C.1: Error 0.2

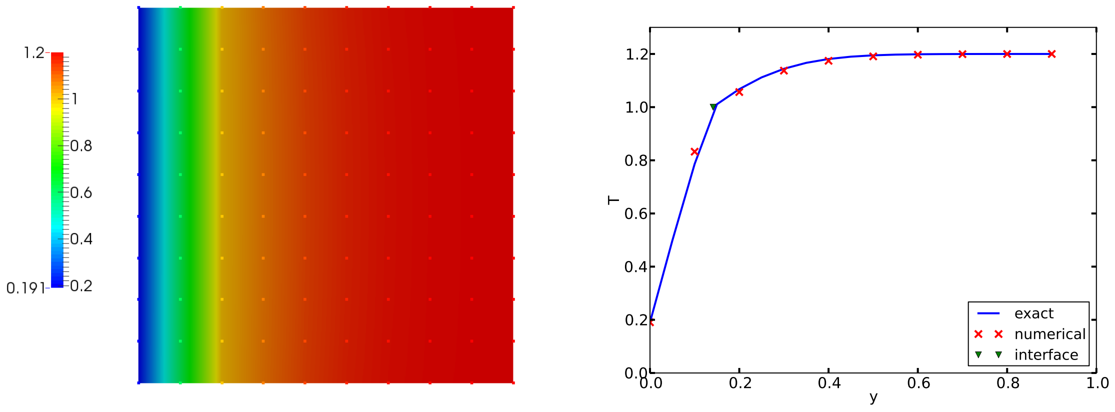

Figure C.2: Error 0.1

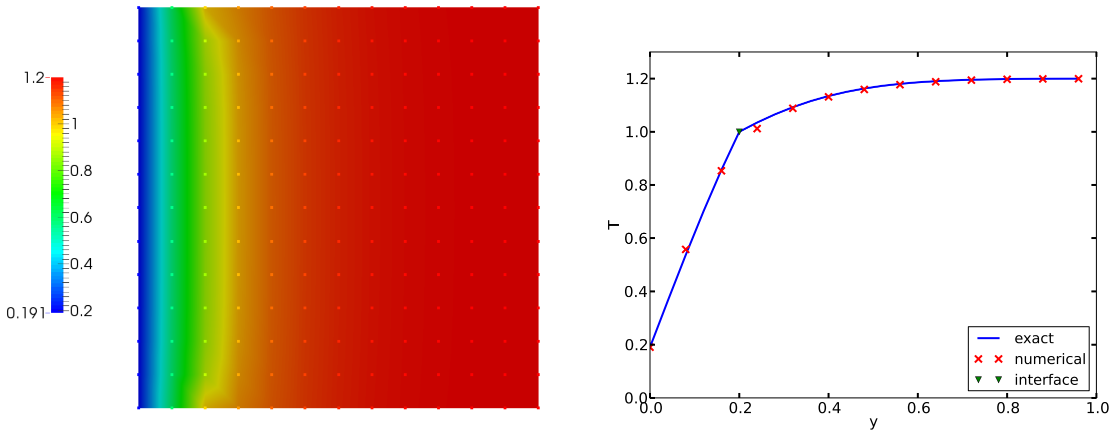

Figure C.3: Error 0.08

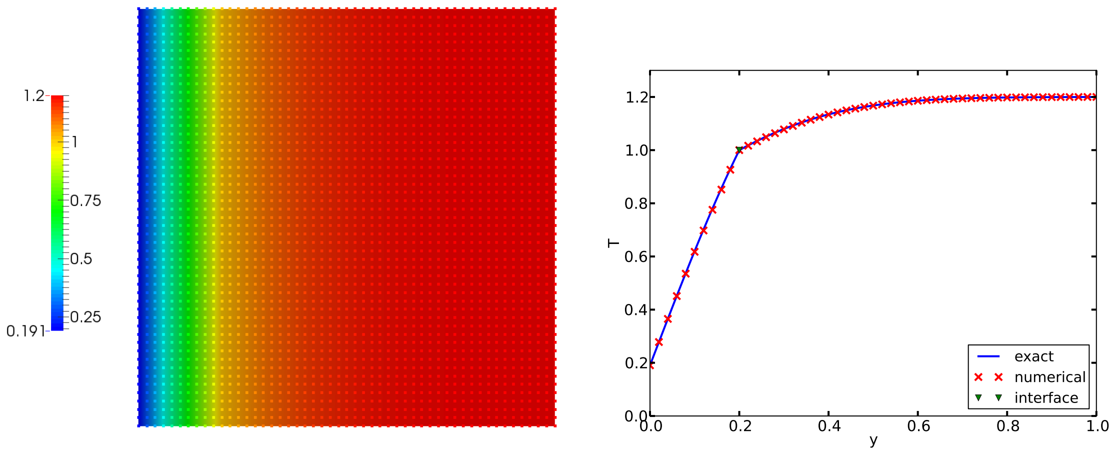

Figure C.4: Error 0.02

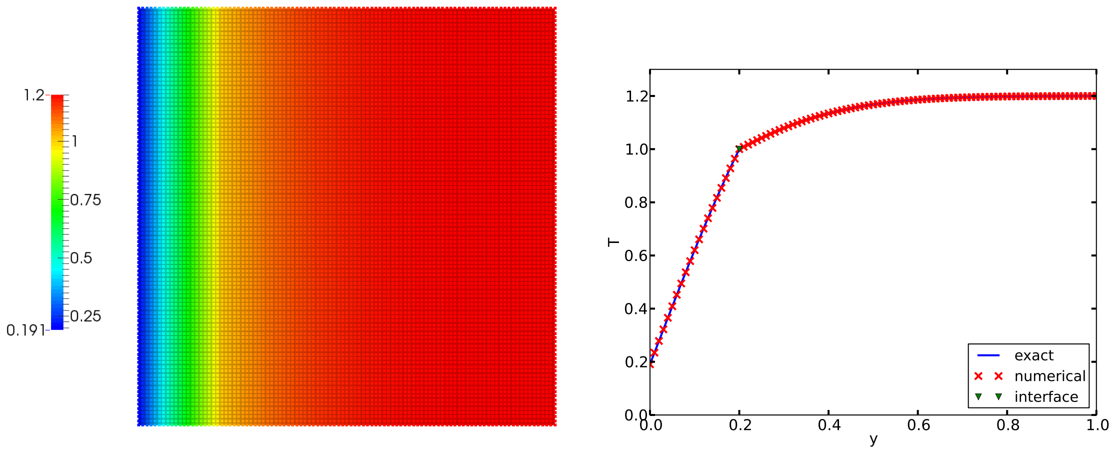

Figure C.5: Error 0.01

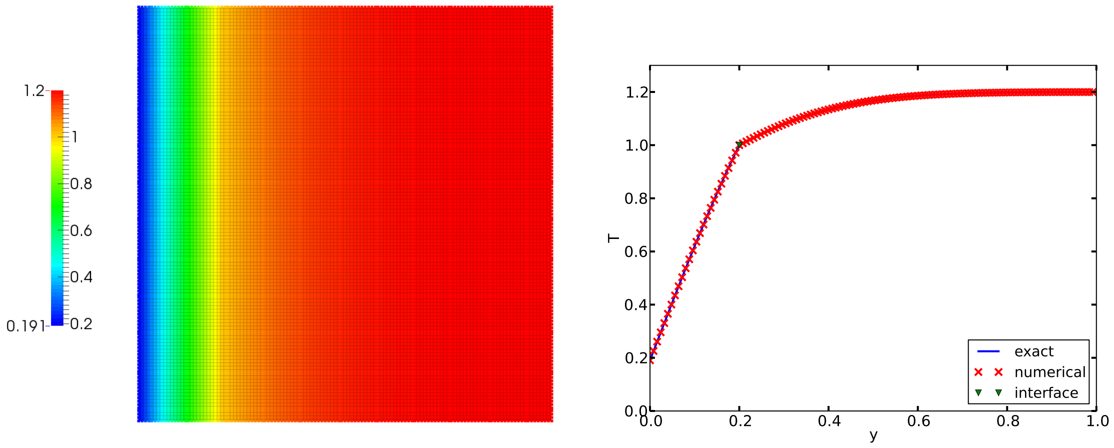

Figure C.6: Error 0.008

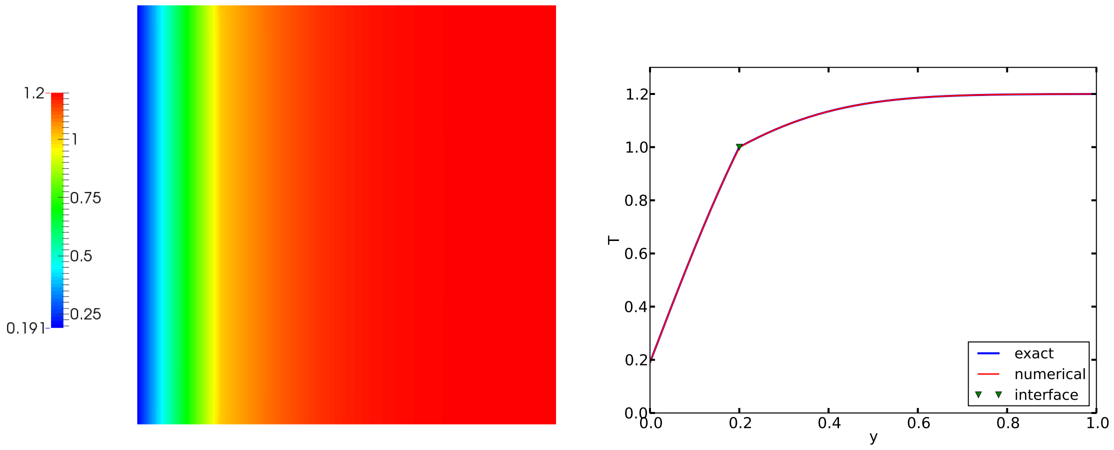

Figure C.7: Error 0.003

Appendix D

Code Structure

The current code (as of Nov. 2013) is 11398 lines of C/C++ and CUDA. Some parts of the code can and in practice should use libraries beyond the C++ Standard Template Library. Implementation of VTK can use VTK's API for handling the solution output. Large aspects of parsing text files can greatly benefit from the use of the Boost library. The use of these libraries puts the burden of maintaining these dependencies. As such, the current code has implemented the basic necessary features.

An overview of the key code structure is shown in Figure D.1 and Figure D.2. These represent the host and device interfaces. The host (CPU) side handles the initial problem construction, input setting and material properties parsing, and data writing as shown in Figure D.1. STL dependencies are not shown in Figure D.1. There are also other supporting code necessary to achieve the main functions such as exception handling. Similarly, the sphapi implements the melt model on the GPU and the numerous supporting function and kernels.

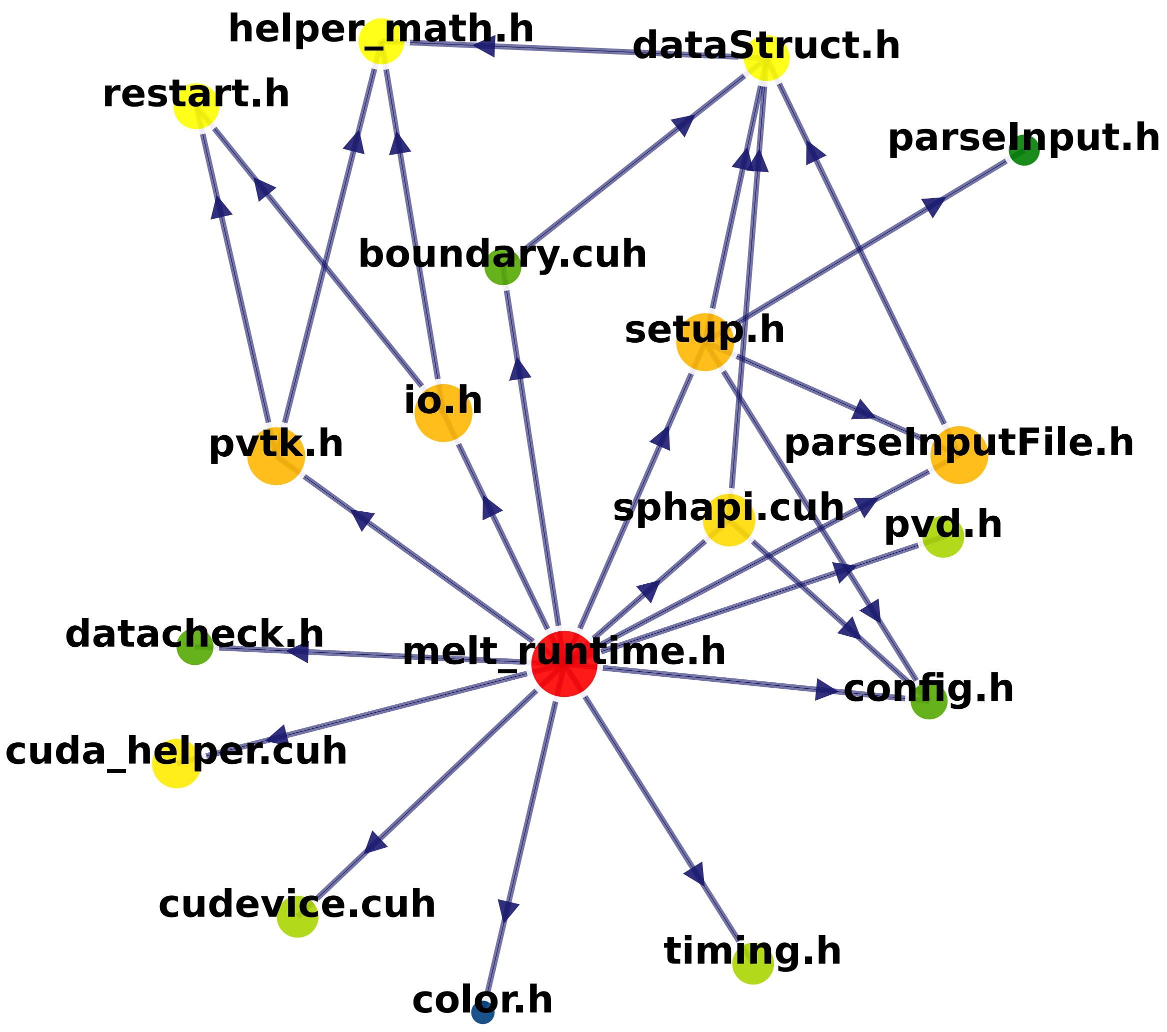

Figure D.1: Code structure for melt

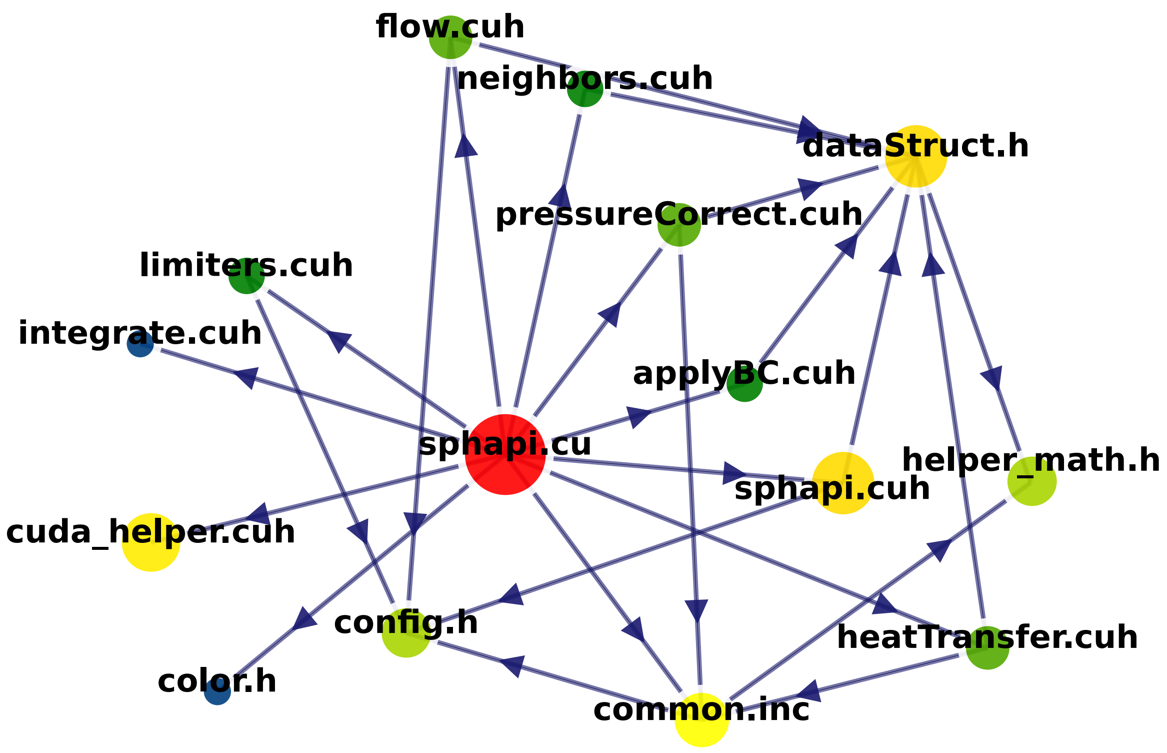

Figure D.2: Code structure

Vita

Hiraku Nakamura grew up in Japan. He went to Virginia Tech and completed a B.S. in Aerospace Engineering in 2008. He then attended the University of Tennessee, Knoxville majoring in Nuclear Engineering and received his M.S. in 2010. He then started his Ph.D. work in 2012.