ON THE MODELING, SIMULATION, AND VISUALIZATION OF MANY-BODY DYNAMICS PROBLEMS WITH FRICTION AND CONTACT

By
Toby D. Heyn

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
(Mechanical Engineering)

at the
UNIVERSITY OF WISCONSIN–MADISON
2013

Date of final oral examination: 05/06/2013

The dissertation is approved by the following members of the Final Oral Committee:

Dan Negrut, Associate Professor, Mechanical Engineering
Stephen Wright, Professor, Computer Science
Nicola J. Ferrier, Professor, Mechanical Engineering
Neil A. Duffie, Professor, Mechanical Engineering
Krishnan Suresh, Associate Professor, Mechanical Engineering
Mihai Anitescu, Professor, Statistics, University of Chicago
To my (growing) family
ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Dan Negrut, for his guidance and support. His dedication and commitment continue to inspire me. I would also like to thank Professor Alessandro Tasora and Dr. Mihai Anitescu for their assistance throughout this work. I am also grateful for the expertise of the other committee members, and the assistance, friendship, and conversation of my colleagues in the Simulation-Based Engineering Laboratory. Finally, I am so grateful to my family. The unconditional love, support, and patience of my wife, parents, and sisters means so much to me.
TABLE OF CONTENTS

Page

LIST OF TABLES ................................................................. vi

LIST OF FIGURES ............................................................... vii

ABSTRACT ........................................................................ xiii

1 Introduction ................................................................. 1
  1.1 Background ................................................................. 2
  1.2 Motivation ................................................................. 3
  1.3 Document Overview .................................................. 5
  1.4 Specific Contributions ............................................... 7

2 Many-Body Dynamics ..................................................... 9
  2.1 DVI Formulation ......................................................... 9
    2.1.1 Bilateral Constraints ............................................ 10
    2.1.2 Unilateral Constraints ......................................... 10
    2.1.3 Overall Model .................................................... 13
    2.1.4 Time-Stepping Scheme ....................................... 14
    2.1.5 Preliminaries and Important Properties .................. 16
    2.1.6 On the Lack of Uniqueness in the DVI Solution ........ 27
  2.2 DEM Formulation ....................................................... 28
    2.2.1 Normal Force Model ......................................... 28
    2.2.2 Tangential Force Model ...................................... 29
    2.2.3 Overall Method for DEM .................................... 30

3 Parallel Computing ....................................................... 32
  3.1 Motivation and Background ....................................... 32
  3.2 GPU Computing ....................................................... 34
    3.2.1 GPU Hardware Structure ................................... 35
    3.2.2 GPU Software Structure .................................... 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>MPI Parallelism</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>OpenMP Parallelism</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Iterative Methods for DVI Formulation</td>
<td>42</td>
</tr>
<tr>
<td>4.1</td>
<td>Jacobi and Gauss-Seidel</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Krylov Methods</td>
<td>45</td>
</tr>
<tr>
<td>4.2.1</td>
<td>MINRES</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Gradient-Projected Minimum Residual Method (Frictionless Problems)</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>Accelerated Projected Gradient Descent</td>
<td>51</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Adaptive Step Size</td>
<td>53</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Adaptive Restart</td>
<td>54</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Fall-Back</td>
<td>55</td>
</tr>
<tr>
<td>4.4.4</td>
<td>APGD Algorithm</td>
<td>55</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Notes on the Performance of APGD</td>
<td>57</td>
</tr>
<tr>
<td>4.5</td>
<td>Primal-Dual Interior Point Method</td>
<td>61</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Application to CCP</td>
<td>65</td>
</tr>
<tr>
<td>4.5.2</td>
<td>PDIP Algorithm</td>
<td>68</td>
</tr>
<tr>
<td>4.6</td>
<td>On the Choice of Stopping Criteria</td>
<td>69</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Description of Residuals</td>
<td>69</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Comparison of Residuals for Stacking Problems</td>
<td>71</td>
</tr>
<tr>
<td>4.7</td>
<td>Numerical Experiments</td>
<td>95</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Single-Step Convergence Experiments</td>
<td>96</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Simulation Performance Experiments</td>
<td>105</td>
</tr>
<tr>
<td>5</td>
<td>Handling Large Many-Body Dynamics Problems within the Discrete Element Method Framework</td>
<td>119</td>
</tr>
<tr>
<td>5.1</td>
<td>Algorithmic Flow</td>
<td>119</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Domain Splitting</td>
<td>120</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Data Structures and Communication Pathways</td>
<td>120</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Collision Detection</td>
<td>121</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Force Computation</td>
<td>124</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Intermediate Communication</td>
<td>125</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Dynamics and Numerical Integration</td>
<td>126</td>
</tr>
<tr>
<td>5.1.7</td>
<td>Final Communication and Synchronization</td>
<td>127</td>
</tr>
<tr>
<td>5.2</td>
<td>Numerical Experiments</td>
<td>127</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Scaling Analysis</td>
<td>128</td>
</tr>
<tr>
<td>Page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>6  Implementation Aspects that Support Parallel Computing</strong></td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>6.1 Overview of Chrono</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>6.2 Computational Flow</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>6.3 Parallel Computing in Chrono::Rigid</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>6.3.1 GPU and CPU Parallel Computing in DVI Approach</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>6.3.2 MPI Support for DEM Approach</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td><strong>7  Validation</strong></td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>7.1 Analytical Validation</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>7.1.1 Block on Incline</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>7.1.2 Sphere on Incline</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>7.1.3 Stack of Spheres</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>7.2 Experimental Validation</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>7.2.1 Mass Flow Rate</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>7.2.2 Ball Drop</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td><strong>8  Demonstration of Technology</strong></td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>8.1 Mars Rover</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>8.2 Simulation 1: Two Million Terrain Bodies</td>
<td>182</td>
<td></td>
</tr>
<tr>
<td>8.3 Simulation 2: Eight Million Terrain Bodies</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td><strong>9  Conclusions</strong></td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>9.1 Future Work</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td><strong>LIST OF REFERENCES</strong></td>
<td>191</td>
<td></td>
</tr>
<tr>
<td><strong>APPENDIX</strong> Using MATLAB to Compare Iterative Methods</td>
<td>198</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Required time to perform 1 second long simulation of 1,000 bodies in Chrono</td>
<td>107</td>
</tr>
<tr>
<td>4.2</td>
<td>Required time to perform 1 second long simulation of 2,000 bodies in Chrono</td>
<td>107</td>
</tr>
<tr>
<td>4.3</td>
<td>Required time to perform 1 second long simulation of 4,000 bodies in Chrono</td>
<td>111</td>
</tr>
<tr>
<td>4.4</td>
<td>Required time to perform 5 second long flow simulation of 1,000 bodies in Chrono</td>
<td>112</td>
</tr>
<tr>
<td>5.1</td>
<td>Simulation time and speed-up for scaling analysis.</td>
<td>129</td>
</tr>
<tr>
<td>7.1</td>
<td>Parameter values for various simulation parameters.</td>
<td>172</td>
</tr>
<tr>
<td>8.1</td>
<td>Dimensions of Mars Rover</td>
<td>182</td>
</tr>
</tbody>
</table>
DISCARD THIS PAGE
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Snapshots of preliminary Mars Rover simulation (left) and tracked vehicle simulation (right) demonstrating the need for improved terrain fidelity.</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Comparison of discrete and continuous material representations.</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Contact $i$ between two bodies $A$ and $B$.</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Projection of $\gamma_i$ onto cone $\Upsilon_i$, resulting in $\hat{\gamma}_i$.</td>
<td>24</td>
</tr>
<tr>
<td>2.3</td>
<td>Projection for single friction cone: (top) definition of projection, (middle) geometry of projection, (bottom) examples of projection.</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Two spherical bodies in contact.</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>State of system used for demonstrating performance of APGD (1,000 bodies; 3,692 Contacts).</td>
<td>57</td>
</tr>
<tr>
<td>4.2</td>
<td>Objective function and residual values for APGD with and without fall-back.</td>
<td>58</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of objective function, residual, and step length for APGD with and without adaptive restarting.</td>
<td>59</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison of objective function, residual, and step length for APGD with and without adaptive step size.</td>
<td>60</td>
</tr>
<tr>
<td>4.5</td>
<td>Test problems for comparison of residuals</td>
<td>72</td>
</tr>
<tr>
<td>4.6</td>
<td>Objective function value for various iterative methods applied to 32 body stacking problem.</td>
<td>74</td>
</tr>
<tr>
<td>4.7</td>
<td>Residual $r_1$ for various iterative methods applied to 32 body stacking problem.</td>
<td>75</td>
</tr>
<tr>
<td>4.8</td>
<td>Residual $r_2$ for various iterative methods applied to 32 body stacking problem.</td>
<td>76</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>Residual $r_3$ for various iterative methods applied to 32 body stacking problem.</td>
<td>77</td>
</tr>
<tr>
<td>4.10</td>
<td>Residual $r_4$ for various iterative methods applied to 32 body stacking problem.</td>
<td>78</td>
</tr>
<tr>
<td>4.11</td>
<td>Objective function value for various iterative methods applied to 64 body stacking problem.</td>
<td>79</td>
</tr>
<tr>
<td>4.12</td>
<td>Residual $r_1$ for various iterative methods applied to 64 body stacking problem.</td>
<td>81</td>
</tr>
<tr>
<td>4.13</td>
<td>Residual $r_2$ for various iterative methods applied to 64 body stacking problem.</td>
<td>82</td>
</tr>
<tr>
<td>4.14</td>
<td>Residual $r_3$ for various iterative methods applied to 64 body stacking problem.</td>
<td>83</td>
</tr>
<tr>
<td>4.15</td>
<td>Residual $r_4$ for various iterative methods applied to 64 body stacking problem.</td>
<td>84</td>
</tr>
<tr>
<td>4.16</td>
<td>Objective function value for various iterative methods applied to 128 body stacking problem.</td>
<td>85</td>
</tr>
<tr>
<td>4.17</td>
<td>Residual $r_1$ for various iterative methods applied to 128 body stacking problem.</td>
<td>86</td>
</tr>
<tr>
<td>4.18</td>
<td>Residual $r_2$ for various iterative methods applied to 128 body stacking problem.</td>
<td>87</td>
</tr>
<tr>
<td>4.19</td>
<td>Residual $r_3$ for various iterative methods applied to 128 body stacking problem.</td>
<td>88</td>
</tr>
<tr>
<td>4.20</td>
<td>Residual $r_4$ for various iterative methods applied to 128 body stacking problem.</td>
<td>89</td>
</tr>
<tr>
<td>4.21</td>
<td>Objective function value for various iterative methods applied to 256 body stacking problem.</td>
<td>90</td>
</tr>
<tr>
<td>4.22</td>
<td>Residual $r_1$ for various iterative methods applied to 256 body stacking problem.</td>
<td>91</td>
</tr>
<tr>
<td>4.23</td>
<td>Residual $r_2$ for various iterative methods applied to 256 body stacking problem.</td>
<td>92</td>
</tr>
<tr>
<td>4.24</td>
<td>Residual $r_3$ for various iterative methods applied to 256 body stacking problem.</td>
<td>93</td>
</tr>
<tr>
<td>4.25</td>
<td>Residual $r_4$ for various iterative methods applied to 256 body stacking problem.</td>
<td>94</td>
</tr>
<tr>
<td>4.26</td>
<td>System configurations for single-step convergence tests for frictionless problems.</td>
<td>97</td>
</tr>
<tr>
<td>4.27</td>
<td>Convergence of iterative solvers for single time step of simulation with 1,000 bodies and 4,192 contacts (frictionless).</td>
<td>99</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.28</td>
<td>Convergence of iterative solvers for single time step of simulation with 2,000 bodies and 8,622 contacts (frictionless).</td>
<td>100</td>
</tr>
<tr>
<td>4.29</td>
<td>Convergence of iterative solvers for single time step of simulation with 4,000 bodies and 17,510 contacts (frictionless).</td>
<td>101</td>
</tr>
<tr>
<td>4.30</td>
<td>System configurations for single-step convergence tests for frictional problems.</td>
<td>102</td>
</tr>
<tr>
<td>4.31</td>
<td>Convergence of iterative solvers for single time step of simulation with 1,000 bodies and 3,872 contacts (with friction).</td>
<td>103</td>
</tr>
<tr>
<td>4.32</td>
<td>Convergence of iterative solvers for single time step of simulation with 2,000 bodies and 7,931 contacts (with friction).</td>
<td>104</td>
</tr>
<tr>
<td>4.33</td>
<td>Initial configurations for test simulations of settled spheres</td>
<td>106</td>
</tr>
<tr>
<td>4.34</td>
<td>Performance of various iterative methods for test with 1,000 spheres</td>
<td>108</td>
</tr>
<tr>
<td>4.35</td>
<td>Performance of various iterative methods for test with 2,000 spheres</td>
<td>109</td>
</tr>
<tr>
<td>4.36</td>
<td>Performance of various iterative methods for test with 4,000 spheres</td>
<td>110</td>
</tr>
<tr>
<td>4.37</td>
<td>Speedup of APGD compared to JACOBI and SOR for various tolerance values.</td>
<td>112</td>
</tr>
<tr>
<td>4.38</td>
<td>Schematic of boundary geometry for flowing spheres simulation test.</td>
<td>113</td>
</tr>
<tr>
<td>4.39</td>
<td>Snapshots from flowing sphere test problem using APGD solver</td>
<td>114</td>
</tr>
<tr>
<td>4.40</td>
<td>Performance of various iterative methods for flowing sphere test with 1,000 spheres</td>
<td>116</td>
</tr>
<tr>
<td>4.41</td>
<td>Computation time for JACOBI, SOR, and APGD for various friction coefficients.</td>
<td>117</td>
</tr>
<tr>
<td>4.42</td>
<td>Speedup of APGD compared to JACOBI and SOR for various friction coefficients.</td>
<td>118</td>
</tr>
<tr>
<td>5.1</td>
<td>Original body geometry (left), sphere-set approximation (right).</td>
<td>123</td>
</tr>
<tr>
<td>5.2</td>
<td>Example of triangle, circumcircle, and three different spheres demonstrating effect of center ratio.</td>
<td>124</td>
</tr>
<tr>
<td>5.3</td>
<td>Snapshot of scaling test simulation, showing case with 4 sub-domains. Bodies are colored by sub-domain, and shared bodies are shown in white. This frame represents the state of the system at t = 16 s.</td>
<td>128</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>5.4</td>
<td>Speed-up of simulation with increasing number of sub-domains.</td>
<td>130</td>
</tr>
<tr>
<td>7.1</td>
<td>The block is released from rest on an inclined plane.</td>
<td>143</td>
</tr>
<tr>
<td>7.2</td>
<td>X-direction position and velocity results for block sliding on incline ($\mu = 0.36 &lt; \mu_c$).</td>
<td>145</td>
</tr>
<tr>
<td>7.3</td>
<td>Y-direction position and velocity results for block sliding on incline ($\mu = 0.36 &lt; \mu_c$).</td>
<td>146</td>
</tr>
<tr>
<td>7.4</td>
<td>Z-direction position and velocity results for block sliding on incline ($\mu = 0.36 &lt; \mu_c$).</td>
<td>147</td>
</tr>
<tr>
<td>7.5</td>
<td>X-direction position and velocity results for block sticking on incline ($\mu = 0.37 &gt; \mu_c$).</td>
<td>149</td>
</tr>
<tr>
<td>7.6</td>
<td>Y-direction position and velocity results for block sticking on incline ($\mu = 0.37 &gt; \mu_c$).</td>
<td>150</td>
</tr>
<tr>
<td>7.7</td>
<td>Z-direction position and velocity results for block sticking on incline ($\mu = 0.37 &gt; \mu_c$).</td>
<td>151</td>
</tr>
<tr>
<td>7.8</td>
<td>Largest magnitude error in x-direction position and velocity results for block on incline.</td>
<td>152</td>
</tr>
<tr>
<td>7.9</td>
<td>Largest magnitude error in y-direction position and velocity results for block on incline.</td>
<td>153</td>
</tr>
<tr>
<td>7.10</td>
<td>Largest magnitude error in z-direction position and velocity results for block on incline.</td>
<td>154</td>
</tr>
<tr>
<td>7.11</td>
<td>The sphere is given an initial velocity parallel to the inclined plane. The sphere slides and eventually begins rolling without slip.</td>
<td>155</td>
</tr>
<tr>
<td>7.12</td>
<td>X-direction position and velocity results for sphere on inclined plane.</td>
<td>159</td>
</tr>
<tr>
<td>7.13</td>
<td>Y-direction position and velocity results for sphere on inclined plane.</td>
<td>160</td>
</tr>
<tr>
<td>7.14</td>
<td>Z-direction position and velocity results for sphere on inclined plane.</td>
<td>161</td>
</tr>
<tr>
<td>7.15</td>
<td>Y-Position error results for Seidel and APGD solvers for stack of 10 spheres.</td>
<td>163</td>
</tr>
<tr>
<td>7.16</td>
<td>Y-Velocity results for Seidel and APGD solvers for stack of 10 spheres.</td>
<td>164</td>
</tr>
<tr>
<td>7.17</td>
<td>Schematic of mass flow rate validation experiment in closed (upper) and open (lower) configurations.</td>
<td>166</td>
</tr>
<tr>
<td>7.18</td>
<td>Weight of collected granular material vs. time for 2.0 mm gap size - comparison of experimental data to simulation data for various friction coefficients.</td>
<td>167</td>
</tr>
<tr>
<td>7.19</td>
<td>Weight of collected granular material vs. time for 3.0 mm gap size.</td>
<td>168</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>7.20</td>
<td>Weight of collected granular material vs. time for 2.5 mm gap size.</td>
<td>169</td>
</tr>
<tr>
<td>7.21</td>
<td>Weight of collected granular material vs. time for 2.0 mm gap size.</td>
<td>169</td>
</tr>
<tr>
<td>7.22</td>
<td>Schematic of ball drop validation experiment showing initial and final positions of projectile.</td>
<td>171</td>
</tr>
<tr>
<td>7.23</td>
<td>Snapshots from impact simulation with $h=20$ cm and $\rho_s=0.7$ g/cm$^3$.</td>
<td>175</td>
</tr>
<tr>
<td>7.24</td>
<td>Penetration depth vs. time for a variety of simulation experiments.</td>
<td>176</td>
</tr>
<tr>
<td>7.25</td>
<td>Penetration depth vs. scaled total drop distance for a variety of simulation experiments.</td>
<td>177</td>
</tr>
<tr>
<td>7.26</td>
<td>Snapshot of the instant of deepest penetration from each simulation.</td>
<td>178</td>
</tr>
<tr>
<td>8.1</td>
<td>Schematic of Mars Rover geometry: (top) top view, (bottom) side view.</td>
<td>181</td>
</tr>
<tr>
<td>8.2</td>
<td>Snapshot from Mars Rover simulation at $t=4.41$ s, with two million terrain bodies.</td>
<td>183</td>
</tr>
<tr>
<td>8.3</td>
<td>Snapshot from Mars Rover simulation at $t=4.41$ s, showing “footprint” of rover wheels with grousers in granular terrain composed of two million rigid bodies.</td>
<td>184</td>
</tr>
<tr>
<td>8.4</td>
<td>Snapshot from Mars Rover simulation at $t=1.0$ s, with over eight million terrain bodies.</td>
<td>186</td>
</tr>
</tbody>
</table>
ABSTRACT

As available memory size and computational speeds increase, engineers increasingly rely on computer simulation for design and analysis tasks, possibly in place of physical experiments. However, simulation capabilities are sometimes inadequate to capture phenomena of interest. This problem is especially evident when simulating the dynamics of many-body systems that may contain hundreds of thousands or millions of discrete rigid bodies interacting through contact, impact, or mutual constraints, such as a vehicle interacting with granular terrain like sand or gravel. To address this issue, it is necessary to leverage parallel computing in order to take advantage of recent advances in computing hardware.

This thesis focuses on two approaches to improve many-body dynamics simulation capabilities, both by increasing the size of tractable problems and by increasing the solution speed. The two most commonly used formulations for handling friction and contact in many-body dynamics simulations are considered. The first, based on a penalty approach called the discrete element method (DEM), is augmented through a domain decomposition approach which can leverage discrete-memory computational clusters. The second, a constraint-based approach posed as a differential variational inequality (DVI), is enhanced through the application of a new iterative method with improved convergence properties. This document describes the development and implementation of both approaches, and demonstrates the resulting simulation capabilities through a set of numerical experiments and validation simulations. The new iterative method for the DVI formulation is based on an accelerated projected gradient descent method with adaptive step size and restarting.
Numerical experiments show a speedup of 5-20 times when comparing the new method to existing methods.

The new implementation is subsequently validated against both analytical and experimental tests. Three simple scenarios are used to validate the new method and its implementation against analytical results, including the motion of a block sliding down an incline, a sphere sliding and rolling down an incline, and a vertical stack of spheres. Two validation exercises are also performed against experimental data. First, the mass flow rate of granular material flowing out of a container is simulated and compared to experimental results. Second, the penetration depth of spherical projectiles impacting granular media is simulated and compared to empirical expressions based on experimental observations. In all cases, the simulation results compare favorably with expectations.

Finally, the capabilities of the new simulation methods are demonstrated through the simulation of a large, complex system. The application of interest is a wheeled vehicle based on the Mars Exploration Rover Opportunity that is operating on granular terrain. The dynamics of such a vehicle are simulated, demonstrating the improved capabilities of the new technology.
Chapter 1

Introduction

Engineers commonly rely on prototypes and physical testing when performing design and analysis tasks. Unfortunately, such work can be expensive and time consuming. It may be difficult or impossible to experimentally measure certain physical parameters, such as the interaction force between two individual grains of sand. Because computational hardware continues to advance in terms of both processing speed and memory size, a trend is growing in which computer simulation is used to augment and, in some cases, replace large amounts of experimental work. With increasing computational power, engineers are able to perform faster, larger, and more accurate simulations. Computer simulation has several advantages over physical experiments. Via simulation, engineers may study a range of parameter values that would prove too costly or too dangerous to study experimentally. Additionally, computer simulation can produce representative data that experimental measurements could never achieve. Experimental insights are limited by the position, fidelity, and number of sensors, whereas a simulation inherently tracks the state of every component of the system. For example, simulation can generate the set of forces acting between all the individual bodies in a collection of granular material, which could not be effectively measured experimentally.

Current simulation capabilities are sometimes inadequate to capture phenomena of interest. This problem is especially evident when simulating the dynamics of many-body systems, which may contain hundreds of thousands or millions of discrete rigid bodies interacting through contact, impact, and constraints. For example, the simulation of one cubic meter of sand would require the solution of a system of approximately 18 billion equations per time step, which is beyond
the scope of current capabilities. The ability to solve such large problems will require significant improvements in terms of both algorithms and implementation.

1.1 Background

Currently, there are two popular approaches for handling friction and contact in many-body dynamics simulations. The discrete element method (DEM) [1] is a penalty method where interaction force is computed based on the kinematics of the interaction, some representative parameters, and an empirical force law. The other method, based on a differential variational inequality (DVI) [2], enforces non-penetration of rigid bodies via a constraint-based approach. In the DVI method, a cone-constrained quadratic optimization problem must be solved at each time step of the simulation, where the unknowns are the normal and frictional contact forces between interacting bodies. These two methods are sometimes referred to as the penalty approach and the Lagrange multiplier approach, respectively. However, in the remainder of this document the methods will be referred to as the DEM and DVI approaches.

Both of these approaches have advantages and disadvantages. The DEM approach is more “tunable”, with many choices available for force models and parameter values. However, the DEM approach always allows some amount of penetration between bodies, and may require very small time steps for stability reasons. These issues are tied directly to the DEM model, which computes contact forces based on fictitious spring-damper elements between colliding bodies. High stiffness is needed to minimize interpenetration, but this high stiffness drives the natural frequency of the system up, and drives the allowable time step down. The DVI approach has fewer parameters which must be specified, can enforce non-penetration, and can often take significantly larger time steps. However, the DVI approach uses an iterative solution process, which leads to a more difficult implementation and significantly more computation required at each time step when compared to the DEM approach.

Each of these approaches is used for different types of problems in different fields. For example, the DEM approach has been used to study rock and concrete behavior [3], powder compaction [4], and railroad ballast [5]. The DVI approach has been used to study part feeders [6], anchoring
in granular material [7], and robotic grasping [8]. Therefore, researchers would benefit most if both approaches could be improved.

1.2 Motivation

The research carried out in this work is motivated by two main considerations. First, many engineers are interested in solving problems that involve granular material, where a large number of rigid bodies interact through contact and friction. Many real-world systems contain or interact with granular material, as granular material is among the most manipulated materials in industry, second only to water [9]. For example, granular material is utilized in a variety of fields, from nanoscale powders in the pharmaceutical industry to large boulders in the construction industry. One specific motivating problem is that of a Mars rover-type vehicle moving on granular terrain. Computer simulation is a perfect fit for such a problem, because it enables engineers to perform tests that would otherwise be physically impossible or too expensive to carry out. For example, simulations could be used to predict the mobility of a Rover in low-gravity environments before an expensive launch is carried out.

The second motivating consideration is that large simulations are difficult to perform with existing simulation tools because they often demonstrate poor performance with increasing problem size. To demonstrate this, a numerical experiment was performed in [10] using one of the most commonly used commercial multi-body dynamics simulation programs. There, it was shown that the considered commercial code had difficulty handling problems with more than a few hundred contacts. This was demonstrated by performing three second long simulations in which a number of balls were dropped into a square bucket. The number of balls varied from two to 32, and the required time to complete the simulation was recorded for each case. The results demonstrated that the simulation time scaled with the square of the number of bodies in the simulation. With quadratic scaling, it would take prohibitively long to perform a simulation of a system with even thousands of bodies in this commercial software package.

Based on these considerations, preliminary simulations were performed with existing simulation methods for benchmarking purposes. Figure 1.1 shows snapshots from these simulations,
Figure 1.1: Snapshots of preliminary Mars Rover simulation (left) and tracked vehicle simulation (right) demonstrating the need for improved terrain fidelity.
which consisted of a Mars Rover-type vehicle (left) and a tracked vehicle (right) moving on granular terrain composed of a few thousand rigid terrain particles. It is immediately clear that the representation of the terrain is too coarse. That is, to achieve more realism, simulations must use finer particles to represent the terrain. However, note that when the characteristic particle size is reduced by half, the required number of particles to represent a fixed amount of granular material increases by a factor of eight. In fact, it is estimated that one cubic meter of sand would contain over a billion individual grains. It is clear that improved simulation codes are needed to perform more realistic simulations.

In the past, such large problems were often tackled by approximating the granular material as a continuum. However, simulating granular material as a collection of discrete bodies, rather than approximating it as a continuum, is valuable as doing so can expose more information about the behavior of the system. For example, consider the images in Figure 1.2. Figure 1.2a shows a cross-section of the vertical stress in a semi-infinite, homogeneous, linearly elastic medium, under application of uniform pressure to a round plate. Figure 1.2b shows a cross section of the normal contact forces in a collection of granular material during an impact by a spherical body. Contacts are drawn by connecting the centers of colliding bodies, and coloring the line based on the magnitude of the normal force. From these images, it is clear that the discrete nature of granular material can be important. When a continuous approximation is used, the results are often smooth and symmetric about the loading. When a discrete representation is used, the results can be seen to be non-smooth, non-symmetric, and non-uniform, and may depend strongly on the changing configuration of the granular material. While continuum approaches are valuable for certain problems, some phenomena can only be captured by discrete models.

1.3 Document Overview

This research work focuses on developing, implementing, and validating computational methods that allow the simulation of larger many-body systems. Simulation capability is increased through the development of parallel numerical methods that scale better than current methods and
(a) Cross-section view of vertical stress in semi-infinite continuum, under application of 20 psi load.

(b) Cross-section view of normal contact forces in granular material, during impact by spherical object.

Figure 1.2: Comparison of discrete and continuous material representations.
through software solutions that leverage distributed computing. The new methods are also validated against simple tests where an analytical solution is known and against physical experiments representing more complex systems.

This document proceeds as follows. Chapter 2 provides an overview of multibody dynamics, focusing on two formulations for handling friction and contact. Chapter 3 acts as an introduction to parallel computing, describing the three most widely-used approaches to parallel programming. Chapter 4 describes the use of iterative methods to solve many-body dynamics problems posed with the DVI formulation for handling friction and contact. Specifically, new methods are proposed, and their performance is compared to existing methods through a sequence of numerical experiments. Chapter 5 focuses on a domain decomposition approach to leverage parallelism when using the DEM formulation for friction and contact. Chapter 6 describes the use of parallel computing in the implementations of the methods from the previous two chapters. Finally, to ensure that simulation results are accurate, a series of validation experiments are performed in Chapter 7. The new iterative method is compared to analytical solutions for simple scenarios, and to experimental data for more complex scenarios.

1.4 Specific Contributions

The specific contributions of the author are summarized as follows:

- Investigated new numerical methods for the Differential Variational Inequality formulation which demonstrate improved convergence properties:
  - Developed Gradient-Projected Minimum Residual (GPMINRES) method for frictionless problems
  - Extended and improved the Accelerated Projected Gradient Descent (APGD) method for frictional problems
  - Implemented APGD to run in parallel with OpenMP or GPU programming
  - Demonstrated potential of Interior Point methods for future work
• Developed and implemented domain decomposition approach for Discrete Element Method formulation:
  
  – Developed domain decomposition, inter-domain communication, and synchronization methods to perform distributed many-body dynamics simulations on computing cluster
  
  – Demonstrated scaling behavior of the implementation
  
  – Demonstrated capabilities through simulation of Mars Rover vehicle operating on granular terrain composed of over eight million terrain bodies

• Validated APGD method against analytical solutions for simple scenarios

• Validated APGD method against experimental data for two scenarios
Chapter 2

Many-Body Dynamics

This chapter introduces the concept of many-body dynamics and describes two formulations for handling frictional contact in the numerical simulation of many-body systems. Here, many-body dynamic simulation is the name given to problems beyond the common scope of multi-body dynamics. In other words, many-body systems may have hundreds of thousands of bodies or more, several orders of magnitude larger than those problems commonly referred to as multi-body. Two contrasting methods are currently being used to simulate many-body systems. The differential variational inequality (DVI) approach enforces non-penetration between rigid bodies through constraints. On the other hand, the discrete element method (DEM) relies on allowing some penetration between bodies and using a constitutive law to compute reaction forces at each contact. The benefits and drawbacks of both methods will be described in following sections along with the details of the formulations and implementations. It is important to note that both formulations are able to leverage high-performance computing to speed-up simulations or allow simulation of larger systems. In this chapter and the remainder of the document, scalars are written in italics, while matrix and vector terms are specified with bold symbols.

2.1 DVI Formulation

Consider a system of rigid bodies which may interact through contact, friction, and possibly bilateral constraints. An absolute Cartesian coordinate system will be used to define the equations of motion for the time evolution of such a system. Therefore, the generalized positions \( \mathbf{q} = [r_1^T, \epsilon_1^T, \ldots, r_{n_b}^T, \epsilon_{n_b}^T]^T \) and their time derivatives \( \dot{\mathbf{q}} = [\dot{r}_1^T, \dot{\epsilon}_1^T, \ldots, \dot{r}_{n_b}^T, \dot{\epsilon}_{n_b}^T]^T \) are used to describe
the state of the system. Here, \( r_j \) is the absolute position of the center of mass of body \( j \) and \( \epsilon_j \) is the quaternion used to represent rotation. Note that the angular velocity of body \( j \) in local coordinates, \( \dot{\omega}_j \), may be used in place of the time derivative of the rotation quaternion. Then, the vector of generalized velocities \( \dot{v} = [\dot{r}_1^T, \dot{\omega}_1^T, \ldots, \dot{r}_{nb}^T, \dot{\omega}_{nb}^T]^T \) can be easily related to \( \dot{q} \) via a linear mapping [11] given as \( \dot{q} = T(q) \dot{v} \).

Due to the rigid body assumption and the choice of centroidal reference frames, the generalized mass matrix \( M \) is constant and diagonal. Further, let \( f(t, q, v) \) be a set of generalized external forces which act on the bodies in the system. Finally, the second order differential equations which govern the time evolution of the system can be written in the matrix form \( M \ddot{v} = f(t, q, v) \) [12]. These equations can be further augmented by the application of bilateral or unilateral constraints to the rigid body system, each of which are described in the following subsections.

### 2.1.1 Bilateral Constraints

A bilateral constraint represents a kinematic relationship between two rigid bodies in the system. For example, spherical joints, prismatic joints, or revolute joints can be expressed as holonomic algebraic equations constraining the relative position and/or orientation of two bodies [13]. Let \( B \) represent a set of bilateral constraints. Then, \( g_i(q, t) = 0, i \in B \) is a set of scalar equations. Note that the number of total constraint equations depends on the type of constraints in set \( B \). For example, a revolute joint imposes five constraint equations, while a spherical constraint imposes only three constraint equations. By taking one time-derivative of the constraint equations, the velocity-level constraint equations which must also be satisfied are obtained as \( \nabla_q g_i^T T(q) \dot{v} + \frac{\partial g_i}{\partial q} = 0 \) [14].

### 2.1.2 Unilateral Constraints

The rigid body assumption implies that bodies which come into contact with one another should not penetrate. Such a condition is enforced here through unilateral constraints. To define the non-penetration constraint, a gap function, \( \Phi(q) \), must be defined for each pair of near-enough bodies. The qualifier ‘near-enough’ means that the gap function should be defined for any pair of bodies
who are in contact or who may come into contact during the next time step. This gap function should describe the minimum distance between the two bodies of interest. For convex, smooth geometries, the gap function is typically easy to define [15]. For example, in the simple case of two spheres, the gap is a function of the distance between the centers of the spheres and their radii.

When two bodies are in contact, or $\Phi(q) = 0$, a normal force acts on each of the two bodies at the contact point. When a pair of bodies is not in contact, or $\Phi(q) > 0$, no normal force exists. This captures a complementarity condition, where one of two scenarios must hold. Either the gap is positive and the normal force is exactly zero, or the gap is zero, and the normal force is positive. When a pair of bodies is in contact, friction forces may be introduced into the system through the Coulomb friction model.

Assume contact $i$ represents a contact between bodies $A$ and $B$ (see Figure 2.1). Let the contact points in the local coordinates of each body be expressed as $\bar{s}_{i,A}$ and $\bar{s}_{i,B}$ respectively. Then, let $n_i$ be the normal direction at the contact point, and let $u_i$ and $w_i$ be two vectors in the contact plane such that $n_i$, $u_i$, and $w_i$ are mutually orthonormal and defined in global coordinates. The force
associated with contact $i$ can then be decomposed into the normal component, $F_{i,N} = \gamma_{i,n} n_i$, and
the tangential component, $F_{i,T} = \gamma_{i,u} u_i + \gamma_{i,w} w_i$, where multipliers $\gamma_{i,n} > 0$, $\gamma_{i,u}$, and $\gamma_{i,w}$
represent the magnitude of the force in each direction. The friction forces are assumed to satisfy
the Coulomb friction model, which can be expressed as [16].

\[
\begin{align*}
\dot{\gamma}_{i,n} & \geq 0, \quad \Phi_i(q) \geq 0, \quad \Phi_i(q) \dot{\gamma}_{i,n} = 0, \quad (2.1) \\
\mu_i \dot{\gamma}_{i,n} & \geq \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}, \quad \|v_{i,T}\| \left(\mu_i \dot{\gamma}_{i,n} - \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}\right) = 0, \quad (2.2) \\
\langle F_{i,T}, v_{i,T} \rangle & = -\|F_{i,T}\|\|v_{i,T}\|. \quad (2.3)
\end{align*}
\]

Equation 2.1 represents the complementarity condition previously described. Equations 2.2 and 2.3 capture the relationship between the magnitude and direction of the friction force, the multipliers, and the tangential slip velocity of the contact. Specifically, the magnitude of the tangential force should be less than or equal to the friction coefficient, $\mu_i$, times the magnitude of the normal force. If the friction force is less than $\mu_i \dot{\gamma}_{i,n}$, the slip velocity should be zero. If the friction force is equal to $\mu_i \dot{\gamma}_{i,n}$, the slip velocity may be positive and the friction force should oppose the slip direction. These equations can be expressed in an equivalent manner using the maximum dissipation principle [16]. Doing so allows the Coulomb friction model to be posed as a minimization problem, which can be expressed as

\[
(\dot{\gamma}_{i,u}, \dot{\gamma}_{i,w}) = \arg \min_{\|v_{i,T}\| \leq \mu_i \dot{\gamma}_{i,n}} v_{i,T}^T (\dot{\gamma}_{i,u} u_i + \dot{\gamma}_{i,w} w_i). \quad (2.4)
\]

It can be seen that the set of possible contact forces is represented by a cone, if yet another form
of the friction force equations is considered. The friction force of the $i^{th}$ contact can be expressed as

\[
F_i = F_{i,N} + F_{i,T} = \gamma_{i,n} n_i + \gamma_{i,u} u_i + \gamma_{i,w} w_i \in \Upsilon_i, \quad (2.5)
\]

where $\Upsilon_i$ is a cone in three dimensions whose slope is $\tan^{-1} \mu_i$, i.e., $\Upsilon_i = \{[x, y, z]^T \in \mathbb{R}^3 | \sqrt{y^2 + z^2} \leq \mu_i x\}$. 
2.1.3 Overall Model

The governing differential equations are obtained by combining the rigid body dynamics equations with the unilateral and bilateral constraint equations. Consider the system at time \( t \), when a set of contacts, \( A \), and a set of bilateral constraints, \( B \), exist between bodies in the system. For contact \( i \in A \) between bodies \( A \) and \( B \), the matrix \( A_{i,p} = [n_i, u_i, w_i] \in \mathbb{R}^{3 \times 3} \) is used to represent the orientation of the contact in global coordinates, and the matrices \( A_A = A(\epsilon_A) \) and \( A_B = A(\epsilon_B) \) as the rotation matrices of bodies \( A \) and \( B \), respectively.

Then, from Equation 2.5, the force due to contact \( i \) applied to the center of mass of bodies \( A \) and \( B \), respectively, can be written as

\[
F_{i,A} = -\hat{\gamma}_{i,n} n_i - \hat{\gamma}_{i,u} u_i - \hat{\gamma}_{i,w} w_i = -A_{i,p} \hat{\gamma}_i \tag{2.6}
\]

\[
F_{i,B} = \hat{\gamma}_{i,n} n_i + \hat{\gamma}_{i,u} u_i + \hat{\gamma}_{i,w} w_i = A_{i,p} \hat{\gamma}_i. \tag{2.7}
\]

Similarly, the associated torque acting on each body is expressed in local coordinates as

\[
\bar{T}_{i,A} = \bar{s}_{i,A} \times \bar{F}_{i,A} = \bar{s}_{i,A} A_A^T F_{i,A} = -\bar{s}_{i,A} A_A^T A_{i,p} \hat{\gamma}_i = (A_{i,p} A_A \bar{s}_{i,A})^T \hat{\gamma}_i \tag{2.8}
\]

\[
\bar{T}_{i,B} = \bar{s}_{i,B} \times \bar{F}_{i,B} = \bar{s}_{i,B} A_B^T F_{i,B} = \bar{s}_{i,B} A_B^T A_{i,p} \hat{\gamma}_i = -(A_{i,p} A_B \bar{s}_{i,B})^T \hat{\gamma}_i, \tag{2.9}
\]

where the skew-symmetric cross-product matrix is defined as

\[
\bar{s} = \begin{bmatrix}
0 & -\bar{s}_z & \bar{s}_y \\
\bar{s}_z & 0 & -\bar{s}_x \\
-\bar{s}_y & \bar{s}_x & 0
\end{bmatrix}. \tag{2.10}
\]

Then, the governing differential equations, which assume the form of a differential variational inequality (DVI) problem, are written as follows [2].
\[ \dot{q} = T(q) v \] (2.11)

\[ M(q) \dot{v} = f(t, q, v) - g^T_q(q, t) \lambda + \sum_{i=1}^{N_c} (\dot{\gamma}_{i,n} D_{i,n}^T + \dot{\gamma}_{i,u} D_{i,u}^T + \dot{\gamma}_{i,w} D_{i,w}^T) \] (2.12)

\[ g(q, t) = 0 \] (2.13)

\[ 0 \leq \Phi_i(q, t) \perp \dot{\gamma}_{i,n} \geq 0 \quad i = 1, 2, \ldots, N_c \] (2.14)

\[ (\dot{\gamma}_{i,u}, \dot{\gamma}_{i,w}) = \arg \min_{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \leq \mu_i \dot{\gamma}_{i,n}} (\gamma_{i,u} v^T D_{i,u} + \gamma_{i,w} v^T D_{i,w}) \] (2.15)

Here, note that the tangent space generators \( D_i = [D_{i,n}, D_{i,u}, D_{i,w}] \in \mathbb{R}^{6nb \times 3} \) are defined as

\[ D_i^T = \begin{bmatrix} 0, \cdots, -A_{i,p}^T, A_{i,p}^T A_{i,p} \ddot{s}_i, 0, \cdots, 0, A_{i,p}^T, -A_{i,p}^T A_{i,p} B \ddot{s}_i, \cdots, 0 \end{bmatrix}, \] (2.16)

and are used to transform the contact forces from local to global frame.

The overall model is captured by Equations 2.11-2.15. Note that Equation 2.12 represents Newton’s Second Law, with the applied forces, reaction forces, and frictional forces on the right hand side of the equation. The bilateral and unilateral constraints, from Sections 2.1.1 and 2.1.2, are represented in Equations 2.13 and 2.14 respectively. Finally, the Coulomb friction model is stated in Equation 2.15 as an optimization problem.

### 2.1.4 Time-Stepping Scheme

To use this model in a computer simulation framework, the equations must be discretized to obtain an approximation of the solution at discrete instants in time. In the following, the superscript \((l)\) denotes a variable at time-step \(t^{(l)}\). For example, \(q^{(l)}\) and \(v^{(l)}\) represent the position and velocity at time-step \(t^{(l)}\) respectively. Further, \(\gamma_i = h \dot{\gamma}_i\) is the contact impulse for contact \(i\). Then, the discretized form of the model is posed as follows, where the solution is desired at the next time-step, \(t^{(l+1)} = t^{(l)} + h\) for some step size \(h\) [17].
\begin{align}
q^{(l+1)} &= q^{(l)} + hT \left(q^{(l)} \right) v^{(l+1)} \\
M \left(v^{(l+1)} - v^{(l)} \right) &= h f \left(t^{(l)}, q^{(l)}, v^{(l)} \right) - g^T_q \left(q^{(l)}, t \right) \lambda \\
&\quad + \sum_{i=1}^{N_c} \left( \gamma_{i,n} D^T_{i,n} + \gamma_{i,u} D^T_{i,u} + \gamma_{i,w} D^T_{i,w} \right) \\
&\quad + \sum_{i=1}^{N_c} \left( \gamma_{i,n} D^T_{i,n} + \gamma_{i,u} D^T_{i,u} + \gamma_{i,w} D^T_{i,w} \right) \\
1 \frac{1}{h} g \left(q^{(l)}, t \right) + g^T_q T \left(q^{(l)} \right) v^{(l+1)} + g_t &= 0 \\
0 \leq \frac{1}{h} \Phi_i \left(q^{(l)}, t \right) + D^T_{i,n} v^{(l+1)} &\perp \gamma_{i,n} \geq 0, \quad i = 1, 2, \ldots, N_c \\
\left( \gamma_{i,u}, \gamma_{i,w} \right) &= \arg \min_{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \leq \mu_i} \left( \gamma_{i,u} v^{(l+1)} T D_{i,u} + \gamma_{i,w} v^{(l+1)} T D_{i,w} \right) \\
\end{align}

In the previous equations, note the stabilization terms \(1 \frac{1}{h} g \left(q^{(l)}, t \right)\) and \(1 \frac{1}{h} \Phi_i \left(q^{(l)}, t \right)\) which help stabilize the bilateral and contact constraints respectively. This stabilized scheme has been shown to converge to the solution of the original continuous-time problem as the step-size tends to zero [18].

Methods exist for solving the above problem. For example, the friction cones can be approximated by linear faceted pyramids, leading to a linear complementarity problem (LCP) which can be solved by pivoting or simplex methods. However, these approaches in the class of direct methods can have exponential complexity in the worst-case [19]. Hence, an alternative is to introduce a relaxation to the complementarity constraints, replacing \(0 \leq \left( \frac{1}{h} \Phi_i \left(q^{(l)}, t \right) + D^T_{i,n} v^{(l+1)} \right) \perp \gamma_{i,n} \geq 0\) with the following [20]:

\begin{align}
0 \leq \left( \frac{1}{h} \Phi_i \left(q^{(l)}, t \right) + D^T_{i,n} v^{(l+1)} \right) - \mu_i \sqrt{\left(D^T_{i,u} v^{(l+1)} \right)^2 + \left(D^T_{i,w} v^{(l+1)} \right)^2} \perp \gamma_{i,n} \geq 0 \\
\end{align}

This modified scheme will shortly be shown to lead to a Cone Complementarity Problem ( CCP), which represents the first order optimality condition of a quadratic optimization problem. It can be seen from Equation 2.22 that the modification is small when \(v\) or \(\mu\) are small. Additionally, it has been shown in [18] that the solution of the modified scheme will still approach the solution of the original problem as the step-size tends to zero. An iterative method has been developed to
solve this problem (see [21, 14]). Note that Chapter 4 will apply and compare several new iterative methods for solving this CCP problem. The remaining portion of this chapter will develop some properties of the CCP problem which are important for the discussion of the iterative methods in Chapter 4.

2.1.5 Preliminaries and Important Properties

This section will provide some important derivations, proofs, and properties associated with the Cone Complementarity Problem formulation.

Derivation of CCP from Time-Stepping Formulation

First, the CCP form of the problem will be derived from the discretized equations of motion in Equations 2.17-2.21. Note that the relaxed form of the complementarity problem is used, i.e., consider Equation 2.22 in place of Equation 2.20, and this section will consider the case that includes only unilateral constraints. The associated equations of motion are reproduced here, where $D = [D_1, \ldots, D_N]$

\begin{align*}
q^{(l+1)} &= q^{(l)} + h T(q^{(l)}) v^{(l+1)} \\
M (v^{(l+1)} - v^{(l)}) &= h f(t^{(l)}, q^{(l)}, v^{(l)}) + D \gamma^{(l+1)} \\
0 &\leq \gamma^{(l+1)}_{i,n} \perp \left( \frac{1}{\bar{r}_i} \Phi_i(q^{(l)}, t) + D^T_{i,n} v^{(l+1)} \right) \\
\gamma^{(l+1)}_{i,u}, \gamma^{(l+1)}_{i,w} &= \arg \min_{\gamma_{i,u}, \gamma_{i,w} \leq \mu_i \gamma_{i,n}} \left( \gamma_{i,u} D^T_{i,u} v^{(l+1)} + \gamma_{i,w} D^T_{i,w} v^{(l+1)} \right)
\end{align*}

The goal is to show that solving the relaxed discretized equations of motion is equivalent to solving a Cone Complementarity Problem (CCP) of the form
Find $\gamma_{i}^{(l+1)}$, for $i = 1, \ldots, N_c$

such that $\Upsilon_i \ni \gamma_{i}^{(l+1)} \perp - (N_{i} \gamma_{i}^{(l+1)} + r) \in \Upsilon_i^{o}$  

(2.27)

where $\Upsilon_i = \{ [x,y,z]^T \in \mathbb{R}^3 | \sqrt{y^2 + z^2} \leq \mu_i x \}$

and $\Upsilon_i^{o} = \{ [x,y,z]^T \in \mathbb{R}^3 | x \leq -\mu_i \sqrt{y^2 + z^2} \}$,

with suitably defined matrix $N$ and vector $r$.

For the sake of clarity, the superscripts which indicate the time step will be dropped in the following derivation. First, write the KKT conditions for the optimization problem in Equation 2.26, where $\lambda_i$ is a slack variable.

\[
D^T_{i,u}v + \lambda_i \frac{\gamma_{i,u}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} = 0 \tag{2.28}
\]

\[
D^T_{i,w}v + \lambda_i \frac{\gamma_{i,w}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} = 0 \tag{2.29}
\]

\[
\mu_i \gamma_{i,n} - \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \geq 0 \tag{2.30}
\]

\[
\lambda_i \geq 0 \tag{2.31}
\]

\[
\lambda_i \left( \mu_i \gamma_{i,n} - \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \right) = 0 \tag{2.32}
\]

Combine Equations 2.28 and 2.29 to obtain the following expression for $\lambda_i$:

\[
\lambda_i = \sqrt{(D^T_{i,u}v)^2 + (D^T_{i,w}v)^2} \tag{2.33}
\]

From Equation 2.25, the following expression can be obtained for the case where $\gamma_{i,n} > 0$:

\[
\frac{1}{h} \Phi_i + D^T_{i,n}v = \mu_i \sqrt{(D^T_{i,u}v)^2 + (D^T_{i,w}v)^2} = \mu_i \lambda_i \tag{2.34}
\]

Now, define $g_i = \left[ \frac{1}{h} \Phi_i + D^T_{i,n}v, D^T_{i,u}v, D^T_{i,w}v \right]^T$ and consider the following:

\[
g_i^T \gamma_i = \gamma_{i,n} \left( \frac{1}{h} \Phi_i + D^T_{i,n}v \right) + \gamma_{i,u} D^T_{i,u}v + \gamma_{i,w} D^T_{i,w}v \tag{2.35}
\]
Substitute Equations 2.34, 2.28, and 2.29 into Equation 2.35:

\[ g_i^T \gamma_i = \gamma_{i,n} \lambda_i - \gamma_{i,u} \lambda_i \frac{\gamma_{i,u}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} - \gamma_{i,w} \frac{\gamma_{i,u}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \]  
\[ (2.36) \]

\[ = \lambda_i \left( \gamma_{i,n} \mu_i - \sqrt{\gamma_{i,w}^2 + \gamma_{i,w}^2} \right) = 0 \]  
\[ (2.37) \]

where the last equality in Equation 2.37 is based on Equation 2.32.

Now, it is easy to verify via direct computation and application of Equation 2.30 that \( \gamma_i \in \Upsilon_i \), and by Equation 2.25 that \(-g_i \in \Upsilon_i^c\). Additionally, Equation 2.37 shows that \(-g_i \perp \gamma_i\), so the complementarity \( \Upsilon_i \ni \gamma \perp -g_i \in \Upsilon_i^c \) holds.

Finally, define \( b_i = \left[ \frac{1}{h} \Phi_i, 0, 0 \right]^T \) and \( k = Mv^{(l)} + hf\left(t^{(l)}, q^{(l)}, v^{(l)}\right) \) so that Equation 2.24 can be expressed as

\[ Mv^{(l+1)} = k + D\gamma^{(l+1)}, \]  
\[ (2.38) \]

which leads to the following expression for \( v^{(l+1)} \) in terms of \( \gamma^{(l+1)} \):

\[ v^{(l+1)} = M^{-1} \left( k + D\gamma^{(l+1)} \right). \]  
\[ (2.39) \]

Finally, define the matrix \( N \) and vector \( r \) as follows:

\[ N = D^T M^{-1} D, \]  
\[ (2.40) \]

\[ r = b + D^T M^{-1} k. \]  
\[ (2.41) \]

With these definitions, it is easy to show that

\[ g_i = D_i^T v^{(l+1)} + b_i \]  
\[ (2.42) \]

\[ = D_i^T \left( M^{-1} k + M^{-1} D\gamma^{(l+1)} \right) + b_i \]  
\[ (2.43) \]

\[ = D_i^T M^{-1} D\gamma^{(l+1)} + D_i^T M^{-1} k + b_i \]  
\[ (2.44) \]

\[ = (N\gamma^{(l+1)} + r)_i. \]  
\[ (2.45) \]
Therefore, it has been shown that solving the relaxed form of the equations of motion (Equations 2.23-2.26) is equivalent to solving the CCP in Equation 2.27.

**Equivalence of CCP and CCQO**

Here, it will be verified that solving the CCP of Equation 2.27 is equivalent to solving a cone-constrained quadratic optimization (CCQO) problem. For the case with only unilateral constraints (contacts), this optimization problem takes the form

\[
\begin{align*}
\min \, q(\gamma) &= \frac{1}{2} \gamma^T N \gamma + r^T \gamma \\
\text{subject to } \gamma_i &\in \Upsilon_i \text{ for } i = 1, 2, \ldots, N_c,
\end{align*}
\]

where \( \gamma_i \) is the triplet of multipliers associated with contact \( i \) and \( \Upsilon_i \) is the friction cone of contact \( i \). Note also that \( \gamma = [\gamma_1^T, \gamma_2^T, \ldots, \gamma_{N_c}^T]^T \).

This can be verified by considering the KKT first-order necessary conditions. First, define the Lagrangian, \( L(\gamma, \lambda) \), for the constrained optimization problem in Eq. 2.46,

\[
L(\gamma, \lambda) = \frac{1}{2} \gamma^T N \gamma + r^T \gamma - \sum_{i=1}^{N_c} \lambda_i \left( \mu_i \gamma_{i,n} - \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \right).
\]

Then, the KKT conditions for Eq. 2.46 can be stated as follows. If \( \gamma^* \) is a local solution of Eq. 2.46, then there exists a vector \( \lambda^* \) of Lagrange multipliers, \( \lambda_i^*, i = 1, \ldots, N_c \), such that the following conditions are satisfied,

\[
\begin{align*}
\nabla L(\gamma^*, \lambda^*) &= 0, \\
c_i(\gamma^*) &\geq 0, \quad \forall i = 1, \ldots, N_c \\
\lambda_i^* &\geq 0, \quad \forall i = 1, \ldots, N_c \\
\lambda_i^* c_i(\gamma^*) &= 0, \quad \forall i = 1, \ldots, N_c
\end{align*}
\]

where \( c_i(\gamma) = \mu_i \gamma_{i,n} - \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \).
Define the vector $s$ as follows:

$$s = \left[ \ldots, \lambda_i \mu_i, \frac{-\lambda_i \gamma_{i,u}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}}, \frac{-\lambda_i \gamma_{i,w}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}}, \ldots \right]^T \in \mathbb{R}^{3N_c} \quad (2.52)$$

Then, the condition 2.48 can be expressed as follows:

$$N\gamma + r = s \quad (2.53)$$

$$D^T M^{-1} D\gamma + D^T M^{-1} k + b = s \quad (2.54)$$

$$D^T M^{-1} (D\gamma) + b = s \quad (2.55)$$

$$D^T v + b = s \quad (2.56)$$

Specifically, the rows of Eq. 2.53 associated with contact $i$ are

$$(N\gamma + r)_i = D^T_i v + b_i = s_i \quad (2.57)$$

$$\begin{bmatrix} D_{i,n}^T v + \frac{1}{h} \Phi_i \\ D_{i,u}^T v \\ D_{i,w}^T v \end{bmatrix} = \begin{bmatrix} \lambda_i \mu_i \\ \frac{-\lambda_i \gamma_{i,u}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \\ \frac{-\lambda_i \gamma_{i,w}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \end{bmatrix} \quad (2.58)$$

By manipulating the second and third lines of Eq. 2.58, $\lambda_i$ can be expressed as follows:

$$\lambda_i = \sqrt{(D_{i,u}^T v)^2 + (D_{i,w}^T v)^2} \quad (2.59)$$

Now, use Eq. 2.51 to write
\[ \lambda_i \left( \mu_i \gamma_{i,n} - \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \right) = 0 \]  
(2.60)

\[ \lambda_i \mu_i \gamma_{i,n} - \lambda_i \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} = 0 \]  
(2.61)

\[ \lambda_i \mu_i \gamma_{i,n} - \frac{\lambda_i}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \left( \gamma_{i,u}^2 + \gamma_{i,w}^2 \right) = 0 \]  
(2.62)

\[ \lambda_i \mu_i \gamma_{i,n} + \frac{-\lambda_i \gamma_{i,u}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \gamma_{i,u} + \frac{-\lambda_i \gamma_{i,w}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \gamma_{i,w} = 0. \]  
(2.63)

Then, the expressions in Equation 2.58 can be used in Equation 2.63 to obtain:

\[ \gamma_{i,n} \left( D_{i,n}^T v + \frac{1}{h} \Phi_i \right) + \gamma_{i,u} D_{i,u}^T v + \gamma_{i,w} D_{i,w}^T v = 0 \]  
(2.64)

\[ \gamma_i^T (D_i^T v + b_i) = 0 \]  
(2.65)

By Equation 2.49, it is clear that \( \gamma_i \in \Upsilon_i \), and we have just shown in Equation 2.65 that \( \gamma_i \perp D_i^T v + b_i \). It remains to be shown that \( - (D_i^T v + b_i) \in \Upsilon_i^\circ \).

Now, consider Eq. 2.49. After multiplying by \( \lambda_i \) and using some of the relations in Eq. 2.58, the following expression is obtained.

\[ \gamma_{i,n} \left( D_{i,n}^T v + \frac{1}{h} \Phi_i \right) - \sqrt{(D_{i,u}^T v)^2 + (D_{i,w}^T v)^2} \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} \geq 0 \]  
(2.66)

Consider two cases corresponding to Eq. 2.49, dealing with equality and strict inequality separately. First, consider the case when \( \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} = \mu_i \gamma_{i,n} \). Then Equation 2.66 becomes

\[ \gamma_{i,n} \left( D_{i,n}^T v + \frac{1}{h} \Phi_i \right) \geq \mu_i \gamma_{i,n} \sqrt{(D_{i,u}^T v)^2 + (D_{i,w}^T v)^2} \]  
(2.67)

\[ - \left( D_{i,n}^T v + \frac{1}{h} \Phi_i \right) \leq - \mu_i \sqrt{(-D_{i,u}^T v)^2 + (-D_{i,w}^T v)^2}, \]  
(2.68)

which simply means that \( - (D_i^T v + b_i) \in \Upsilon_i^\circ \).

Second, consider the case when \( \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2} < \mu_i \gamma_{i,n} \). In this case, Equation 2.51 implies that \( \lambda_i = 0 \). Then, Equation 2.58 shows that \( D_i^T v + b_i = [0, 0, 0]^T \), and it is immediately obvious
that $-\left(D^Tv + b_i\right) \in \Upsilon_i^\circ$ in this case as well. Therefore, it has been shown that the following complementarity problem represents the first-order optimality conditions of the optimization problem in Equation 2.46.

\[ \Upsilon_i \ni \gamma_i \perp -\left(D^Tv + b_i\right) \in \Upsilon_i^\circ \quad (2.69) \]

Note that the CCP in Equation 2.69 is identical to the CCP in Equation 2.27.

**N is Symmetric, Positive Semidefinite**

The matrix $N$ has been defined as $N = D^T M^{-1} D$. Here, $M$ is diagonal, constant, and positive definite because we assume rigid bodies and centroidal reference frames. Therefore, we can write

\[
N = D^T M^{-1} D = D^T M^{-\frac{1}{2}} M^{-\frac{1}{2}} D = D^T \left(M^{-\frac{1}{2}}\right)^T M^{-\frac{1}{2}} D = \left(M^{-\frac{1}{2}} D\right)^T \left(M^{-\frac{1}{2}} D\right). \quad (2.73)
\]

It is known that $B^T B$ is positive semidefinite for any $B$, so the matrix $N$ is positive semidefinite.

Because $N$ is symmetric positive semidefinite, $x^T N x$ is convex. Additionally, $x^T r$ is convex, so the sum $x^T N x + x^T r$ is also convex. Therefore, the objective function of the optimization problem in Equation 2.46 is convex.

**Projection onto Cones**

Here, the projection operator $\Pi_K$, which projects a vector onto the Cartesian product of the cones $\Upsilon_i$, is described. First, some definitions and properties of the associated cones are provided. Recall the definition of $\Upsilon_i$, $i = 1, \ldots, N_c$, where $\Upsilon_i = \{[x, y, z]^T \in \mathbb{R}^3 | \sqrt{y^2 + z^2} \leq \mu_i x\}$.

A set $C$ is a convex cone if
\[ \alpha a + \beta b \in C \quad \forall \alpha, \beta > 0, \forall a, b \in C. \] (2.74)

The cone \( \Upsilon_i \) can be re-expressed as \( \Upsilon_i = \{ c = [x, w^T]^T \in \mathbb{R}^3 \mid ||w||_2 \leq \mu_i x \} \). Then, define the following:

\[
\begin{align*}
\mathbf{a} &= [d, e^T]^T \in \Upsilon_i \implies ||e||_2 \leq \mu_i d \quad (2.75) \\
\mathbf{b} &= [f, g^T]^T \in \Upsilon_i \implies ||g||_2 \leq \mu_i f. \quad (2.76)
\end{align*}
\]

Consider the following, where \( \alpha, \beta > 0 \):

\[
||\alpha e + \beta g||_2 \leq ||\alpha e||_2 + ||\beta g||_2 \quad (2.77)
\]

\[
= \alpha ||e||_2 + \beta ||g||_2 \quad (2.78)
\]

\[
\leq \alpha \mu_i d + \beta \mu_i f \quad (2.79)
\]

\[
= \mu_i (\alpha d + \beta f). \quad (2.80)
\]

This shows that \( \alpha a + \beta b = [\alpha d + \beta f, \alpha e^T + \beta g^T]^T \in \Upsilon_i \) and therefore, \( \Upsilon_i \) is a convex cone.

Next, define \( \Upsilon \) as the Cartesian product of the cones \( \Upsilon_i \):

\[
\Upsilon = \Upsilon_1 \times \Upsilon_2 \times \cdots \times \Upsilon_{N_c} = \left\{ \left[ \gamma_1^T, \ldots, \gamma_{N_c}^T \right]^T \in \mathbb{R}^{3N_c} \mid \gamma_1 \in \Upsilon_1, \ldots, \gamma_{N_c} \in \Upsilon_{N_c} \right\}. \quad (2.81)
\]

It is immediate that \( \Upsilon \) is also a convex set. Additionally, note that \( \Upsilon \) is exactly the feasible region of 2.46.

Now, consider the projection of a vector \( \gamma_i \in \mathbb{R}^3 \) onto \( \Upsilon_i \).

\[
\hat{\gamma}_i = \Pi_{\Upsilon_i} (\gamma_i) = \arg \min_{\hat{\gamma}_i \in \Upsilon_i} ||\gamma_i - \hat{\gamma}_i||_2^2. \quad (2.82)
\]

The result of the projection is the point \( \hat{\gamma}_i \) in \( \Upsilon_i \) closest to \( \gamma_i \). If the point \( \gamma_i \) is inside the cone \( \Upsilon_i \) it remains unchanged. If the point \( \gamma_i \) is inside the polar cone \( \Upsilon_i^\circ \) it is projected to the origin. If the point is outside both the cone and the polar cone, it is projected to the surface of the cone.
Figure 2.2: Projection of $\gamma_i$ onto cone $\Upsilon_i$, resulting in $\hat{\gamma}_i$
Figure 2.3: Projection for single friction cone: (top) definition of projection, (middle) geometry of projection, (bottom) examples of projection.
An analytical expression can be obtained geometrically for the case of projection onto the surface of the cone. First, let $\gamma_{i,t} = \sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}$ be the tangential component of $\gamma_i$. Then, consider the triangles $BEO$ and $OBC$ in Figure 2.3. Using the definition of $\tilde{\gamma}_i$, the expressions $(BE) = \tilde{\gamma}_{i,n}$, $(OE) = \tilde{\gamma}_{i,t} = \mu_i \tilde{\gamma}_{i,n}$, and $(OB) = \sqrt{\tilde{\gamma}_{i,n}^2 + \tilde{\gamma}_{i,t}^2} = \tilde{\gamma}_{i,n} \sqrt{\mu_i^2 + 1}$ can be written. Then, use the similarity of triangles $BEO$ and $OBC$ to obtain the following:

\[
(OC) = \frac{(OB)^2}{(BE)} = \tilde{\gamma}_{i,n} \left( \mu_i^2 + 1 \right) \tag{2.83}
\]

\[
(CB) = \frac{(OC)(OE)}{(OB)} = \mu_i \tilde{\gamma}_{i,n} \sqrt{\mu_i^2 + 1}. \tag{2.84}
\]

Next, consider triangle $AFC$, where $(AF) = \gamma_{i,t}$ and $(CF) = \tilde{\gamma}_{i,n} \left( \mu_i^2 + 1 \right) - \gamma_{i,n}$. Finally, similarity between triangles $OBC$ and $AFC$ leads to the following:

\[
\frac{(CB)}{(OB)} = \frac{(CF)}{(AF)} \Rightarrow \frac{\mu_i \tilde{\gamma}_{i,n} \sqrt{\mu_i^2 + 1}}{\tilde{\gamma}_{i,n} \sqrt{\mu_i^2 + 1}} = \frac{\tilde{\gamma}_{i,n} \left( \mu_i^2 + 1 \right) - \gamma_{i,n}}{\gamma_{i,t}} \tag{2.85}
\]

\[
\Rightarrow \mu_i \tilde{\gamma}_{i,t} = \tilde{\gamma}_{i,n} \left( \mu_i^2 + 1 \right) - \gamma_{i,n} \tag{2.86}
\]

\[
\Rightarrow \tilde{\gamma}_{i,n} = \frac{\gamma_{i,n} + \mu_i \gamma_{i,t}}{\mu_i^2 + 1} \tag{2.87}
\]

Note that the following expressions are necessary to transform back to $u, w$ coordinates.

\[
\tilde{\gamma}_{i,t} = \mu_i \tilde{\gamma}_{i,n} \tag{2.89}
\]

\[
\tilde{\gamma}_{i,u} = \gamma_{i,u} \frac{\tilde{\gamma}_{i,t}}{\gamma_{i,t}} = \gamma_{i,u} \frac{\mu_i \tilde{\gamma}_{i,n}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \tag{2.90}
\]

\[
\tilde{\gamma}_{i,w} = \gamma_{i,w} \frac{\tilde{\gamma}_{i,t}}{\gamma_{i,t}} = \gamma_{i,w} \frac{\mu_i \tilde{\gamma}_{i,n}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \tag{2.91}
\]

Therefore, a closed form expression can be written for the projection as follows:
\[\Pi_{\Upsilon_i} (\gamma_i) = \begin{cases} 
\gamma_i & \text{if } \gamma_i \in \Upsilon_i \\
0 & \text{if } \gamma_i \in \Upsilon_i^o \\
\left[ \frac{\gamma_{i,n} + \mu_i \gamma_{i,t}}{\mu_i^2 + 1}, \frac{\mu_i \gamma_{i,n}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}}, \frac{\mu_i \gamma_{i,n}}{\sqrt{\gamma_{i,u}^2 + \gamma_{i,w}^2}} \right]^T & \text{if } \gamma_i \notin (\Upsilon_i \cup \Upsilon_i^o) \end{cases} \] (2.92)

Finally, the projection of a vector \( \gamma \in \mathbb{R}^{3N_c} \) onto the set \( \Upsilon \) is defined as \( \hat{\gamma} \in \mathbb{R}^{3N_c} \), where

\[\hat{\gamma} = \Pi_{\Upsilon} (\gamma) = \left[ \Pi_{\Upsilon_1} (\gamma_1)^T, \ldots, \Pi_{\Upsilon_{N_c}} (\gamma_{N_c})^T \right]^T \] (2.93)

### 2.1.6 On the Lack of Uniqueness in the DVI Solution

It is important to note that the DVI formulation suffers from a lack of uniqueness for the numerical solution of both the force and velocity distributions when dealing with problems with friction [22, 23]. This is true for any method based on a DVI formulation [24, 25, 26, 27, 28]. In the case of rigid body dynamics, this can be explained by the nature of the rigid body assumption. For example, consider a rigid symmetric four-legged table resting on a perfectly flat rigid plane. The vertical reaction forces at the four legs are non-unique. For example, if the weight of the table is 100 N, it is possible that the vertical reaction force at each leg is 25 N. However, it is equally possible that two diagonally opposite legs have reaction forces of 30 N each, while the other pair of diagonally opposite legs have reaction forces of 20 N each. In fact, there are infinitely many sets of reaction forces which satisfy the DVI equations corresponding to the case of the rigid table resting on a rigid plane.

In the frictionless case, when \( \mu_i = 0 \) for all contacts, the force distribution is still non-unique but the velocity distribution is unique. Once again, consider the previous example of the rigid table resting on a plane. The two sets of vertical reaction forces were different, yet they led to the same velocity solution.
2.2 DEM Formulation

Cundall proposed the discrete element method in 1971 to model the mechanical behavior of granular material [29, 1]. The DEM can be classified as a penalty method, where the force acting between two colliding bodies is computed based on the associated interpenetration. In fact, the reaction force is often modeled by a spring-damper element where the spring and damping coefficients can be determined from continuum mechanics theory or from experimental results [30]. After computing contact forces, Newton’s Second Law can be used to compute the acceleration of each body, and numerical integration can be used to compute new velocities and positions. Numerous contact force models have been developed and used for both normal and tangential contact forces (for example, see [31]). One such model is described in the remainder of this section.

2.2.1 Normal Force Model

As described in [32], normal force models can be categorized into four groups: continuous potential models, linear viscoelastic models, non-linear viscoelastic models, and hysteretic models. Different normal force models are tailored to capture the behavior of different types of contacts, so the choice of a force model is problem dependent. A linear viscoelastic model will be described next as a simple representative model.

Figure 2.4 shows two bodies \( i \) and \( j \) at positions \( \mathbf{r}_i \) and \( \mathbf{r}_j \). For simplicity, the spheres are assumed to have the same diameter, \( d \), and mass, \( m \). The relative position, \( \mathbf{r}_{ij} \), is defined as
The normal force is modeled as a spring-damper acting at the interface between the two colliding bodies and is computed as

\[ F_{n_{ij}} = f \left( \frac{\delta_{ij}}{d} \right) \left( k_n \delta_{ij} n_{ij} - \gamma_n \frac{m}{2} v_{n_{ij}} \right). \]  

(2.95)

Here, \( \delta_{ij} = d - r_{ij} \) is the normal compression, \( r_{ij} = \|r_{ij}\|_2 \), \( n_{ij} \) is the unit normal, \( v_{n_{ij}} \) is the relative normal velocity, \( k_n \) is the spring stiffness and \( \gamma_n \) is the damping coefficient. Additionally, the spring can be linear \((f(x) = 1)\) or Hertzian \((f(x) = \sqrt{x})\).

### 2.2.2 Tangential Force Model

As in the case of the normal force, many models have been proposed for the tangential force [33, 34]. A linear viscoelastic type tangential force model will be briefly described next. The tangential force depends on the tangential spring stiffness \( k_t \), the tangential damping coefficient \( \gamma_t \), the relative tangential velocity \( v_{t_{ij}} \), and the relative tangential displacement \( u_{t_{ij}} \). The relative tangential velocity and displacement take the form

\[ v_{t_{ij}} = v_{ij} - v_{n_{ij}} - \frac{1}{2} (\omega_i - \omega_j) \times r_{ij} \]  

(2.96)

\[ u_{t_{ij}} = \int_{t_{init}}^{t_{final}} v_t(\tau) \, d\tau. \]  

(2.97)

Note that computing the tangential displacement is difficult because the sliding distance at the contact point must be tracked throughout the duration of the contact. Finally, the tangential force can be computed in the same manner as the normal force,

\[ F_{t_{ij}} = f \left( \frac{\delta_{ij}}{d} \right) \left( -k_t u_t - \gamma_t \frac{m}{2} v_{t_{ij}} \right). \]  

(2.98)
The Coulomb-criterion, which states that the magnitude of the tangential force must be less than or equal to the friction coefficient, \( \mu \), times the magnitude of the normal force, must also be satisfied. For example, this can be enforced by truncating \( u_{t_{ij}} \) as necessary.

In practice, the tangential displacement is approximated and not truly tracked over the duration of the contact, using, for example, the estimate \( u_{t_{ij}} = v_t h \) for the translational displacement, where \( h \) is the time step of the simulation. Then, the tangential force is computed as the maximum of \( \mu F_{n_{ij}} \) and the value from Equation 2.98.

### 2.2.3 Overall Method for DEM

Once a force model has been selected or developed to capture the interaction between colliding bodies, the remaining steps of the discrete element method are straightforward. The computational flow takes the following form at each time step. First, collision detection is performed to determine which bodies are colliding and the data associated with each contact (normal penetration, contact normal, etc.). Next, the normal and tangential contact forces for each contact are computed using the appropriate models. The acceleration of each particle is then computed from the sum of forces acting on the body and the mass using Newton’s Second Law. Finally, explicit numerical methods such as Explicit Euler or Velocity Verlet are used to compute the velocity and then the position update for each body [35].

The DEM is attractive in its simplicity and robustness. The implementation is straightforward and can be parallelized easily. For example, after collision detection the contact forces can be computed in parallel with one thread per contact. The wide variety of contact force models gives a high level of control over the results. However, there are some drawbacks to DEM. First, the method generally requires very small time steps, on the order of \( 10^{-6} \) seconds. The time step must be small enough such that a body does not move more than a fraction of its diameter during one step so that the process of contact, compression, and rebound can be captured. The stiff springs common in contact models can also induce numerical instability or artificial bounciness in the dynamics. Note that a GPU-based parallel implementation of the DEM has been described in [35]. There, the largest problem that was solved had over three million spherical bodies.
Implementation details for this method will be given in Chapter 6.
Chapter 3

Parallel Computing

The purpose of this chapter is to motivate the use of parallel computing to accelerate many-body dynamics simulations. This chapter will describe the reasons that parallel computing is becoming increasingly necessary for the next generation of computational dynamics simulation tools. Then, three specific models of parallel computing will be described and the advantages of each will be considered. Note that this chapter only provides background on parallel computing, while Chapter 6 describes the use of parallel computing in the many-body dynamics implementation.

3.1 Motivation and Background

Parallel computing is increasingly the only reliable way to achieve increased performance in applicable computing tasks. In the past, programmers could rely on increasing clock frequencies to achieve better performance in serial programming applications. However, this trend is losing speed and cannot be relied upon for future gains. In fact, Moore’s Law [36] is still in effect, in which the number of transistors per unit area doubles roughly every 18 months. However, in recent years this trend can only manifest itself in the development of multi-core technology. The shift from sequential to multi-core technology has been driven by three primary obstacles to continued increases in performance of the sequential model [37].

First, the memory wall represents the growing disparity between memory access speed and CPU processing speed. While CPU speeds have increased regularly over the years, memory access speeds have increased at a significantly lower rate. This disparity exists both in terms of memory latency and memory bandwidth. Memory latency represents the amount of time it takes to initiate
a memory transfer; i.e., the time it takes to transmit one bit. Memory bandwidth represents the amount of data which can be transferred per unit time. Note that memory bandwidth has proven easier to increase by adding more input/output pins. Latency is significantly more difficult to improve, as it is dependent on conduction of the data and may rely on material science, for example, for improved performance.

Second, the instruction level parallelism (ILP) wall represents the exhaustion of speed improvements achieved through speculative execution and branch prediction. ILP is a class of techniques which rely on compilers or chip hardware to identify code segments which can be executed in parallel. Various ILP techniques include out-of-order execution, speculative execution, and branch prediction. In speculative execution, for example, a future segment of code may be executed in parallel with current execution, before it has been determined whether or not the result will really be needed. If it is needed, the result is ready when the decision is made. If it is not needed, the result is discarded. ILP has provided improvements in execution speed in the past, but continued gains are more and more difficult to achieve. Further significant gains from ILP are unlikely as they would require even longer horizons in terms of branch prediction, and even more power consumption from unused computation results.

Finally, the power wall represents the ceiling on useful clock frequencies due to the associated power consumption. In fact, power consumption is limited by heat dissipation. Therefore, further increases in clock frequencies (and associated increases in power consumption) are infeasible without dramatically new cooling strategies or available materials.

These three walls have steered the multiprocessor industry away from single core CPUs to multi-core and many-core architectures. In this paradigm, chip designers are using multiple compute cores on the same chip, each of which runs at a lower clock frequency. With this trend, computational speed, measured in floating point operations per second (FLOPS), has still been increasing at the same rate as in the sequential single-core era. However, accessing this computational speed requires more than simply waiting for faster CPUs to be released. Parallel computing requires special programming practices to leverage the compute power of chips with multiple compute cores.
Three main models for parallel programming have risen to prominence. The first model is based on the distributed-memory multiprocessor architecture. In this model, a set of nodes are connected by a network. Each node consists of a processor and some discrete memory. Data can be transferred from one node to another; but only the memory in the same node is directly accessible to the processor. This model can be leveraged by the Message Passing Interface (MPI), for example, which will be described in Section 3.3.

The second model is based on the shared-memory multiprocessor architecture. In this model, a set of compute cores uses the same address space. In other words, each processor has direct access to all of the same memory locations. This model is most commonly used by multi-core CPUs. In this case, multiple compute cores on the same chip have access to the same system memory. This model can be leveraged by OpenMP, for example, which will be described in Section 3.4.

The third model is based on the concept of a compute accelerator. In this model, a device separate from the CPU is used to perform parallel computations. This is most commonly used by Graphics Processing Unit (GPU) architecture. The GPU is a specialized device with its own memory and many lightweight parallel compute cores. GPU programming will be further described in a following section. Another compute accelerator which has been recently released is Intel’s Xeon Phi coprocessor [38]. The Xeon Phi architecture is similar to a GPU in that it is a separate card with its own processors and memory. However, the Xeon Phi has a smaller number of more capable compute cores (60 x86 cores) than a GPU.

### 3.2 GPU Computing

The Graphics Processing Unit (GPU), traditionally used primarily for high end graphics in video games, is increasingly being used for general purpose computing. NVIDIA has facilitated this trend through their introduction of the Compute Unified Device Architecture (CUDA) [39], an application programming interface (API) that provides many of the programming constructs (via software) necessary to leverage the computing power of the GPU (the hardware). The GPU is a powerful tool for large scale simulation due to its parallel structure. This section gives some background on the hardware and software which are necessary for GPU computing.
3.2.1 GPU Hardware Structure

The GPU was originally designed to aid computers in the display of complex graphics, requiring high frame rates and the ability to process and display hundreds of thousands of polygons in each frame. Therefore, the physical structure of the GPU is such that many simple calculations can be performed simultaneously. The strengths of the GPU can be most easily observed by comparison with the central processing unit (CPU), also called the host. High end CPUs may have, for example, sixteen processing cores, while a high-end GPU from NVIDIA may have nearly 3000 scalar processors [40]. While the GPU has more physical cores, it is important to recognize that GPU cores are less sophisticated and run at a lower clock speed than CPU cores.

To describe the physical structure of the GPU, the NVIDIA TESLA C1060 GPU will be used as an example. This GPU has one Stream Processor Array (SPA) which is composed of ten Texture Processor Clusters (TPC). Each TPC has three Stream Multiprocessors (SM), each of which has eight Scalar Processors (SP). In total, the TESLA card has 240 scalar processors [41].

In addition to the processing cores, the second important aspect of GPU hardware is that of the memory hierarchy. The memory on the GPU is divided into several types, each with different access patterns, latencies, and bandwidths.

- Registers (read/write per thread): Used to hold temporary data while performing computations. Each computational thread has its own registers. Latency: 1 cycle, Bandwidth: 1 TB/s

- Shared memory (read/write per block): Used to hold data which needs to be shared by the threads within a block. Latency: 1 cycle, Bandwidth: 1 TB/s

- Global memory (read/write per grid): Used to hold input and output data, accessible by all threads. Latency: 400-800 cycles (uncached), Bandwidth: 150 GB/s

- Constant memory (read only per grid): Used to hold constants, accessible by all threads. Latency: 1-800 cycles (depending if cached), Bandwidth: 150 GB/s
• Texture memory (read only per grid): Used to hold data arranged in a 2D grid, accessible by all threads. Latency: 400-800 cycles (depending if cached), Bandwidth: 150 GB/s

3.2.2 GPU Software Structure

A straightforward way to program the GPU is through the use of NVIDIA’s CUDA software development kit (SDK). The CUDA SDK provides a collection of tools which can be used to execute C code on the GPU device. CUDA code is organized into functions, or kernels, which run on the device. A kernel is launched by specifying the execution configuration, calling the kernel, and passing in pointers to the necessary data.

As an example, consider the following listing, where a kernel called myKernel is launched. The execution configuration is specified inside of the <<< >>> construct. The kernel is launched by a total of 25,600 threads. These threads are organized into $10 \times 10 = 100$ blocks of $16 \times 16 = 256$ threads each. Note that threads in the same block can synchronize their execution and share data through shared memory. Threads in different blocks cannot use the same shared memory [39].

Listing 3.1: Example of launching a CUDA kernel

```c
__global__ void myKernel(); // declaration of kernel
dim3 dimBlock(16,16); // each block should be 16x16=256 threads
dim3 dimGrid(10,10); // use a 10x10 grid of blocks, for a total of 25600 threads
myKernel<<<dimGrid,dimBlock>>>(Ad,Bd,Cd);
```

The execution configuration is related to the actual execution of the kernel by each thread on the device. The device manages the execution via two levels of scheduling. First, thread blocks are assigned to each SM of the device. Second, each SM schedules the execution of the threads in the block until all the threads of the block have finished the kernel. Then, the SM will start execution of the next waiting block. An important concept in the execution of a block by an SM is that of a warp of threads. A warp is a group of 32 threads which execute the kernel together. The SM hardware is able to switch between warps with almost zero overhead. This behavior is called context switching, and is much more costly on the CPU than on the GPU. Therefore, the
GPU is able to effectively hide memory transactions by execution since the SM has more threads assigned to it than it can actually process simultaneously. For example, one warp of threads in a block accesses global memory and has to wait approximately 400 cycles for the data to arrive. This warp can be set aside, and another warp from the block can use the hardware resources to perform computations for which all the operands are available. Therefore, it is important that a kernel has a high ratio of computation to memory accesses so that memory access latency can be hidden by the warp scheduling.

For optimal performance, several important aspects must be considered when writing code to be executed on the GPU. First, the GPU can be considered a single instruction multiple thread (SIMT) or single instruction multiple data (SIMD) device. Therefore, GPU computing is only applicable if the problem can be posed as a sequence of operations, each of which is performed on a set of different data. Note that conditionals can be used to cause different threads to perform different computations, which will adversely affect performance. Next, there is a bandwidth limit on moving data to and from the device of 8 GB/s due to the PCI Express 2.0 x16 connection between GPU and CPU. Therefore, the ideal usage pattern is to move data to the device and perform a large amount of computation before bringing the results back. Finally, memory access patterns on the device can also affect performance. If each thread in a kernel requires a significant amount of data, shared memory and registers may not be sufficient. In that case, the much slower global memory must be used, significantly slowing down the computation. Additionally, memory access speeds can suffer if consecutive threads do not access data in adjacent locations. Taking these considerations into account will help achieve best performance from a parallel algorithm.

GPU programming using CUDA can be challenging because the programmer must manage the device memory and execution configuration manually. However, in recent years software libraries have been developed which provide commonly used operations at a higher level. The performance of these implementations may not be quite as high as if the programmer had implemented a fully optimized version tailored to the problem of interest, but the code is easier to develop and is significantly more readable. One such library is Thrust [42], which is a template-based library similar to
the C++ Standard Template Library (STL) [43]. One feature that Thrust provides are vector containers to store data on the GPU. For example, the analog of STL’s `std::vector<int> myVec` would be Thrust’s `thrust::device_vector<int> myVec`.

The Thrust library also provides a set of data-parallel functions which can be executed on the GPU, including scan, sort, and reduce operations. The functions are provided at a high level, meaning that they operate directly on the `thrust device_vector<>` objects. Thrust manages the process of choosing the best execution configuration depending on the available hardware. The current implementation of the many-body dynamics simulation code used in this work relies heavily on Thrust (see Chapter 6).

In general, many-body dynamics maps well to the GPU computing paradigm. When the number of bodies in the system is large, the implementation can easily take advantage of the data-parallel nature of the GPU. For example, body forces can be computed in parallel with one thread per body and no dependencies between threads. Collision detection can also be accelerated on the GPU [44]. Furthermore, the GPU can be used to solve the equations of motion for constrained multibody dynamics through the use of linear algebra libraries like CUSP [45].

### 3.3 MPI Parallelism

The Message Passing Interface (MPI) is a standard which defines a collection of routines in a library which can be used to write message passing parallel programs. MPI itself is a standard, meaning an agreed-upon syntax and protocol for communicating among parallel computers [46]. Several implementations of the MPI standard exist, including MPICH and OpenMPI. MPI is relevant in distributed-memory hierarchies, where a collection of nodes are connected, each of which has an independent processor and memory.

In MPI computing, the same program is launched independently as a collection of processes. Each process is given a rank, or an identifier, which can be used to differentiate one process from another. Each process executes the program independently, but the program can be written to do different things depending on the rank of the process. This is an example of the Single Program, Multiple Data (SPMD) paradigm, where each instance of the program has the same variables,
stored in different memory locations with possibly different values. The ranks execute independently, subject to interprocess communication or synchronization.

Various operations can be leveraged to coordinate multiple MPI ranks. These generally act to pass or exchange data among ranks, or synchronize the ranks of the program. The most important concept in MPI programming is that of passing messages between ranks. The defining attributes of a message are the rank, memory location, data type, and data size for each rank participating in passing of the message.

The simplest type of message passing is called point-to-point communication, when one rank sends a message to a different rank. The sending rank makes a call to MPI_Send, while the receiving rank makes a call to MPI_Recv. Note that a send command in MPI can be either blocking or non-blocking. A blocking send returns only after the message has actually been sent. In other words, the data in the source buffer can be modified as soon as the send call returns because it has already been sent. The drawbacks of blocking sends are that data transfer and computation cannot be overlapped, and that the program may deadlock if the send/receive pairs are not ordered correctly. A non-blocking send returns immediately, whether or not the message has been sent. Therefore, care must be taken to avoid modifying the source data before the send actually occurs. The benefit of non-blocking send/receive calls (MPI_Isend/MPI_Irecv) is that computation can be done while waiting for the message to be sent or received. Auxiliary functions such as MPI_Probe and MPI_Wait can be used to check for the existence of an incoming message and to wait for a send/receive to complete, respectively.

The second type of message passing is called collective communication. This type of communication involves a group of ranks, all of which must make the same collective call. For example, MPI_Barrier is a blocking call which forces synchronization among all ranks. Note that this explicit synchronization is rarely needed, as the message passing itself usually embeds the required synchronization. Other useful collective operations include MPI_Bcast, in which one data element is broadcast to all ranks, MPI_Scatter, in which a local serial array is distributed among all ranks, MPI_Gather, in which a distributed array is collected in a serial array of a single rank, and MPI_Reduce, which performs a reduction operation on distributed data.
MPI can be used to accelerate many-body dynamics simulation by using domain decomposition approaches, for example. In this type of approach, the simulation domain is divided into a number of sub-domains. Each sub-domain is simulated by a separate MPI process, with appropriate communication and synchronization. This approach will be described in Chapter 5, with implementation details given in Chapter 6.

### 3.4 OpenMP Parallelism

OpenMP is a portable API for programming parallel applications in shared memory architectures [47]. The motivation for OpenMP was to offer a low overhead, high level means to specify and achieve parallelism in source code. To this end, OpenMP provides directives which the compiler uses to instantiate parallel execution regions. The use of directives allows most OpenMP code to be portable and scalable, to run on a wide variety of machines, and to take advantage of available hardware resources. Moreover, on a machine without the OpenMP libraries the directive will be ignored, and the same code will execute sequentially, leading to very portable, scalable code.

OpenMP relies on user-specified parallel regions in the code. The main program is the master thread, and is responsible for launching a set of parallel worker threads to perform the parallel computations. The programmer is responsible for identifying regions of the code which may be executed in parallel and ensuring that there is no data dependency between parallel threads. In general, when a section of code is marked as parallel, the master thread divides the work among the available threads.

Perhaps the simplest and most useful OpenMP construct is `omp parallel for`, which distributes the iterations of a for-loop among the available parallel threads. For example, in Listing 3.2, each iteration of the for-loop is independent of the others. Therefore, the iterations can be distributed to parallel threads and performed in any order. Note that `omp parallel for` has an implicit synchronization at the end of the parallel section.

Listing 3.2: Example of OpenMP parallel for-loop

```
#pragma omp parallel for
for (int i=0; i<MAX; i++) {
```
res[i]=foo(i);
}

OpenMP also allows for more complex threading. For example, a programmer can identify and mark sections of code or tasks which can be executed in parallel. Additionally, there are more advanced concepts to guard against race conditions and other conflicts. For example, the critical construct allows only one thread at a time to enter the marked region.

In Listing 3.3, note that there is no race condition because only one thread at a time may increment the sum.

Listing 3.3: Example of OpenMP parallel for-loop with critical section

```c
sum=0.0;
#pragma omp parallel for shared(sum)
for(int i=0; i<N; i++) {
#pragma omp critical
    sum+=a[i]*b[i];
}
```

OpenMP can be used to accelerate many-body dynamics simulation by parallelizing operations which occur in loops. For example, an operation which is applied to all bodies in the system in a for-loop can be accelerated via OpenMP. Such an approach will be described in Chapter 6.
Chapter 4

Iterative Methods for DVI Formulation

The purpose of this chapter is to describe a collection of iterative methods which can be used to solve the Cone Complementarity Problem (CCP) of the DVI Formulation (see Chapter 2). First, the Jacobi and Gauss-Seidel methods are described, which are the most commonly used methods in the literature. Then, a set of methods are developed which will demonstrate significant improvements over existing methods in terms of accuracy and computation time. These methods include: the Gradient-Projected Minimum Residual Method (GPMinres) for frictionless problems, Accelerated Projected Gradient Descent Method (APGD), and the Primal-Dual Interior Point Method (PD) for frictional problems. Note that the PD method is a second order method, meaning it uses information about the second derivative of the objective function, while all of the other methods are first order methods, meaning they only use information about the first derivative of the objective function.

In particular, this chapter motivates the selected methods, provides the implementation details, and demonstrates their performance for a collection of representative problems. Additionally, several residuals are considered to motivate the selection of the residual used to judge convergence.

4.1 Jacobi and Gauss-Seidel

The algorithm most commonly used to solve the CCP of the DVI formulation is based on an extension of the the Gauss-Jacobi and Gauss-Seidel methods with over-relaxation for symmetric convex linear complementarity problems [48]. These methods were extended to a general form capable of solving the CCP [21]. In [21], a proof of convergence and a demonstration of scalability
are given for the general algorithm. Two specific and commonly used versions of the general algorithm are similar to projected Jacobi and projected Gauss-Seidel methods. Both are proved to converge by the same arguments in [21]. The algorithms for both methods are given below. More implementation details can be found in [14].

In both algorithms below, the matrix $B$ is block diagonal. Each $3 \times 3$ block $B_i$ is given as $B_i = \frac{1}{g_i}I$, where

$$g_i = \frac{\text{tr}(D_i^T M^{-1} D_i)}{3},$$

and each $i \in [1, N_c]$ corresponds to a single contact.

This choice was inspired by the Gauss-Jacobi idea of using the inverse of the diagonal of the system matrix, with the inclusion of averaging to improve performance for systems with large mass ratios [14].

Further, note that both methods take as inputs the matrix $N$ and vector $r$ (recall Equation 2.46), the tolerance, $\tau$, the maximum number of iterations to perform, $N_{max}$, and the initial guess for the vector of unknowns, usually taken as $\gamma_0 = 0$. Both methods rely on a measure of convergence, $r$, computed in lines 4 and 6 of the Jacobi and Seidel algorithms, respectively. Note that $r$ is a function of the unknown, $\gamma_{(k+1)}$, because it should indicate when the iterative process is nearing a solution. The quantity $r$ is also called the residual norm, and is further described in Section 4.6.

**Algorithm Jacobi**($N$, $r$, $\tau$, $N_{max}$, $\gamma_0$)

1. for $k := 0$ to $N_{max}$
2. $\hat{\gamma}_{(k+1)} = \Pi_K \left( \gamma_{(k)} - \omega B \left( N \gamma_{(k)} + r \right) \right)$
3. $\gamma_{(k+1)} = \lambda \hat{\gamma}_{(k+1)} + (1 - \lambda) \gamma_{(k)}$
4. $r = r \left( \gamma_{(k+1)} \right)$
5. if $r < \tau$
6. break
7. endfor
8. return Value at time step $t^{(l+1)}$, $\gamma^{(l+1)} := \gamma_{(k+1)}$. 
Algorithm Gauss-Seidel($N$, $r$, $\tau$, $N_{max}$, $\gamma_0$)

1. for $k := 0$ to $N_{max}$
2.     for $i = 1$ to $n_c$
3.         $\hat{\gamma}_{i,(k+1)} = \prod_K \left( \gamma_{i,(k)} - \omega B_i (N \gamma_k + r)_i \right)$
4.         $\gamma_{i,(k+1)} = \lambda \hat{\gamma}_{i,(k+1)} + (1 - \lambda) \gamma_{i,(k)}$
5.     endfor
6.     $r = r \left( \gamma_{k+1} \right)$
7.     if $r < \tau$
8.         break
9.     endfor
10. return Value at time step $t^{(l+1)}$, $\gamma^{(l+1)} := \gamma_{(k+1)}$.

The Jacobi and Gauss-Seidel methods can be seen to be very similar. In fact, the only difference
between the two is that in Gauss-Seidel, once a contact has been updated in a given iteration,
its new value is used for the remainder of the iteration. In Jacobi, all contacts are updated in a
given iteration using the values from the previous iteration. This difference has two important
consequences. First, Gauss-Seidel usually performs better than Jacobi, especially in the case of
redundant contacts [14]. Second, Jacobi is more amenable to parallel computing as it has no
dependencies between updates of various contacts in a given iteration. Therefore, Gauss-Seidel is
favored in sequential computing scenarios, but Jacobi is required if parallel computing is desired.
The Jacobi-type method has been implemented in parallel to run on the Graphics Processing Unit
(GPU) [49, 50, 44, 51]. Additionally, in terms of performance, the methods can be sensitive to the
value of the parameter $\omega$. For example, the performance of the Seidel method improved when the
parameter was increased from 0.3 to 1.0. However, the Jacobi method often failed with the higher
value. Therefore, for performance and stability reasons, the value $\omega = 0.3$ was used for Jacobi,
and $\omega = 1.0$ was used for Seidel in the remainder of this work.
4.2 Krylov Methods

Projection methods form a large class of iterative methods which can be used to find an approximate solution to a linear system \( Ax = b \), where \( A \in \mathbb{R}^n \). In projection methods, the approximate solution \( \tilde{x} \) is found within a subspace \( K_m \) of \( \mathbb{R}^n \) of dimension \( m \), where \( m < n \). One way to accomplish this is to constrain the residual, \( b - Ax \), to be orthogonal to \( m \) linearly independent vectors, which form another subspace \( L_m \) of dimension \( m \) [52].

\[
\text{Find } \tilde{x} \in x_0 + K_m \text{ such that } b - A\tilde{x} \perp L_m \tag{4.2}
\]

Krylov subspace methods are a subset of Projection Methods where, \( K_m \) is the Krylov subspace. The Krylov subspace of dimension \( m \) is defined as follows, where \( r_0 = b - Ax_0 \):

\[
K_m (A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}. \tag{4.3}
\]

In general, Krylov methods effectively generate a sequence of approximations, \( x_m \), of the following form:

\[
A^{-1}b \approx x_m = x_0 + \alpha_0 b + \alpha_1 Ab + \alpha_2 A^2b + \cdots + \alpha_{m-1} A^{m-1}b \tag{4.4}
\]

The dimension of the Krylov subspace is increased at each iteration. Therefore, the approximate solution \( x_m \) becomes increasingly more accurate and is expected to be exact after at most \( n \) iterations when the Krylov subspace spans all of \( \mathbb{R}^n \). In practice, a solution of acceptable accuracy is often reached in much fewer than \( n \) iterations.

At each iteration \( k \), a Krylov method should compute \( x_k = \tilde{x} \), where \( \tilde{x} \) is the ‘best’ vector from \( K_k \) such that \( b - A\tilde{x} \in L_k \). Different Krylov methods can be developed by choosing different sets of constraints; i.e., different choices of \( L_m \), and different measures of ‘best’ when choosing a vector satisfying the stated conditions.

For example, the choice \( L_k = AK_k \) leads to the the Generalized Minimum Residual (GMRES) and Minimum Residual (MINRES) methods where the approximate solution \( x_k \in K_k \) is the vector
which minimizes $||b - Ax_k||_2$. In this case, it can be shown that the solution $\tilde{x}$ generated by starting from $x_0$ is the vector which minimizes the 2-norm of the residual over the space $x_0 + K$.

$$\tilde{x} = \arg \min_{x \in x_0 + K} ||b - Ax||_2$$

Similarly, the choice $L_k = K_k$ leads to the conjugate gradient (CG) method when $A$ is symmetric, real and positive definite. In this case, it can be shown that the solution $\tilde{x}$ generated by the CG method starting from $x_0$ is the vector which minimizes the A-norm of the error over the space $x_0 + K$. Note that $||v||_A^2 = v^T Av$.

$$\tilde{x} = \arg \min_{x \in x_0 + K} ||x^* - x||_A$$

### 4.2.1 MINRES

One specific Krylov method which is used in the GPMINRES algorithm developed in the following section is the Minimum Residual (MINRES) method. MINRES was developed by Paige and Saunders to handle the case when the matrix $A$ is symmetric and possibly indefinite [53]. In this case, the subspace of constraints is $L_k = AK_k$.

In the MINRES algorithm, the vectors of the Krylov subspace $K_m$ are not used directly, but instead an orthonormal basis $\{q_1, q_2, \ldots, q_m\}$ which spans $K_m$ is used. This orthonormal basis is generated via the Lanczos iteration for symmetric matrices [54]. MINRES also takes advantage of a three-term recurrence which is possible due to the symmetry of $A$. Therefore, there is no need to explicitly store the entire orthonormal basis $\{q_1, q_2, \ldots, q_m\}$. Instead, the solution vector $x_m$ can be updated via a three-term recurrence, saving significant computation and storage effort [55].

The MINRES algorithm can be seen below [56].

**Algorithm MINRES($A, b, \tau, N_{max}, x_0$)**

1. $v_0 = 0$

2. $r_1 = b - Ax_0$

3. $\beta_1 = ||r_1||_2$

4. $\eta = \beta_1$
\begin{align*}
    \gamma_0 &= \gamma_1 = 1 \\
    \sigma_0 &= \sigma_1 = 0 \\
    w_{-1} &= w_0 = 0 \\
    \text{for } k := 1 \text{ to } N_{\text{max}} & \\
    v_k &= \frac{r_k}{\beta_k} \\
    \alpha &= v_k^T A v_k \\
    r_{k+1} &= A v_k - \alpha v_k - \beta_k v_{k-1} \\
    \beta_{k+1} &= ||r_{k+1}||_2 \\
    \delta &= \gamma_k \alpha - \gamma_{k-1} \sigma_k \beta_k \\
    \rho_1 &= \sqrt{\delta^2 + \beta_{k+1}^2} \\
    \rho_2 &= \sigma_k \alpha + \gamma_{k-1} \gamma_k \beta_k \\
    \rho_3 &= \sigma_{k-1} \beta_k \\
    \gamma_{k+1} &= \frac{\delta}{\rho_1} \\
    \sigma_{k+1} &= \frac{\beta_{k+1}}{\rho_1} \\
    w_k &= \frac{v_k - \rho_3 w_{k-2} - \rho_2 w_{k-1}}{\rho_1} \\
    x_k &= x_{k-1} + \gamma_{k+1} \eta w_k \\
    \eta &= -\sigma_{k+1} \eta \\
    \epsilon &= \epsilon |\sigma_{k+1}| \\
    \text{if } \frac{\epsilon}{||r_{1}||} < \tau & \\
    \text{break} & \\
    \text{endfor} & \\
    \text{return } x_k .
\end{align*}

4.3 Gradient-Projected Minimum Residual Method (Frictionless Problems)

The Gradient-Projected Minimum Residual Method has been developed for frictionless problems. When \( \mu_i = 0 \), the optimization problem which must be solved at each time step, Eq. 2.46, is modified. Specifically, the conic constraints become bound constraints. The frictionless problem is stated as
\[
\min q(\gamma) = \frac{1}{2} \gamma^T N \gamma + r^T \gamma
\]
subject to \( \gamma_i \geq 0 \) for \( i = 1, 2, \ldots, N_c \),

where \( \gamma = [\gamma_{1,n}, \gamma_{2,n}, \ldots, \gamma_{N_c,n}]^T \) is the vector of normal contact impulses.

This is an active set algorithm, and draws on ideas presented in [57, 58]. The active set is defined as the set of constraints which are satisfied with equality at a given iteration \( \gamma_k \). For the frictionless problem, this means that the active set is defined as

\[
\mathcal{A}(\gamma_k) = \{i | \gamma_i = 0\},
\]

As in [57], the algorithm has two stages. In the first stage, projected gradient descent is used to decrease the objective function while maintaining feasibility. All components of \( \gamma \) may be updated at each gradient descent step, which means the associated active set may change arbitrarily. At some point, based on the improvement in objective function value and/or the evolution of the active set, the gradient descent stage is stopped. Solution moves to the second stage, where a more aggressive Krylov method is used to solve an unconstrained problem in the variables which were not in the active set. If solution again stagnates in stage two, the algorithm returns to stage one to identify a possibly new active set.

In [59], Burke and Moré proved that this approach (alternating projected gradient descent with the conjugate gradient Krylov method) has finite termination and converges to the unique solution of the problem when the objective function is strictly convex. For quadratic objective functions, this corresponds to the quadratic term matrix being positive definite. In the case of the frictionless multibody dynamics problem in Equation 4.7, the matrix \( N \) is positive semi-definite, and the associated convex problem may have multiple solutions. Therefore, the method developed here no longer seeks a unique solution, so the conjugate gradient in [59] was replaced with a minimum residual Krylov subspace method (MINRES) [52].

**Algorithm GPMINRES**

\[
\text{Algorithm GPMINRES}(N, r, \tau, \eta_1, \eta_2, N_{\text{max}}, M_{\text{max}})
\]

(1) \( \gamma_0 := 0_{nc} \)
\( k := 0 \) to \( N_{\text{GP}}^{\text{MINRES}} \)
\( y(0) = \gamma(k) \)
\( j := 0 \) to \( N_{\text{GP}}^{\text{MINRES}} \)
\( \alpha = \frac{[\nabla q(y(j))]^T - [\nabla q(y(j))]}{[\nabla q(y(j))]^T \cdot [\nabla q(y(j))] } \)
repeat
\( y(j+1) = \Pi_K(y(j) - \alpha \nabla q(y(j))) \)
until \( q(y(j+1)) \leq q(y(j)) + \mu \langle \nabla q(y(j)), y(j+1) - y(j) \rangle \)
if \( A(y(j)) = A(y(j+1)) \)
break //Stuck on a face, switch to minres for faster search
if \( q(y(j)) - q(y(j+1)) \leq \eta_1 \max \{ q(y(l+1)) - q(y(l)) : 1 \leq l \leq j \} \)
break //Convergence not peppy, bail out, move on to minres
endwhile

\( \gamma(k) := y(j) \)

Determine active set \( A(\gamma(k)) \) and selection matrix \( Z_k \in \mathbb{R}^{n \times m_k} \) that goes
with the free set
\( N_k \equiv Z_k^T \cdot N \cdot Z_k \)
\( r_k \equiv Z_k^T \cdot \nabla q(\gamma(k)) \)
\( w(0) = 0_{m_k} \)
\( q(0) = \frac{1}{2} \langle w(0), N_k w(0) \rangle + \langle r_k, w(0) \rangle \)
\( j := 0 \) to \( N_{\text{MINRES}}^{\text{CG}} \)

Improve value of \( w(j) \rightarrow w(j+1) \) by taking one MINRES step
\( q(j+1) = \frac{1}{2} \langle w(j+1), N_k w(j+1) \rangle + \langle r_k, w(j+1) \rangle \)
\( \delta(j+1) = q(j) - q(j+1) \)
\( j = j + 1 \)
if \( \delta(j+1) \leq \eta_2 \max \{ \delta(l) : 1 \leq l < j \} \) or \( j = N_{\text{CG}}^{\text{MAX}} \)
//Stalling or reached max number of iterations
\( d(k) = Z_k \cdot w(j+1) \) //The prolongation operation
\( i = 0 \)
\(\alpha = 1\)

repeat
\(\gamma_{(k+1)} = \Pi_K(\gamma_{(k)} + \alpha d_{(k)})\)
\(\alpha = \frac{\alpha}{2}\)
\(i = i + 1\)
until \(i \leq N_{PS}^{\max}\) or \(q(\gamma_{(k+1)}) \leq q(\gamma_{(k)}) + \mu \langle \nabla q(\gamma_{(k)}), \gamma_{(k+1)} - \gamma_{(k)} \rangle\)
\(\text{if } A(\gamma_{(k+1)}) = B(\gamma_{(k+1)})\)
//Active and binding sets are the same; maybe we are there
\(\text{if } \eta_2 < \tau\)
\(\text{converged=true}\)
\(\text{break} //\text{gets out of the MINRES loop, infact GPMINRES is done}\)
\(\text{else}\)
\(\eta_2 = \frac{\eta_2}{2} //\text{continue with MINRES, might be close to solution}\)
\(N_{max}^{\minres} = N_{max}^{\minres} + N_{AI}^{\minres}\)
\(\text{endif}\)
\(\text{else}\)
//This is the case where \(N_{max}^{\minres}\) has been reached
\(\text{converged=false}\)
\(k = k + 1 //\text{finished one GPMINRES iteration; move on}\)
\(\text{break} //\text{gets out of MINRES}\)
\(\text{endif}\)
\(\text{endif}\)
\(\text{endfor} //\text{the MINRES loop}\)
\(\text{if } \text{converged=true}\)
\(\text{break} //\text{Approximate solution found}\)
\(\text{endif}\)
\begin{align}
55\quad r &= r (\gamma_{k+1}) \\
56\quad \text{if } r < \tau \\
57\quad \text{break} \\
58\quad \text{endif} \\
59\quad \text{endfor} \\
60\quad \text{return Value at time step } t^{(l+1)}, \gamma^{(l+1)} := \gamma_{(k+1)}.
\end{align}

### 4.4 Accelerated Projected Gradient Descent

Nesterov first proposed accelerated gradient schemes in 1983 [60]. Accelerated gradient descent methods can be seen as simple gradient descent methods with the introduction of ‘momentum’ in the search direction. Note that in this case, momentum refers to the concept that the search direction should depend on past iterations in addition to the current iteration. Instead of taking the search direction to be opposite of the gradient direction at the current iteration, the introduction of momentum effectively uses a weighted combination of the current and past gradient directions. The remainder of this section describes the origins of the accelerated projected gradient descent (APGD) method in the gradient descent method, the extension to constrained minimization, the introduction of acceleration, and a series of improvements to solve the targeted problem.

The original gradient descent method, often attributed to Cauchy [61], can be expressed as follows, where \( f(x) \) is a smooth function to be minimized and \( \alpha_k \) is a step size for iteration \( k \).

\[
x_{k+1} = x_k - \alpha_k \nabla f(x_k) \\
\text{where } \alpha_k = \arg \min_{\alpha} f(x_k - \alpha \nabla f(x_k))
\]

Under certain conditions, for example if \( f(x) \) is convex and Lipschitz continuous, convergence to the global solution can be guaranteed. Note that the scheme in Equations 4.9-4.10 solves an unconstrained optimization problem. The gradient descent method can be extended to the projected gradient descent method for solving constrained optimization problems under certain conditions (see, for example, [62]). Projected gradient descent takes the following form, where \( \Pi_C \) represents
projection onto the convex set $C$ and $\alpha_k$ is the step size which should satisfy some condition to ensure sufficient decrease:

$$x_{k+1} = \Pi_C (x_k - \alpha_k \nabla f(x_k)). \tag{4.11}$$

The projected gradient descent method can be shown to have a sub-linear rate of convergence when the objective function $f(x)$ is convex and $C$ is a convex set:

$$f(x_k) - f(x^*) \simeq O(1/k) \tag{4.12}$$

In [60], Nesterov developed a method with an improved convergence rate of $O(1/k^2)$. In fact, the method in [60] was shown to be an ‘optimal’ first-order method for smooth problems [63] in terms of its performance among all first-order methods, up to a constant.

The following set of equations represents one iteration of the accelerated gradient descent (AGD) scheme [64]. Note that $y_0 = x_0 \in \mathbb{R}^n$, $\theta_0 = 1$, $q \in [0, 1]$ is a tuning parameter, and $t_k$ is the step size for the current iteration.

$$x_{k+1} = y_k - t_k \nabla f(y_k) \tag{4.13}$$

$$\theta_{k+1} \text{ solves } \theta_{k+1}^2 = (1 - \theta_{k+1}) \theta_k^2 + q \theta_{k+1} + 1 \tag{4.14}$$

$$\beta_{k+1} = \frac{\theta_k (1 - \theta_k)}{\theta_k^2 + \theta_{k+1}} \tag{4.15}$$

$$y_{k+1} = x_{k+1} + \beta_{k+1} (x_{k+1} - x_k) \tag{4.16}$$

Assume $f(x)$ is convex and Lipschitz continuous with constant $L$; i.e., $||\nabla f(x) - \nabla f(y)||_2 \leq L ||x - y||_2, \forall x, y \in \mathbb{R}^n$. Then, the method described by equations 4.13-4.16 converges for any $t_k \leq 1/L$. In the above, note that $q = 1$ leads to $\theta_k = 1$, $\beta_k = 0$, and $y_k = x_k$ for all $k \geq 0$, which reduces to the gradient descent method. In general, the parameter $q$ can tune the performance of the method depending on the specifics of the objective function $f(x)$. For example, if $f(x)$ is also strongly convex, i.e., $\exists \mu > 0: f(x) \geq f(x^*) + (\mu/2) ||x - x^*||_2^2, \forall x \in \mathbb{R}^n$, then the optimal value is $q = \mu / L$, which achieves a linear convergence rate. If the objective function is not strongly
convex, or the strong convexity parameter $\mu$ is unknown, then it is often assumed that $q = 0$. Note that the original statement of the accelerated method in [60] had $q = 0$, so the convergence rate of $O\left(1/k^2\right)$ is still valid.

The AGD scheme can be extended to constrained optimization in the same way that gradient descent was extended to projected gradient descent. The resulting algorithm, called Accelerated Projected Gradient Descent (APGD) can be expressed by the following set of computations to be performed at each iteration $k \geq 0$. Once again, let $y_0 = x_0 \in \mathbb{R}^n$, and $\theta_0 = 1$.

\begin{align*}
x_{k+1} &= \Pi_C (y_k - t_k \nabla f(y_k)) \quad \text{(4.17)} \\
\theta_{k+1} &\text{ solves } \theta_{k+1}^2 = (1 - \theta_{k+1}) \theta_k^2 \quad \text{(4.18)} \\
\beta_{k+1} &= \frac{\theta_k (1 - \theta_k)}{\theta_k^2 + \theta_{k+1}} \quad \text{(4.19)} \\
y_{k+1} &= x_{k+1} + \beta_{k+1} (x_{k+1} - x_k) \quad \text{(4.20)}
\end{align*}

When $f(x)$ is convex and Lipschitz continuous with constant $L$, then the method described by Equations 4.17-4.20 converges for any $t_k \leq 1/L$. An equivalent algorithm was proved in [65] to converge with the same $O\left(1/k^2\right)$ rate as the AGD method.

### 4.4.1 Adaptive Step Size

This section describes an adaptation which allows the step size, $t_k$, to vary at each iteration while still guaranteeing convergence. This is necessary because the global Lipschitz constant may be unknown or too restrictive, implying that choosing a constant $t < 1/L$ may not achieve best performance. Two possible adaptive strategies are considered, both adapted from [66].

In the first, a local estimate of the Lipschitz parameter, $L_k$, is computed and used as long as the associated step, $t_k = 1/L_k$, is appropriate to preserve convergence (i.e., the following condition is satisfied):

\begin{equation}
f(x_{k+1}) \leq f(y_k) + \nabla f(y_k)^T (x_{k+1} - y_k) + \frac{L_k}{2} \|x_{k+1} - y_k\|^2 \quad \text{(4.21)}
\end{equation}
If Equation 4.21 is violated, then backtracking is performed until Equation 4.21 is satisfied. To this end, take $L_k = 2L_k$ until Equation 4.21 is satisfied, then proceed with the new estimate. With this approach, the step length $t_k$ will shrink as $L_k$ increases. Once $L_k \geq L$, no more backtracking steps will be needed and $t_k$ will be constant for the remaining iterations. Note that the initial estimate $L_0$ can be estimated as $L_0 = \|\nabla f (z_0) - \nabla f (z_1)\|_2 / \|z_0 - z_1\|_2$, for any points $z_0$ and $z_1$.

The second strategy differs from the first in that it allows the step to both grow and shrink throughout the iterative process. In this approach, the same backtracking is used to increase $L_k$ as needed to preserve convergence. However, $L_k$ may also be decreased to improve performance by taking larger steps when allowed locally. To this end, $L_k$ is decreased at each iteration according to $L_{k+1} = 0.9L_k$, and increased if needed to satisfy Eq. 4.21. Note that this approach may result in slightly more total backtracking steps, but may allow longer steps and therefore increased performance.

### 4.4.2 Adaptive Restart

This section describes an adaptation which may restart the method, or reset the momentum, periodically to improve performance of the iterative method. This is necessary because the performance of the iterative method can be sensitive to the amount of momentum applied. In Equations 4.17-4.20, note that $\beta_k \to 1$ which is characterized as high momentum. In [67], it is shown that high momentum leads to rippling in the objective function value and decreased performance of the iterative method. Therefore, it is suggested in [67] that the method be restarted when the momentum is no longer appropriate; i.e., if $\nabla f (y_{k-1})^T (x_k - x_{k-1}) > 0$ at iteration $k$, then set $\theta_k = 1$ and $y_k = x_k$. The justification for this rule is that the momentum is reset whenever the projection of the momentum term, $\beta_k (x_k - x_{k-1})$, onto the negative gradient, $-\nabla f (y_{k-1})$, is negative. This would imply that the momentum opposes the negative gradient, which is known to be a descent direction, and is therefore not helping the method. It is demonstrated in [67] that the adaptive restart scheme results in significant improvement for several test problems.
4.4.3 Fall-Back

To achieve monotone behavior in terms of the residual, a fall-back strategy is suggested. Such an approach is desirable when an iterative method is non-monotone and may be terminated prematurely due to some constraints. If a non-monotone iterative method is terminated at iteration \( k \), it is possible that the current iterate, \( x_k \), is not the best solution considered. In fact, the fall-back strategy should return \( \hat{x} = x_j \), where \( x_j \) is the ‘best’ solution among \( i \in [0, k] \). The measure of ‘best’ may be made in terms of either the objective function value, \( f(x_i) \), or the residual norm, \( r(x_i) \).

In this work, the fall-back is performed in terms of the norm of the residual. Therefore, the fall-back strategy returns \( \hat{x} = x_j \), where \( j = \arg \min_{i \in [0, k]} r(x_i) \). Note that details of the residual used will be given in Section 4.6.

4.4.4 APGD Algorithm

This section states the overall algorithm obtained when applying the APGD method to the problem in Equation 2.46. As stated, the algorithm includes an adaptive step size which may both shrink and grow, an adaptive restart scheme based on the gradient, and a fall-back strategy to allow early termination.

\[
\text{ALGORITHM APGD}(\mathbf{N}, \mathbf{r}, \tau, N_{\text{max}})
\]

\begin{enumerate}
\item \( \gamma_0 = 0_{nc} \)
\item \( \hat{\gamma}_0 = 1_{nc} \)
\item \( y_0 = \gamma_0 \)
\item \( \theta_0 = 1 \)
\item \( L_k = \frac{||N(\gamma_0 - \hat{\gamma}_0)||_2}{||\gamma_0 - \hat{\gamma}_0||_2} \)
\item \( t_k = \frac{1}{L_k} \)
\item \textbf{for} \( k := 0 \) \textbf{to} \( N_{\text{max}} \)
\item \( g = \mathbf{N}y_k - \mathbf{r} \)
\item \( \gamma_{k+1} = \Pi_K (y_k - t_k g) \)
\end{enumerate}
while $\frac{1}{2}y_k^T N \gamma_{k+1} - \gamma_k^T r \geq \frac{1}{2}y_k^T N y_k - y_k^T r + g^T (\gamma_{k+1} - y_k) + \frac{1}{2}L_k ||\gamma_{k+1} - y_k||^2_2$

$L_k = 2L_k$
$t_k = \frac{1}{L_k}$
$\gamma_{k+1} = \Pi_K (y_k - t_k g)$

endwhile

$\theta_{k+1} = -\frac{\theta_k^2 + \theta_k \sqrt{\theta_k^2 + 4}}{2}$
$\beta_{k+1} = \theta_k \frac{1 - \theta_k}{\theta_k^2 + \theta_{k+1}}$
$y_{k+1} = \gamma_{k+1} + \beta_{k+1} (\gamma_{k+1} - y_k)$
$r = r (\gamma_{k+1})$
if $r < \epsilon_{\min}$
$r_{\min} = r$
$\hat{\gamma} = \gamma_{k+1}$
endif
if $r < \tau$
break
endif
if $g^T (\gamma_{k+1} - \gamma_k) > 0$
y_{k+1} = \gamma_{k+1}
$\theta_{k+1} = 1$
endif
$L_k = 0.9 L_k$
t_k = \frac{1}{L_k}$
endfor
return Value at time step $t_{l+1}, \gamma^{l+1} := \hat{\gamma}$. 
4.4.5 Notes on the Performance of APGD

This section provides a simple numerical experiment to demonstrate the performance of the APGD method, focusing on the effects of the modifications to the base method which were described in previous sub-sections. The problem considered here is that of Equation 2.46 for a single time step of a simulation with 1,000 spherical bodies contained in a cylindrical container (see Figure 4.1).

The matrix $N$ and vector $r$ which specify the problem were extracted from a dynamic simulation at the desired time step. Together with knowledge that $\mu_i = 0.25$ for all 3,692 contacts in the problem, this data completely specifies the minimization problem of interest and allows application of the APGD method.

Figures 4.2-4.4 show the effects of the proposed modifications to the base method for this sample problem. Figure 4.2 shows the effect of the fall-back strategy. In Figure 4.2, the upper plots show the objective function value with fall-back (left) and without fall-back (right). The lower plots show the residual value with fall-back (left) and without fall-back (right). Here, the fall-back was implemented with respect to the residual, so it can be seen that the residual is monotone when fall-back is used. No matter what step-length rule is used, the fall-back strategy results in a monotone residual. If fall-back was not used, the lower-right plot shows how the final iteration may have a larger residual than a previous iteration. Therefore, fall-back is considered necessary and will be
Figure 4.2: Objective function and residual values for APGD with and without fall-back.
Figure 4.3: Comparison of objective function, residual, and step length for APGD with and without adaptive restarting.
Figure 4.4: Comparison of objective function, residual, and step length for APGD with and without adaptive step size.
used in the remaining comparisons. The only drawback to the fall-back method is the requirement of one extra vector of storage and one extra comparison per iteration, but these are minor concerns.

Figure 4.3 shows the effect of the adaptive restart strategy for each step-length rule. In Figure 4.3, the first column of plots shows the objective function, while the second column shows the residual norm at each iteration. Each row of plots corresponds to a different step-length rule. The green circles mark the iterations at which the momentum was restarted. Using the adaptive restart rule leads to improvements in terms of both the objective function value and the residual for all three of the considered step-length rules.

Figure 4.4 shows the effect of the adaptive step size strategies. In Figure 4.4, the left column of plots does not use restarting, while the right column uses adaptive restarting based on the gradient. The upper row of plots shows the objective function, the middle row shows the residual, while the lower row shows the step length at each iteration. Using the adaptive restart rule leads to improvements in terms of both the objective function value and the residual for all three of the considered step-length rules. This can be understood by considering the step-length plots, which show that the fixed step length rule, where \( t_k = 1/L \), was the most restrictive, resulting in the shortest steps and the worst performance. The shrink step rule, where the step length is halved when necessary to guarantee convergence, resulted in slightly longer steps and slightly improved performance. Finally, the shrink/grow step rule, where the step is halved when necessary but allowed to grow otherwise, resulted in the longest steps on average and even better performance. These same trends can be observed whether or not the adaptive restart scheme is used.

Overall, the best performance is observed when all three modifications are used. This motivates the inclusion of these improvements in the NAPG algorithm given in Section 4.4.4.

4.5 **Primal-Dual Interior Point Method**

This section describes the Interior Point (IP) method for solving the cone-constrained quadratic optimization problem in Equation 2.46. Recall that the IP method is a second order method, and therefore uses information about the second derivative of the objective function, while all of the previously described methods were first order methods, meaning they only used information
about the first derivative of the objective function. The remainder of this section will develop
the background of IP methods, and then give a Primal-Dual IP method capable of solving the
optimization problem of interest.

In general, IP methods can solve convex optimization problems that include inequality con-
straints:

\[
\begin{align*}
\min & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m
\end{align*}
\]  

In Equation 4.22, \( f_0(x), \ldots, f_m(x) : \mathbb{R}^n \to \mathbb{R} \) should be convex and twice continuously
differentiable. Then, the Karush-Kuhn-Tucker (KKT) conditions can be expressed as follows,
where \( x^* \in \mathbb{R}^n \) and \( \lambda^* \in \mathbb{R}^m \) are the optimal primal and dual respectively.

\[
\begin{align*}
f_i(x^*) & \leq 0, \quad i = 1, \ldots, m \quad (4.23) \\
\lambda^*_i & \geq 0, \quad i = 1, \ldots, m \quad (4.24) \\
\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda^*_i \nabla f_i(x^*) & = 0 \quad (4.25) \\
\lambda^*_i f_i(x^*) & = 0, \quad i = 1, \ldots, m \quad (4.26)
\end{align*}
\]

To enable the solution of problem 4.22, the constraints are moved into the objective function
via an indicator function, resulting in the following equivalent unconstrained problem,

\[
\min f_0(x) + \sum_{i=1}^{m} I(f_i(x)) ,
\]  

where \( I : \mathbb{R} \to \mathbb{R} \) is the indicator function for non-positive reals,

\[
I(z) = \begin{cases} 
0 & \text{if } z \leq 0 \\
\infty & \text{if } z > 0 
\end{cases}
\]  

However, the resulting objective function is not differentiable. Therefore, the indicator function
is replaced with a differentiable approximation, specifically the logarithmic barrier function
\[ B(z) = -\frac{1}{t} \log(-z), \quad (4.29) \]

where \( z < 0 \), and \( t > 0 \) is a parameter which controls the strength of the barrier. Note that \( B(z) \to I(z) \) as \( t \to \infty \).

In this manner, each of the \( m \) constraints can be added to the objective function as \( B(f_i(x)) \), resulting in an unconstrained optimization problem which approximates the original problem. Note that the approximation approaches the original problem as \( t \to \infty \).

\[
\min f_0(x) + \sum_{i=1}^{m} B(f_i(x)) \quad (4.30)
\]

To solve the problem in Equation 4.30, a Primal-Dual IP method is proposed. This method can be obtained by considering the KKT conditions of problem 4.30, stated as follows:

\[
f_i(x) < 0, \quad i = 1, \ldots, m \quad (4.31)
\]

\[
\nabla f_0(x) + \sum_{i=1}^{m} -\frac{1}{t f_i(x)} \nabla f_i(x) = 0 \quad (4.32)
\]

Now, define \( \lambda_i = -\frac{1}{t f_i(x)} \), and write the modified KKT conditions as follows:

\[
f_i(x) < 0, \quad i = 1, \ldots, m \quad (4.33)
\]

\[
\lambda_i > 0, \quad i = 1, \ldots, m \quad (4.34)
\]

\[
\nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) = 0 \quad (4.35)
\]

\[
-\lambda_i f_i(x) = \frac{1}{t}, \quad i = 1, \ldots, m \quad (4.36)
\]

Note that the modified KKT conditions in Equations 4.33-4.36 are very similar to those in Equations 4.23-4.26. Only the last equation in each set is different, and it is clear that Equation 4.36 approaches Equation 4.26 as \( t \to \infty \).
The Primal-Dual IP method is obtained by applying Newton’s Method to the modified KKT conditions. Here, it is convenient to define the residual of this system of equations for a certain value of $t$ as

$$
\mathbf{r}_t (\mathbf{x}, \lambda) = \begin{bmatrix} \nabla f_0 (\mathbf{x}) + \nabla \mathbf{f} (\mathbf{x})^T \lambda \\ -\text{diag} (\lambda) \mathbf{f} (\mathbf{x}) - \frac{1}{t} \mathbf{1} \end{bmatrix} = 0.
$$

(4.37)

Note that Equation 4.37 makes use of the following terms:

$$
\mathbf{f} (\mathbf{x}) = \begin{bmatrix} f_1 (\mathbf{x}) \\ \vdots \\ f_m (\mathbf{x}) \end{bmatrix}, \; \nabla \mathbf{f} (\mathbf{x}) = \begin{bmatrix} \nabla f_1 (\mathbf{x})^T \\ \vdots \\ \nabla f_m (\mathbf{x})^T \end{bmatrix}, \; 1 = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}
$$

(4.38)

Newton’s method can be applied to solve Equation 4.37

$$
\begin{bmatrix} \mathbf{x} + \Delta \mathbf{x} \\ \lambda + \Delta \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} - \nabla \mathbf{r}_t (\mathbf{x}, \lambda)^{-1} \mathbf{r}_t (\mathbf{x}, \lambda).
$$

(4.39)

Rather than computing the Newton step as in Equation 4.39, it is advantageous to compute the search direction by solving the following linear system:

$$
\begin{bmatrix} \nabla^2 f_0 (\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \nabla^2 f_i (\mathbf{x}) & \nabla \mathbf{f} (\mathbf{x})^T \\ -\text{diag} (\lambda) \nabla \mathbf{f} (\mathbf{x}) & -\text{diag} (\mathbf{f} (\mathbf{x})) \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x} \\ \Delta \lambda \end{bmatrix} = -\mathbf{r}_t (\mathbf{x}, \lambda)
$$

(4.40)

Note that Equation 4.40 represents one Newton step for the linear system of Equation 4.37, and a given value of $t > 0$. The idea of the Primal-Dual IP method is to drive $t \to \infty$ as we take Newton steps of the form in Equation 4.40. The PD-IP method can be described in pseudocode as follows [68]:

```
ALGORITHM PD-IP($f_0, f_1, \ldots, f_m$, $\mu \geq 1$, $\epsilon$)
(1) while $||\mathbf{r}_t (\mathbf{x}, \lambda)||_2 > \epsilon$
(2) Compute $t = \frac{\mu m}{\eta}$
(3) Compute search direction $[\Delta \mathbf{x}^T \Delta \lambda^T]^T$
```
(4) Compute step length \( s > 0 \) via line search

(5) Update: \( x = x + s\Delta x, \lambda = \lambda + s\Delta \lambda \)

(6) \textbf{endwhile}

(7) \textbf{return} Solution \( x^* = x, \lambda^* = \lambda \).

In the preceding algorithm, the line search can be performed via backtracking to ensure that \( f_i(x) < 0 \) and \( \lambda_i > 0 \) are maintained for all \( i \). The backtracking will require two parameters, \( \alpha, \beta \in (0, 1) \).

This is achieved by first directly computing the largest step length, \( s_{\text{max}} \), which maintains \( \lambda_i \geq 0, \forall i \):

\[
s_{\text{max}} = \min\{1, \min\{-\lambda_i/\Delta\lambda_i|\Delta\lambda_i < 0\}\}\]

Then, to ensure \( \lambda_i > 0, \forall i \), take \( s = 0.99s_{\text{max}} \). Next, repeatedly multiply \( s \) by \( \beta \) until we achieve \( f_i(x) < 0, \forall i \). Then, continue multiplying \( s \) by \( \beta \) until the following condition is satisfied:

\[
||r_i(x + \Delta x, \lambda + \Delta \lambda)||_2 \leq (1 - \alpha s) ||r_i(x, \lambda)||_2
\]

According to [68], it is common to take \( \mu = 10, \alpha \in [0.01, 0.1] \), and \( \beta \in [0.3, 0.8] \) in the preceding.

4.5.1 Application to CCP

Several terms are required when applying the PD-IP method to the CCP of Equation 2.46. Recall that \( \gamma = [\gamma_1^T, \gamma_2^T, \ldots, \gamma_{N_c}^T]^T \), where \( \gamma_i = [\gamma_{i,n}, \gamma_{i,u}, \gamma_{i,v}]^T \).

First, the objective function in this case is

\[
f_0(\gamma) = \frac{1}{2} \gamma^T N \gamma + r^T \gamma. \quad (4.43)
\]

Define \( f(\gamma) \) as follows:
\[ f_i (\gamma) = \begin{cases} \frac{1}{2} (\gamma_{i,u}^2 + \gamma_{i,v}^2 - \mu_i^2 \gamma_{i,n}^2) & : i \in [1, \ldots, N_c] \\ -\gamma_{(i-N_c),n} & : i \in [N_c + 1, \ldots, 2N_c] \end{cases} \]  

(4.44)

Therefore,

\[
\nabla f_i (\gamma) = \begin{bmatrix} 0 \\ \\
\vdots \\
0 \\
-\mu_i^2 \gamma_{i,n} \\
\gamma_{i,u} \\
\gamma_{i,v} \\
0 \\
\vdots \\
0 \\
\end{bmatrix}, \quad i = 1, \ldots, N_c, \quad \text{and} \quad \nabla f_i (\gamma) = \begin{bmatrix} -1 \\ \\
\vdots \\
0 \\
0 \\
0 \\
0 \\
\vdots \\
0 \\
\end{bmatrix}, \quad i = N_c + 1, \ldots, 2N_c. \]  

(4.45)

The following expressions for the second derivatives can be obtained.

\[
\nabla^2 f_i (\gamma) = \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \\
\vdots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & \cdots \\
-\mu_i^2 & 0 & 0 & 0 \\
\vdots & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & \cdots & \cdots & 0 \\
\end{bmatrix}, \quad i = 1, \ldots, N_c \]  

(4.46)

\[
\nabla^2 f_i (\gamma) = 0_{(3N_c \times 3N_c)}, \quad i = N_c + 1, \ldots, 2N_c. \]  

(4.47)
Let $A$ be the Newton step matrix as in Equation 4.40. Then we have the following, where $B = \nabla f (\gamma)^T \in \mathbb{R}^{3N_c \times 2N_c}$, $C = -\text{diag} (\lambda) \nabla f (\gamma) \in \mathbb{R}^{2N_c \times 3N_c}$, and $D = -\text{diag} (f (\gamma)) \in \mathbb{R}^{2N_c \times 2N_c}$.

$$A = \begin{bmatrix} N + \hat{M} & B \\ C & D \end{bmatrix}$$

(4.48)

Note that the matrix $\hat{M}$ is defined as follows:

$$\hat{M} = \sum_{i=1}^{2N_c} \lambda_i \nabla^2 f (\gamma) = \text{diag} (\hat{m}) \in \mathbb{R}^{3N_c \times 3N_c}$$

(4.49)

$$\hat{m} = [-\mu_1^2 \lambda_1, \lambda_1, \lambda_1, -\mu_2^2 \lambda_2, \lambda_2, \ldots, -\mu_{N_c}^2 \lambda_{N_c}, \lambda_{N_c}, \lambda_{N_c}]^T \in \mathbb{R}^{3N_c}$$

(4.50)

Next, let the right hand side of the Newton step matrix $r_t$ (see Equation 4.40) be partitioned as follows:

$$r_t = \begin{bmatrix} r_d \\ r_g \end{bmatrix}$$

(4.51)

Then, the Newton step for this problem is expressed as follows:

$$A \begin{bmatrix} \Delta \gamma \\ \Delta \lambda \end{bmatrix} = -r_t$$

(4.52)

An alternative, reduced form of this Newton step equation can be obtained by considering the Shur complement of the matrix $A$. Note that the matrix $D$ is easily invertible as it is diagonal with all entries positive.

$$\left( N + \hat{M} - BD^{-1}C \right) \Delta \gamma = BD^{-1}r_g - r_d$$

(4.53)

Then, $\Delta \lambda$ can be recovered easily.

$$\Delta \lambda = D^{-1} (-C \Delta \gamma - r_g)$$

(4.54)
4.5.2 PDIP Algorithm

**ALGORITHM PD**\( (N, r, \tau, N_{\text{max}}, \gamma_0) \)

1. \( f = f (\gamma_0) \)
2. \( \lambda_0 = -1/f \)
3. for \( k := 0 \) to \( N_{\text{max}} \)
4. \( f = f (\gamma_k) \)
5. \( \dot{\eta} = -f^T \lambda_k \)
6. \( t = \frac{\mu_m}{\eta} \)
7. \( A = A (\gamma_k, \lambda_k, f) \)
8. \( r_t = r_t (\gamma_k, \lambda_k, t) \)
9. Solve the linear system \( Ay = -r_t \)
10. \( s_{\text{max}} = \sup \{ s \in [0, 1] | \lambda + s \Delta \lambda \succeq 0 \} = \min \{ 1, \min \{ -\lambda_i/\Delta \lambda_i | \Delta \lambda_i < 0 \} \} \)
11. \( s = 0.99s_{\text{max}} \)
12. while \( \max \{ f (\gamma_k + s \Delta \gamma) > 0 \} \)
13. \( s = \beta s \)
14. endwhile
15. while \( \| r_t (\gamma_k + s \Delta \gamma, \lambda_k + s \Delta \lambda) \|_2 > (1 - \alpha s) \| r_t \|_2 \)
16. \( s = \beta s \)
17. endwhile
18. \( \gamma_{k+1} = \gamma_k + s \Delta \gamma \)
19. \( \lambda_{k+1} = \lambda_k + s \Delta \lambda \)
20. \( r = r (\gamma_{k+1}) \)
21. if \( r < \tau \)
22. break
23. endif
24. endfor
25. return Value at time step \( t_{l+1}, \gamma^{l+1} := \gamma_{k+1} \).
4.6 On the Choice of Stopping Criteria

This section describes studies undertaken to select a stopping criteria for the considered iterative methods. In other words, a measure was required which quantified ‘closeness’ to the solution of the cone complementarity problem (CCP). Then, a tolerance could be specified to stop the iterative process at an arbitrary level of accuracy. This is especially important given the non-unique attribute of the solution, as described in Section 2.1.6.

Specifically, this section describes four measures of the residual, including the motivation for their consideration. A set of numerical experiments is then considered to compare and contrast the various options. Finally, a conclusion is made regarding the use of stopping criteria in this application.

4.6.1 Description of Residuals

The first considered residual, \( r_1 \), is defined as

\[
\begin{align*}
\begin{bmatrix}
\text{...}, \min \left( (N\gamma + r)_{i,n}, 0 \right), \\
\text{...}
\end{bmatrix}^T &\in \mathbb{R}^{Nc}.
\end{align*}
\tag{4.55}
\]

Residual \( r_1 \) is proportional to the worst normal penetration among all contacts at each iteration. It is easily verified that \((N\gamma + r)_{i,n} = (D^T v + b)_{i,n} = D^T_{i,n} v + \frac{1}{h} \Phi_i, \) which is exactly the gap distance for each contact when scaled by \( h \). Recall that one of the original complementarity conditions stated that \( \Phi_i \geq 0 \). Therefore, if the non-penetration conditions are satisfied for all contacts, \( f_{1,i} \geq 0 \forall i \) and \( r_1 = 0 \).

The second considered residual, \( r_2 \), is defined as

\[
\begin{align*}
\begin{bmatrix}
\text{...}, \gamma_{i,n} (N\gamma + r)_{i,n}, \\
\gamma_{i,u} (N\gamma + r)_{i,u}, \\
\gamma_{i,v} (N\gamma + r)_{i,v}, \\
\text{...}
\end{bmatrix}^T &\in \mathbb{R}^{3N_c}.
\end{align*}
\tag{4.56}
\]

Residual \( r_2 \) is a measure of element-wise complementarity. Note that this element-wise complementarity is different than the cone-wise complementarity required by the CCP formulation. However, if this complementarity is achieved (i.e., \( r_2 = 0, \gamma_i \in \Upsilon_i, (N\gamma + r)_i \in \Upsilon^c_i \)), then the
complementarity in Eq. 2.27 is also achieved, so it is in some sense more strict than the actual required complementarity. In other words, if \( r_2 = 0 \) and the quantities involved are feasible, then Eq. 2.27 is satisfied and \( \gamma^* \) is optimal. However, there is no guarantee that \( r_2 = 0 \) at the optimal \( \gamma^* \). Therefore, given that \( \gamma \) and \((N\gamma + r)\) are feasible, it is sufficient for optimality that \( r_2 = 0 \), but not necessary.

The third considered residual, \( r_3 \), is defined as

\[
r_3 = \left\|f_3\right\|_\infty, \quad f_3 = [\ldots, \gamma_i^T (N\gamma + r)_i, \ldots]^T \in \mathbb{R}^{N_c}.
\] (4.57)

Residual \( r_3 \) is a measure of the worst violation of the actual complementarity condition as stated in Eq. 2.27. Note that all methods explicitly enforce \( \gamma_i \in \Upsilon_i \), while \( f_3, i = \gamma_i^T (N\gamma + r)_i = 0 \) at the solution. Therefore, if \( \gamma \) and \((N\gamma + r)\) are feasible, \( r_3 \) should be small close to the solution.

The fourth considered residual, \( r_4 \), is defined as

\[
r_4 = \left\|f_4\right\|_2, \quad f_4 = \frac{1}{gd} (\gamma - \Pi_K (\gamma - gd (N\gamma + r))) \in \mathbb{R}^{3N_c},
\] (4.58)

where \( gd \) is a small constant parameter, for example \( gd = 1 \times 10^{-6} \). Residual \( r_4 \) is a scaled version of the projected gradient, and is designed to ensure that the computed projected gradient direction is tangent to the constraint manifold at the current iterate. This is desirable because it is known that at the optimal solution, the gradient is orthogonal to the constraint manifold. Therefore, it is logical to use the component of the gradient which is tangent to the constraint manifold as a measure of error. To understand this residual, first note that if \( \gamma = \gamma^* \) is optimal, then \( \Pi_K (\gamma^* - gd (N\gamma^* + r)) = \gamma^* \), so \( f_4 = 0 \) and \( r_4 = 0 \) as expected. Second, consider the case when \( \gamma \) is not optimal. Then, it can be verified that

\[
\Pi_K (\gamma - gd (N\gamma + r)) = \gamma - gd f_4.
\] (4.59)

In the preceding, the left hand side is equivalent to taking a step of length \( gd \) in the negative gradient direction and projecting back to the feasible region. The right hand side says that the same point can be reached by taking a step of length \( gd \) in the direction opposite of \( f_4 \). In the limit, as
$g_d \to 0$, the direction $f_4$ approaches the plane tangent to the constraint manifold. Note that $r_4$ could be used to measure convergence for any value of $g_d$, but a small value was used in practice for the reasons just stated.

### 4.6.2 Comparison of Residuals for Stacking Problems

This section describes a set of numerical experiments used to study the residuals introduced in the previous section. The goal of these experiments was to understand which residual should be used to judge the convergence of the iterative methods used to solve the CCP problem. To this end, four simulations were carried out using the Chrono simulation tool [69] with increasing numbers of bodies. In each case, a collection of spherical bodies were released and allowed to settle in a cylindrical container. The cylinder had an inner radius of 0.3 m, while each particle had a radius of 0.1 m, a mass of 6.28 kg, and friction coefficient of 0.25. The simulations were performed with the Jacobi solver using 20,000 iterations per time-step to ensure that each time-step was solved accurately, leading to a system in a consistent state with as little violation of the non-penetration constraints as possible. Once each simulation had settled, the matrix $N$ and the vector $r$ specifying the quadratic optimization problem (in Eq. 2.46) for the current time-step were saved to files. Then, the various iterative methods were used to solve each problem while tracking each of the residuals previously described. The state of the systems considered can be seen in Fig. 4.5.

Note that the radius of the boundary cylinder was held fixed while an increasing number of spherical bodies was added. This results in taller stacks of bodies, which results in more challenging problems to solve. For the case with 32 bodies, the stack height was approximately 5 times the particle diameter. For the case with 256 bodies, the stack height was approximately 35 times the particle diameter.

Note that this benchmark was selected because it is representative of problems of interest, in particular granular dynamics. In granular dynamics simulations, there is often a collection of similarly-sized rigid bodies that have settled under gravity. In general, deeper stacks of bodies are more challenging to resolve. Therefore, the range of considered systems can be considered
Figure 4.5: Test problems for comparison of residuals

(a) 32 Bodies
(b) 64 Bodies
(c) 128 Bodies
(d) 256 Bodies
a simplified granular dynamics scenario, which captures the effect of tall stack heights on the convergence of the iterative methods.

The methods used in this test include Gauss-Jacobi with $\omega = 0.3$, Gauss-Seidel with $\omega = 1.0$, Spectral Projected Gradients (SPG), Gradient-Projected Minimum Residual with Friction (GPMF), Primal-Dual Interior Point Method, and Nesterov’s Accelerated Projected Gradient Descent (NAPG). Note that in practice, SPG and NAPG are implemented with a fall-back strategy due to their non-monotone behavior. However, in this set of tests the fall-back strategy was not used to avoid masking the true behavior of the targeted residuals. The SPG and GPMF methods are included here for completeness. The GPMF method is a generalization of the GPMINRES method for frictionless problems (see Section 4.3). Its behavior was not predictable, so it was not included in any detail in this document. The SPG method is described in [70].

**Test 1: 32 Bodies**

The state of the system can be seen in Fig. 4.5a, while Fig. 4.6 shows the objective function value plotted vs iteration count for the various iterative methods.

Residuals $r_1$ through $r_4$ for the 32 body stacking problem are plotted in Figures 4.7 through 4.10 respectively. Similar qualitative behavior can be observed in all four of the residual measures. In all residuals, the Primal-Dual method reduces the residual most quickly and to the smallest value. NAPG and SPG both achieve significantly steeper reductions in residual values than Jacobi or Gauss-Seidel. Finally, as expected, Gauss-Seidel performs slightly better than Jacobi method.

**Test 2: 64 Bodies**

The state of the system can be seen in Fig. 4.5b, while Fig. 4.11 shows the objective function value plotted vs iteration count for the various iterative methods.

Residuals $r_1$ through $r_4$ for the 64 body stacking problem are plotted in Figures 4.12 through 4.15 respectively. Note that $r_1$, $r_3$, and $r_4$ show qualitatively similar trends. Specifically, these residuals tend to decrease for all iterative methods. The relative performance among iterative methods is similar to that observed in Test 1. Specifically, the Primal-Dual method performs best,
Figure 4.6: Objective function value for various iterative methods applied to 32 body stacking problem.
Figure 4.7: Residual $r_1$ for various iterative methods applied to 32 body stacking problem.
Figure 4.8: Residual $r_2$ for various iterative methods applied to 32 body stacking problem.
Figure 4.9: Residual $r_3$ for various iterative methods applied to 32 body stacking problem.
Figure 4.10: Residual $r_4$ for various iterative methods applied to 32 body stacking problem.
Figure 4.11: Objective function value for various iterative methods applied to 64 body stacking problem.
followed by NAPG, then SPG. Finally, Jacobi, Seidel, and GPMF all show similar performance. Residual $r_2$ shows different behavior, in which residual values tend to increase, and achieve a steady value which is indistinguishable between iterative methods.

**Test 3: 128 Bodies**

The state of the system can be seen in Fig. 4.5c, while Fig. 4.16 shows the objective function value plotted vs iteration count for the various iterative methods.

Residuals $r_1$ through $r_4$ for the 128 body stacking problem are plotted in Figures 4.17 through 4.20 respectively. Note that $r_1$ and $r_4$ show qualitatively similar performance for all methods except Primal-Dual. Residuals $r_2$ and $r_3$ show extremely similar behavior for all methods. In general, the relative performance among methods is once again the same. SPG and NAPG show more rapid decrease and a smaller achieved residual value in terms of $r_1$ and $r_4$ than Jacobi, Seidel, and GPMF. Note that $r_1$ shows good performance for the Primal-Dual method, while $r_4$ shows no change in residual for the same method. This can be understood by recalling that the Primal-Dual method enforces that $\gamma_i \in \text{int}(\Upsilon_i)$. Therefore, the small step in $r_4$ which is taken inside of the projection operator is likely still in the cone, and the difference between $\gamma$ and the projected quantity is non-negligible, especially after scaling. This could be resolved by taking a projected gradient descent step before computing $r_4$ for the Primal-Dual method. Doing so allows $\gamma_i$ to be in the boundary of $\Upsilon_i$, and $r_4$ performs similarly to $r_1$.

**Test 4: 256 Bodies**

The state of the system can be seen in Fig. 4.5d, while Fig. 4.21 shows the objective function value plotted vs iteration count for the various iterative methods.

Residuals $r_1$ through $r_4$ for the 256 body stacking problem are plotted in Figures 4.22 through 4.25 respectively. Note that $r_1$ and $r_4$ show qualitatively similar performance for all methods except Primal-Dual. Residuals $r_2$ and $r_3$ show extremely similar behavior for all methods. Once again, $r_4$ shows poor performance for the Primal-Dual method for the same reason stated for the 128 body test.
Figure 4.12: Residual $r_1$ for various iterative methods applied to 64 body stacking problem.
Figure 4.13: Residual $r_2$ for various iterative methods applied to 64 body stacking problem.
Figure 4.14: Residual $r_3$ for various iterative methods applied to 64 body stacking problem.
Figure 4.15: Residual $r_4$ for various iterative methods applied to 64 body stacking problem.
Figure 4.16: Objective function value for various iterative methods applied to 128 body stacking problem.
Figure 4.17: Residual $r_1$ for various iterative methods applied to 128 body stacking problem.
Figure 4.18: Residual $r_2$ for various iterative methods applied to 128 body stacking problem.
Figure 4.19: Residual $r_3$ for various iterative methods applied to 128 body stacking problem.
Figure 4.20: Residual $r_4$ for various iterative methods applied to 128 body stacking problem.
Figure 4.21: Objective function value for various iterative methods applied to 256 body stacking problem.
Figure 4.22: Residual $r_1$ for various iterative methods applied to 256 body stacking problem.
Figure 4.23: Residual $r_2$ for various iterative methods applied to 256 body stacking problem.
Figure 4.24: Residual $r_3$ for various iterative methods applied to 256 body stacking problem.
Figure 4.25: Residual $r_4$ for various iterative methods applied to 256 body stacking problem.
Overall, residuals $r_1$ and $r_4$ show consistent performance for all considered algorithms and test problems. They both tend to decrease with increasing iterations. Residuals $r_2$ and $r_3$ can show different qualitative behavior depending on the problem. When the stack height is small (e.g., 32 or 64 body problems), these residuals show the expected behavior of decreasing with iterations. When the stack height is larger they tend to stay constant, with less difference between methods.

There are two important differences between $r_1$ and $r_4$. First, $r_4$ is a measure of the residual for both normal and frictional components, while $r_1$ is related only to the normal component. Secondly, $r_4$ does not work well for the Interior-Point method without a temporary gradient descent step each time the residual is saved.

Given the results in this section, the residual $r_4$ has been selected to measure convergence of the considered iterative methods. Residual $r_4$ provides predictable behavior and measures the residual for both normal and frictional components. Additionally, it is based on the optimality conditions for the constrained problem. Note that the numerical experiments in the following section use the $r_4$ residual for the stopping criteria.

### 4.7 Numerical Experiments

With the theoretical considerations concluded, it remains to study the convergence, accuracy, and performance of the developed iterative methods. First, a set of experiments will be performed which represent the solution of a single time-step of a many-body simulation. The purpose of these tests is to demonstrate convergence properties of the methods, by monitoring the evolution of the objective function value and the $r_4$ residual during the iterative solution process. Second, a set of simulation experiments will be performed with the Jacobi, Seidel, and APGD solvers. These tests will demonstrate the performance of the considered methods for actual rigid body simulations. For each simulation, the speed-up of the APGD solver compared to the Jacobi and Seidel solvers will be computed. The effect of problem size, friction coefficient, and stopping criteria on the relative performance of the methods will also be studied. For all experiments, the Jacobi solver uses $\omega = 0.3$ and the Seidel solver uses $\omega = 1.0$, as these values were found to give the best stability and performance.
Once again, the problems selected for these numerical experiments are representative of granular dynamics scenarios, in which a collection of rigid bodies has settled under their own weight within a boundary geometry.

4.7.1 Single-Step Convergence Experiments

The purpose of this section is to demonstrate the convergence properties of the developed iterative methods. Both frictionless and frictional tests will be considered. In this section, the appropriate methods will be used to solve the optimization problem associated with a single time step of a simulation. To this end, a simulation was run using the Chrono implementation. At the desired point in the simulation, the matrix $N$ and vector $r$ corresponding to the problem in Equation 2.46 for the current configuration were extracted and saved to files. Then, this data was loaded in MATLAB and the iterative methods were each applied in turn to solve the same problem. For more details, see Appendix A. Note that similar results for frictionless problems were reported in [70].

4.7.1.1 Frictionless Experiments

Here, we consider a collection of spheres contained in a cylinder with an inner radius of $1.2 \text{ m}$. The spheres have radius $0.1 \text{ m}$ and mass $6.28 \text{ kg}$, and are considered in the configurations shown in Figures 4.26a-4.26c. The problems in this subsection are frictionless, i.e., all contact interactions are characterized by $\mu = 0$. The iterative methods considered for this set of tests are JACOBI, SEIDEL, GPMINRES, APGD, and PD-IP, which are all capable of solving frictionless problems. The methods were applied for a fixed number of iterations, i.e., with no stopping criteria, so that a more complete picture of the convergence properties could be observed.

Figures 4.27-4.29 show the convergence behavior of the various iterative methods applied to problems with $1,000$ bodies/$4,192$ contacts, $2,000$ bodies/$8,622$ contacts, and $4,000$ bodies/$17,510$ contacts respectively. Each figure shows the evolution of both the objective function value and the residual value. Note that the objective function value is shifted by a constant, i.e., $f(x) = \frac{1}{2} x^T N x + x^T r + C$. The constant $C$ was selected for each test problem such that the ‘best’
Figure 4.26: System configurations for single-step convergence tests for frictionless problems.
method for each test reached a shifted objective function value of 0. Also, the residual plotted in these figures is the $r_4$ residual as described in Section 4.6.1.

These results demonstrate the relative performance of the various iterative methods on frictionless problems. APGD and GPMINRES perform significantly better than JACOBI and SEIDEL in terms of both the objective function and the residual. SEIDEL performs better than JACOBI, although the difference is smaller for larger problems. Clearly, PD-IP performs the best out of all considered methods in terms of iteration count, although each iteration is much more computationally intensive than those of the first-order methods. The performance of APGD and GPMINRES is very similar for the test case with 1,000 bodies, but APGD is noticeably better for the larger test case. Further, APGD seems to fare the best as the problem size increases. For example, the qualitative evolution of the objective function using APGD is quite similar for all three tests. In contrast, JACOBI and SEIDEL show a noticeably shallower slope as the problem size grows.

### 4.7.1.2 Frictional Experiments

Here, consider a collection of spheres contained in a cylinder with inner radius 1.2 m. The spheres have radius 0.1 m and mass 6.28 kg. The friction coefficient between spheres and between the cylinder and the spheres is 0.25. At the beginning of the time step of interest, the configuration of the spheres can be seen in Figures 4.30a-4.30b.

The iterative methods considered for this set of tests are JACOBI, SEIDEL, APGD, and PD-IP. The methods were applied for a fixed number of iterations.

Figures 4.31-4.32 show the convergence behavior of the various iterative methods applied to problems with 1,000 bodies and 2,000 bodies, respectively. Each figure shows the evolution of both the objective function value and the residual value.

These results show several important facts. First, APGD provides the best performance among the first-order methods. Second, the performance of PD-IP is very good in terms of iteration count, but each iteration is much more computationally expensive because it involves solving a linear system instead of performing matrix-vector multiplication.
Figure 4.27: Convergence of iterative solvers for single time step of simulation with 1,000 bodies and 4,192 contacts (frictionless).
Figure 4.28: Convergence of iterative solvers for single time step of simulation with 2,000 bodies and 8,622 contacts (frictionless).
Figure 4.29: Convergence of iterative solvers for single time step of simulation with 4,000 bodies and 17,510 contacts (frictionless).
Figure 4.30: System configurations for single-step convergence tests for frictional problems.
Figure 4.31: Convergence of iterative solvers for single time step of simulation with 1,000 bodies and 3,872 contacts (with friction).
Figure 4.32: Convergence of iterative solvers for single time step of simulation with 2,000 bodies and 7,931 contacts (with friction).
4.7.2 Simulation Performance Experiments

This section demonstrates the performance of various iterative methods over the course of actual dynamic simulations. These simulations rely on the Chrono [69] framework. Existing implementations of the Jacobi and Seidel solvers were used in addition to a new implementation of the APGD solver. The PD-IP solver was not used as it has not yet been implemented in the Chrono framework. All tests in this section were run with a serial implementation on a single CPU core.

While the form of the problem which must be solved at each time step is the same throughout a simulation, the dimension and difficulty may vary greatly. Therefore, two classes of problems are considered. In the first, a collection of spheres is released from rest in a settled configuration. In other words, the bodies should not move significantly, and all time steps are expected to be similar in terms of difficulty. In the second set, a collection of spheres are released from rest in a cylindrical container. The cylinder has a circular hole in the base, and the spherical bodies flow out of the cylinder into a lower cylindrical receptacle. In this situation, the configuration of the bodies is changing at each time step and the system is more dynamic than in the first set of simulations.

Test Set 1: Settled Spheres

Here, we consider a collection of spheres contained in a cylinder. The spheres have radius 0.1 m and mass 6.28 kg. The friction coefficient between all bodies is 0.25. The spheres are released from rest from the configurations seen in Figures 4.33a-4.33c.

The iterative methods considered for this set of tests are JACOBI, Seidel (SOR), and APGD. Each configuration of bodies was simulated for one second with each of the iterative methods using a time step of 0.01 seconds and a tolerance of $10^{-3}$ in the $r_4$ residual (see Section 4.6.1). At each time step, the data recorded included the number of iterations required to converge, the CPU time required to converge, and the CPU time required to solve the CCP problem (excludes collision detection and other data management).
Figure 4.33: Initial configurations for test simulations of settled spheres

(a) $N_b=1,000$, $N_c=3,752$

(b) $N_b=2,000$, $N_c=7,678$

(c) $N_b=4,000$, $N_c=15,653$
Table 4.1: Required time to perform 1 second long simulation of 1,000 bodies in Chrono

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Computation Time [seconds]</th>
<th>Overall Speedup of APGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>JACOBI</td>
<td>2,243.60</td>
<td>11.4282</td>
</tr>
<tr>
<td>SOR</td>
<td>927.55</td>
<td>4.7246</td>
</tr>
<tr>
<td>APGD</td>
<td>196.32</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.2: Required time to perform 1 second long simulation of 2,000 bodies in Chrono

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Computation Time [seconds]</th>
<th>Overall Speedup of APGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>JACOBI</td>
<td>9,311.4</td>
<td>14.7341</td>
</tr>
<tr>
<td>SOR</td>
<td>5,047.1</td>
<td>7.9864</td>
</tr>
<tr>
<td>APGD</td>
<td>631.96</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.34 shows the performance of APGD compared to JACOBI and SOR for the test with 1,000 bodies and 3,752 contacts. Table 4.1 gives the total computation time required to perform each simulation, and the overall speedup that APGD achieves compared to JACOBI and SOR.

Figure 4.35 shows the performance of APGD compared to JACOBI and SOR for the test with 2,000 bodies and 7,678 contacts. Table 4.2 gives the total computation time required to perform each simulation, and the overall speedup that APGD achieves compared to JACOBI and SOR.

Figure 4.36 shows the performance of APGD compared to JACOBI and SOR for the test with 4,000 bodies and 15,653 contacts. Table 4.3 gives the total computation time required to perform each simulation, and the overall speedup that APGD achieves compared to JACOBI and SOR.

Overall, several important trends can be seen from these tests. First, the speedup of the APGD method compared to both the JACOBI and SOR methods is higher for taller stack heights. Second, the performance advantage of SOR compared to JACOBI seems to decrease as the stack height grows. In fact, in the test with 4,000 bodies, JACOBI was nearly as fast as SOR, while APGD was about 9 times faster than both.

Next, a set of similar simulations were performed to understand the relative performance of the iterative methods for different levels of accuracy. For these tests, the initial configuration in Figure
Figure 4.34: Performance of various iterative methods for test with 1,000 spheres
Figure 4.35: Performance of various iterative methods for test with 2,000 spheres

(a) Iterations taken for convergence at each step

(b) CPU time for convergence at each step

(c) Speedup at each step
Figure 4.36: Performance of various iterative methods for test with 4,000 spheres

(a) Iterations taken for convergence at each step

(b) CPU time for convergence at each step

(c) Speedup at each step
Table 4.3: Required time to perform 1 second long simulation of 4,000 bodies in Chrono

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Computation Time [seconds]</th>
<th>Overall Speedup of APGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>JACOBI</td>
<td>26,592.3</td>
<td>9.3993</td>
</tr>
<tr>
<td>SOR</td>
<td>25,147.6</td>
<td>8.8887</td>
</tr>
<tr>
<td>APGD</td>
<td>2,829.2</td>
<td>1</td>
</tr>
</tbody>
</table>

4.33a was used. In each simulation, the dynamics of the system was simulated for one second with a time step of 0.01 seconds. The simulation was performed three times for each solver, using a residual-based stopping criteria tolerance of $10^{-2}$, $10^{-3}$, and $10^{-4}$. Figure 4.37 shows the speedup of APGD compared to JACOBI and SOR for each tolerance value.

Note that the performance of APGD relative to both JACOBI and SOR increases as the tolerance decreases. In other words, if one is interested in performing a simulation with higher accuracy, the APGD method becomes even more attractive.

**Test Set 2: Flowing Spheres**

Here, we consider a collection of spheres contained in a set of cylinders. The spheres have radius 0.1 m and mass 6.28 kg. The friction coefficient between all bodies is 0.25. The boundary geometry is composed of two cylinders of radius 1.2 m stacked on top of each other. The base of the lower cylinder is 4 m below the base of the upper cylinder. There is a circular hole of radius 0.6 m in the center of the base which separates the cylinders. A schematic of the boundary geometry is shown in Figure 4.38.

1000 spheres are released from rest in a loosely packed configuration in the upper cylinder and allowed to begin flowing into the lower cylinder. The simulation captures the dynamics of the system for five seconds with a time step of 0.01 seconds. Figure 4.39 shows the state of the system at $t = 0$, $t = 1$, $t = 2$, $t = 3$, $t = 4$, and $t = 5$ seconds.

The simulation was performed once each with the JACOBI, SEIDEL, and APGD solvers. The required computation times for each simulation and the net speed-up of APGD compared to JACOBI and SOR can be seen in Table 4.4.
Figure 4.37: Speedup of APGD compared to JACOBI and SOR for various tolerance values.

Table 4.4: Required time to perform 5 second long flow simulation of 1,000 bodies in Chrono

<table>
<thead>
<tr>
<th>Method</th>
<th>Total Computation Time [seconds]</th>
<th>Overall Speedup of APGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>JACOBI</td>
<td>49356.4</td>
<td>20.5372</td>
</tr>
<tr>
<td>SOR</td>
<td>11580.1</td>
<td>4.8185</td>
</tr>
<tr>
<td>APGD</td>
<td>2403.3</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 4.38: Schematic of boundary geometry for flowing spheres simulation test.
Figure 4.39: Snapshots from flowing sphere test problem using APGD solver
The plots in Figure 4.40 show the required number of iterations, the required CPU time, and the speedup for each time step of the simulation using each of the three considered solvers. Recall that the main difference between this test and that of Table 4.1 is that this test captures a highly dynamic system in which the contact configuration is changing at each time step while in the previous test the system configuration stayed nearly constant. The results in Table 4.4 and Figure 4.40 demonstrate that the relative performance of APGD compared to JACOBI and SOR is at least as good in the dynamics case as in the quasi-static case. In fact, the relative performance of APGD compared to SOR was about the same for the dynamic and quasi-static 1,000 body test problems, while the relative performance of APGD compared to JACOBI was almost two times better in the dynamic case than in the quasi-static case.

Next, the same scenario was used to test the effectiveness of the various iterative methods for different friction coefficients. To this end, simulations were run with $\mu = 0.0$, $\mu = 0.25$, $\mu = 0.50$, and $\mu = 0.75$ with each solver. The simulation set-up in terms of boundary conditions and initial conditions was the same as the previous test. For each value of the friction coefficient, the speed-up of the APGD method compared to the JACOBI and the SOR solvers was computed based on the total computation time for the simulation.

Figure 4.41 shows the required computation time for JACOBI, SOR, and APGD for each value of $\mu$. The results demonstrate that all methods solve the problem fastest when $\mu = 0$. The required computation time increases with increasing friction coefficient for JACOBI, while it seems to increase and then plateau for SOR and APGD. In fact, APGD takes slightly less time to solve the problem with $\mu = 0.75$ than it did with $\mu = 0.5$, while SOR takes about the same amount of time for those two cases. Figure 4.42 shows the speed-up achieved by APGD compared to the other two considered solvers for each value of $\mu$. The results demonstrate that the APGD method handles high friction values better than JACOBI and SOR. In fact, the speed-up increases with the friction coefficient compared to both other methods. The JACOBI method seems to degrade most quickly relative to the other solvers.
Figure 4.40: Performance of various iterative methods for flowing sphere test with 1,000 spheres

(a) Iterations taken for convergence at each step

(b) CPU time for convergence at each step

(c) Speedup at each step
Figure 4.41: Computation time for JACOBI, SOR, and APGD for various friction coefficients.
Figure 4.42: Speedup of APGD compared to JACOBI and SOR for various friction coefficients.
Chapter 5

Handling Large Many-Body Dynamics Problems within the Discrete Element Method Framework

This chapter describes the domain decomposition framework created to enable large many-body dynamics simulations on distributed computing architectures. Specifically, the framework is designed to divide a simulation domain into a number of sub-domains which can be simulated in parallel on discrete compute cores. The framework consists of a method for splitting the simulation domain into a number of sub-domains, setting up the necessary communication channels between compute cores, modeling constrained assemblies of rigid bodies, and carrying out the simulation including collision detection, dynamics computations, and synchronization. The remainder of the chapter describes these components in more detail.

5.1 Algorithmic Flow

The general algorithmic flow of the domain decomposition approach for DEM simulations is given in the following pseudocode. The pseudocode specifies the operations performed by a given compute core simulating a single sub-domain. All compute cores operate independently, in parallel, subject to some communication and synchronization barriers.

\begin{verbatim}
PSEUDOCODE DEM MPI()
(1) Identify boundaries of sub-domain \( S_i \)
(2) Initialize communication pathways to each sub-domain \( S_j \)
(3) Initialize data structures
(4) for \( t = 0 \) to \( t_{end} \)
\end{verbatim}
(5) Perform collision detection
(6) Compute collision forces
(7) Barrier: Intermediate communication of net forces on shared objects
(8) Compute net forces
(9) Solve dynamics problem
(10) Perform integration to update positions
(11) Barrier: Final communication/synchronization
(12) \( t \leftarrow t + h \)
(13) \text{ endfor}

These components will be explained in more detail in the following sections.

5.1.1 Domain Splitting

The domain decomposition is performed as a pre-processing step at the beginning of the simulation. The overall simulation domain is divided into a user-specified number of sub-domains by axis-aligned separating planes. The global Cartesian x-, y-, and z-axes are divided independently at specified locations, resulting in a grid of axis-aligned rectangular cuboid sub-domains. Separate MPI processes are mapped to each sub-domain. Note that the domain decomposition is static, and does not change during the simulation. Therefore, sub-domain boundaries should be carefully chosen so as to maintain good load-balancing during the course of the simulation.

For illustrative purposes, assume a simulation is divided into a set \( S \) of \( m \) sub-domains.

\[
S = \{S_1, S_2, \ldots, S_m\}
\]  

5.1.2 Data Structures and Communication Pathways

Two approaches for inter-domain communication have been developed. One approach is more general, allowing any sub-domain to communicate with any other sub-domain. This general approach allows arbitrary combinations of sub-domain and rigid body sizes, but results in more
communication overhead. In the second approach, sub-domains communicate only with their 26 immediate neighbors. This reduces the communication overhead, but can only be used when all objects span no farther than adjacent sub-domains.

The general, full-communication approach will be described in more detail. The specialized, limited-communication approach is simple to understand based on the full approach.

In general, communication occurs twice per time step: intermediate communication and final communication/synchronization. To facilitate this communication, each sub-domain maintains at all times \( m + 1 \) lists of shared objects. The first list contains all objects which are even partially contained in the associated sub-domain. Each of the next \( m \) lists contains the objects which are shared with each of the \( m \) sub-domains. Note that ‘object’ usually refers to a single rigid body in the simulation. However, it may also refer to a sub-collection of bodies connected with joints or constraints (called an assembly herein, and described later in this chapter). Each object in the simulation should have a unique ID number.

For example, sub-domain \( S_i \) maintains the lists \( L^0_i, L^1_i, L^2_i, \ldots, L^m_i \). List \( L^0_i \) is the list of all bodies which touch sub-domain \( S_i \), while list \( L^j_i \) is the list of all bodies which touch both sub-domains \( S_i \) and \( S_j \). Due to the discrete nature of the targeted hardware infrastructure, some redundant data storage is necessary. For example, sub-domain \( S_i \) maintains a list \( L^j_i \) of bodies it shares with sub-domain \( S_j \). Similarly, sub-domain \( S_j \) maintains an identical list \( L^j_i \) of bodies it shares with sub-domain \( S_i \). All lists are maintained in order sorted by object ID number.

### 5.1.3 Collision Detection

Collision detection for sub-domain \( S_i \) is performed among all objects which appear in list \( L^0_i \). Given the states of all objects in \( L^0_i \), the collision detection step should compute and return a list of collision pairs, \( C_i \), where entry \( k \) is composed of the ID numbers of the colliding objects. In other words, \( C^k_i \) has two entries, specified as \( C^k_{i,A} \) and \( C^k_{i,B} \).

Two different implementations have been used to this end. The first is based on the open-source Bullet Physics Library [71]. Bullet provides collision detection routines targeted at use in video games. Therefore, Bullet provides a number of primitive collision geometries such as spheres,
cylinders, and boxes. However, Bullet has some significant drawbacks. For instance, Bullet is
tuned to handle objects of unit size, systems where objects are of roughly the same dimensions,
and bodies which are geometric primitives or can be constructed from a small number of primitives.
This is often not the case in systems targeted by this thesis, where there are often a large number of
very small bodies and a collection of a few complex, large objects. Therefore, a second collision
detection routine has been developed and implemented for use when the limitations of Bullet result
in decreased performance.

The new collision detection routine is based on a binning approach as described in [44], which
is based on an extension of the method described in [72]. In this approach, the collision detection
domain is divided into a collection of bins. The bins can be processed independently, performing
a brute-force search for collisions among all the bodies in a given bin. When the number of bodies
per bin is small, the brute force search can be performed quickly. In [44], this parallel, scalable
approach for collision detection was implemented to leverage the parallel computing power of the
Graphics Processing Unit (GPU). In this work, the method has been implemented to run on the
CPU. The motivation for this choice was that on the targeted hardware, each sub-domain/CPU-
core pair may not have access to a GPU on which to perform collision detection. A summary of
the collision detection method is provided next.

In a pre-processing step, the spatial subdivision is carried out with a constant bin size, dividing
the simulation sub-domain into bins. Next, a loop over all the objects in the sub-domain determines
and stores the IDs of the bins with which the given object intersects. A radix sort algorithm [73] is
used to group the data by bin. Next, the start of each bin’s list of data in the sorted list is determined.
Finally, a loop over each bin is used to compare all bodies in the bin and determine contacts. Note
that this approach is the same as that described in [44], except that parallel kernels executed on the
GPU are replaced with loops executed on the CPU. While the absolute performance of the method
is not as good without the acceleration of the GPU, the method still scales linearly with the number
of objects and is very scalable.

The method described here could be used for bodies of any shape. In practice, however, it can
be quite difficult to determine contacts between complex geometries. Therefore, this method is
Figure 5.1: Original body geometry (left), sphere-set approximation (right).

primarily of use for determining contacts between primitives. In [44], the method is implemented and tested using spheres, while [74] extends the method to handle ellipsoids. To handle more complex geometries, a method called spherical decomposition was introduced in [13], where the collision geometry of an object was approximated as a union of spheres for the collision detection step of a dynamic simulation. Figure 5.1 shows an example of the exact body geometry as defined in a CAD package and a detailed view of the sphere-set used for collision detection.

Approximate collision geometry is created from the original geometry through a three step process. First, the surface of the original geometry is meshed with triangular elements. Next, a single sphere is fit through the three vertices of each triangle. Finally, the sphere-set is checked for accuracy and refinement is performed. An overview of the sphere-fitting method is provided next.

For each triangle of the surface mesh, the center, $C$, of the corresponding circumcircle is found. Additionally, the outward unit surface normal, $N$, of the triangle is computed. Now, a sphere can be placed at $S = C + tN$, where $t$ is a parameter which can take any value and the radius of the sphere, $R$, is such that the sphere passes through the three points of the triangle. Note that $t = 0$ corresponds to the sphere of minimum radius and $t < 0$ sinks the sphere center into the body while increasing its radius. In general, it is better to place spheres below the surface because this better approximates the plane of the triangle (see Figure 5.2). In practice, a dimensionless quantity called
the center ratio, defined as the distance from the sphere center to the triangle plane divided by the radius of the sphere, was considered as constant for all spheres in a given object. In practice, a center ratio of 0.7 gave good results. A detailed account of the implementation is provided in [13].

5.1.4 Force Computation

At this stage, each sub-domain $S_i$ has a list $C_i$ of collisions among bodies in list $L_i^0$. The purpose of this step is to compute the force contribution to each body in $L_i^0$ due to all effects in $S_i$. Possible forces include body forces, such as gravitational force, applied forces, or contact forces.

Gravitational forces are applied to each body $k$ in $L_i^0$ only if its center of mass, $x_k$, is in $S_i$ (see Equation 5.2). This prevents multiple sub-domains from computing and adding the gravitational force.

$$ f_i^{g,k} = \begin{cases} m_k g & : x_k \in S_i, \forall k \in L_i^0 \\ 0 & : x_k \notin S_i \end{cases} $$ (5.2)

For the same reason, contact forces are computed for each contact $l$ in $C_i$ only if the center of the contact, $c_l$, is in $S_i$. The contact force for collision between bodies $C_i^{l,A}$ and $C_i^{l,B}$ is computed based on the selected DEM force model as $f \left( C_i^{l,A}, C_i^{l,B} \right)$, where the force may be a function of the states of the bodies, the duration of the contact, and a number of constants or material properties. Note that the computed force should be applied to body $C_i^{l,A}$, and its negative should be applied to body $C_i^{l,B}$.
The net force due to sub-domain $S_i$ acting on body $k$, $F_{i}^{n,k}$, is computed as

$$F_{i}^{n,k} = F_{i}^{a,k} + \sum_{l \in A_k} F_{i}^{c,l} - \sum_{l \in B_k} F_{i}^{c,l}, \forall k \in L_i^0,$$

(5.4)

where $F_{i}^{a,k}$ is the applied force, $A_k = \{l | C_{i}^{l,A} = k\}$ and $B_k = \{l | C_{i}^{l,B} = k\}$.

### 5.1.5 Intermediate Communication

The purpose of the intermediate communication is to obtain the true net force acting on each body in the simulation domain. Specifically, each sub-domain which shares a given body must exchange with all other sub-domains which share that body the associated force computed therein. Note that sub-domain $S_i$ now has a list of forces, $F_i^n$, where $F_{i}^{n,k}$ is the force component acting on body $k$ in $L_i^0$.

Each sub-domain $i$ should now send to each other sub-domain $j$ the force which acts on each body in $L_i^j$, as computed in $S_i$ (see Equation 5.5). Similarly, each sub-domain $i$ should also receive from each other sub-domain $j$ the force component which acts on each body in $L_i^j$, as computed in $S_j$ (see Equation 5.6).

Send $F_{i}^{n,k}$ to $S_j$, $\forall k \in L_i^j$  

(5.5)

Receive $F_{j}^{n,k}$ from $S_j$, $\forall k \in L_i^j$  

(5.6)

Finally, the true net force on each body in $L_i^0$ can be computed by summing the force components from each sub-domain (see Equation 5.7).

$$F_{i}^{k} = \sum_{j=1}^{m} F_{j}^{n,k}, \forall k \in L_i^0$$

(5.7)
These forces represent the contributions of all sub-domains, resulting in the true net force on each body for the full simulation domain. Care was taken to add each force component only once, no matter how many sub-domains a given body or interaction was present in. Finally, note that each sub-domain which shares a given body end up with the same net force acting on that body. In other words, if body $k$ is shared by sub-domains $i$ and $j$, then $F^k_i = F^k_j$.

5.1.6 Dynamics and Numerical Integration

For dynamics and numerical integration, each sub-domain can be handled independently because $S_i$ has all the state and force information needed to advance the states of its bodies by one time step. If the system consists of only bodies and contacts, i.e., if there are no bilateral constraints, then the solution of the dynamics problem is trivial, and the new state of all bodies in the sub-domain can be computed directly.

$$v^{(l+1)}_k = v^{(l)}_k + \frac{h F^k_i}{m_k}, \forall k \in L^0_i \quad (5.8)$$

$$x^{(l+1)}_k = x^{(l)}_k + h v^{(l+1)}_k, \forall k \in L^0_i \quad (5.9)$$

In the preceding, $x^{(l)}_k$ and $v^{(l)}_k$ are respectively the position and velocity of body $k$ at time $t^{(l)}$, while $x^{(l+1)}_k$ and $v^{(l+1)}_k$ are the position and velocity of body $k$ at time $t^{(l+1)} = t^{(l)} + h$, and $m_k$ is the mass of body $k$.

If the sub-domain has a collection of bodies with bilateral constraints, called herein an assembly, the solution progresses differently. In this case, the problem is treated as in the DVI formulation (see Section 2.1) and solved with an iterative method (see Chapter 4). Note that the contacts have already been accounted for in the DEM force computation, so the unilateral constraints are not included in the iterative DVI solution method.

It is important to note that an assembly is always treated as a unit in this domain-decomposition framework. In other words, the entire assembly is considered as shared if any of its bodies touch different sub-domains. Therefore, if any body in the assembly would show up in $L^j_i$, then all bodies involved in the assembly would show up in that list.
5.1.7 Final Communication and Synchronization

The purpose of the final communication is to synchronize the sub-domains, clean up the data structures, and prepare for the next time step. Synchronization is performed to avoid drift due to round-off error. Note that if a body is shared between sub-domains $i$ and $j$, both sub-domains should compute exactly the same new state for that body. However, due to differences in the order of computation, the two sub-domains may compute a slightly different value. Therefore, a synchronization step is used to ensure consistency among sub-domains. To this end, the ‘master’ sub-domain of a given body (the one which contains the center of mass) sends to all other sharing sub-domains the ‘true’ state of the body which will be used moving forward.

Next, the data structures, lists $L_i^0$ through $L_i^m$, should be processed and updated for the next time step. Specifically, lists $L_i^1, \ldots, L_i^m$ should be cleared. Then, sub-domain $i$ should loop over all bodies in $L_i^0$ and check the bodies in relation to the boundaries of $S_i$. If body $k$ is contained completely within sub-domain $S_i$, no action should be taken. If body $k$ is completely outside of $S_i$, then it should be streamed to the appropriate sub-domain and removed from $S_i$. If body $k$ is neither completely inside nor completely outside of $S_i$, then it must be shared with one or more sub-domains and should be streamed to the appropriate sub-domains where it is added to the appropriate shared body lists. Note that when a body is streamed to another sub-domain, all of its representative and state data must be sent. This includes the mass, inertia, position, orientation, velocity, acceleration, and collision geometry.

At this point, simulation can advance to the next time step.

5.2 Numerical Experiments

This section describes some sample simulations using the domain decomposition framework. Specifically, a scaling analysis is performed to demonstrate the strong scaling attribute of the framework. The scaling analysis also serves as a demonstration of the importance of load-balancing.
5.2.1 Scaling Analysis

The simulation used here consists of a collection of 10,000 spherical bodies interacting with each other and a complex boundary geometry made of boxes. The simulation used a time step of $10^{-5}$ seconds and represents 40 seconds of dynamics. The same simulation was performed with 1, 2, 4, and 8 sub-domains. With 2 sub-domains, the domain was divided by a vertical plane. With 4 sub-domains, the domain was divided by two orthogonal vertical planes (see Figure 5.3). With 8 sub-domains, the domain was divided by three mutually orthogonal planes.

Figure 5.3 shows the state of the system 16 seconds into the simulation when using 4 sub-domains. The bodies are colored by the sub-domain they are in, while shared bodies which span sub-domain boundaries are colored white. In this test, bodies are added to the upper box until 10,000 bodies are contained therein. At 8 seconds into the simulation, two square holes open up in opposite corners of the upper box, and the spheres begin to flow out. The spheres interact with a series of ramps as they move into the lower box. At the end of the simulation, at $t = 40$ seconds, nearly all of the spheres have flowed into the lower box.
Table 5.1: Simulation time and speed-up for scaling analysis.

<table>
<thead>
<tr>
<th>Number of Sub-Domains</th>
<th>Computation Time [hrs]</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>136.12</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>75.00</td>
<td>1.815</td>
</tr>
<tr>
<td>4</td>
<td>43.60</td>
<td>3.122</td>
</tr>
<tr>
<td>8</td>
<td>40.15</td>
<td>3.390</td>
</tr>
</tbody>
</table>

Table 5.1 gives the computation time and the speed-up compared to the single sub-domain case, while the speed-up is plotted in Figure 5.4.

Note that this framework shows good strong scaling as long as the computational load is evenly balanced between the sub-domains. This is the case when using two and four sub-domains as the dividing planes result in symmetric sub-domains. Perfect strong scaling would result in a speed up factor equal to the number of sub-domains (the same as the number of cores). The framework and implementation described here cannot, of course, achieve that performance because of overhead and synchronization requirements.

The effect of poor load balancing can be seen in the case of eight sub-domains. In this case, the simulation is not symmetric with respect to the horizontal dividing plane. In fact, all of the bodies start above the dividing plane, meaning that the lower sub-domains and their associated CPU cores have no work to perform at the beginning of the simulation. Similarly, nearly all the bodies end up below the dividing plane, meaning that there is a severe imbalance between the required work done by the upper and lower sub-domains. From the performance data, it is clear that using four more cores in this configuration results in very little improvement in performance compared to the four sub-domain case. Due to the static nature of the domain decomposition, it is therefore very important to consider the expected dynamics of the system when specifying sub-domain boundaries.
Figure 5.4: Speed-up of simulation with increasing number of sub-domains.
Chapter 6

Implementation Aspects that Support Parallel Computing

This chapter describes details related to the implementation of the rigid many-body dynamics methods described in previous chapters. The implementation is built within the physics engine called Chrono [69]. First, an overview of the Chrono framework will be described, including the support for both the DVI and DEM formulations for handling contact and friction. Second, the use of parallel computing in Chrono will be described. Details related to the use of GPU computing, OpenMP parallelism, and MPI parallelism will be provided.

6.1 Overview of Chrono

Chrono is a high-performance, parallel-computing multi-physics engine developed to solve rigid and flexible body dynamics, fluid dynamics, and fluid-solid interaction problems. Chrono consists of a collection of vertical applications built on top of a common core layer, called Chrono::Engine, which provides the commonly used subroutines which all computational dynamics applications require. To this end, Chrono::Engine provides routines for modeling, numerical methods, proximity computation and collision detection, domain decomposition, and pre/post-processing. The vertical applications currently include: Chrono::Rigid, for the simulation of systems of rigid bodies interacting through friction, contact, and bilateral constraints; Chrono::Flex, for systems of flexible bodies with contact and constraints; Chrono::Fluid, for fluid dynamics and fluid-solid interaction problems via the smoothed particle hydrodynamics method; Chrono::Terrain, a deformable terrain database system; Chrono::Mobility, for the simulation of a multibody vehicle model interacting
with deformable terrain; and Chrono::Render, a high-quality rendering pipeline for simulation data.

The focus of this chapter is on Chrono::Rigid, which implements the rigid body dynamics and approaches described in this thesis. Chrono::Rigid was originally written leveraging the DVI formulation for handling friction and contact. It is an object-oriented code written in C++. As such, Chrono::Rigid relies heavily on a hierarchical class structure using inheritance. To create a simulation in Chrono::Rigid, one instantiates an object of the ChSystem class. This object represents and contains the entire dynamic system. Then, the system can be populated by adding objects of type ChObject. Note that inheritance is used heavily. For example, the class ChBody represents a rigid body and extends the ChObject class.

Once a collection of rigid bodies and joints are added to the system, the Chrono::Rigid module performs the dynamic simulation. The user can control the parameters of the simulation, such as the time step, tolerance, and any other parameter used in the solver or collision detection. The user may also select from among a set of provided solvers. Chrono::Rigid includes iterative solvers based on the Jacobi and Gauss-Seidel methods, and has recently been extended with a solver based on the Accelerated Projected Gradient Descent method [14, 51, 50, 49, 75, 44].

### 6.2 Computational Flow

Simulations in Chrono::Rigid are written as calls to library functions. The user writes a main function which carries out the simulation. First, the user instantiates the system by calling the constructor, ChSystem mySys = new ChSystem();. The user can then create the appropriate physics objects, setting the properties (mass, inertia, collision geometry) and initial conditions (position, velocity, acceleration) for each. The user may also specify joints or constraints between physics objects. Then, each entity should be added to the system container. The user may also set some properties of the simulation itself, such as the max number of iterations to perform at each time step and the tolerance for the solver.

To perform the time-stepping, the user should write a loop which calls the time-stepping function repeatedly. For example, mySys->DoStepDynamics(0.01); advances the state of the system
by 0.01 s. Note that collision detection and solution of the dynamics problem are handled internally and no user code is required. The user may modify, add, or remove the physics objects during the course of the simulation.

The listing below provides a pseudo-code overview of a simulation in the Chrono::Rigid framework. Note that some of the steps in the following are implicitly called and do not need to be called by the user, but they are included here for completeness. Further, notations have been provided regarding the potential for parallelism at appropriate steps. For example, the steps marked ‘parallel-per-body’ are parallel with one thread per body.

Two steps deserve special note. First, the collision detection step can be performed in parallel via an implementation of a spatial subdivision approach. Details of this method are available in [76]. Second, the solution of the CCP can leverage parallelism through a special implementation. This will be described in more detail in the following section.

**Computational Flow: Chrono::Rigid()**

1. Create ChSystem object
2. Create ChBody objects
3. Set parameters and initial conditions
4. while $t < t_{end}$
5. Perform collision detection (parallel)
6. Compute collision terms (parallel-per-contact)
7. Compute body forces (parallel-per-body)
8. Solve CCP problem (parallel)
9. Perform integration to update positions (parallel-per-body)
10. $t \leftarrow t + h$
11. endwhile

### 6.3 Parallel Computing in Chrono::Rigid

Chrono::Rigid has been developed to leverage parallel computing to accelerate many-body dynamics simulations via three different parallel architectures. First, the DVI formulation for
contact and friction has been implemented to support both GPU computing and CPU parallelism through CUDA and OpenMP, respectively. Second, the DEM formulation can use MPI parallelism in conjunction with the domain decomposition approach described in Chapter 5.

As described in Chapter 3, parallel computing is often needed to take advantage of the capabilities of new computing hardware. Using parallel computing enables simulations to be performed more quickly, and may allow longer or larger simulations to be carried out, increasing the realism and usefulness of simulation results. This section describes in more detail the use of parallel computing to accelerate rigid body simulations in the Chrono::Rigid module. The use of GPU and OpenMP parallelism in the DVI formulation will be described, followed by the use of MPI computing in the DEM formulation.

### 6.3.1 GPU and CPU Parallel Computing in DVI Approach

The Differential Variational Inequality (DVI) formulation for handling friction and contact is supported in the Chrono::Rigid framework through both serial and parallel implementations. The parallel implementation has been written to support GPU computing via CUDA and CPU parallelism via OpenMP while using as much common code as possible. To this end, the Thrust library [42], described in Section 3.2.2, was used in the implementation. The code was written to easily switch between CPU and GPU execution at compile-time, using a compiler flag. If the line `#define SIM_ENABLE_GPU_MODE` is included before compilation, the code will use GPU computing when possible. If the line is not included, the code will execute on the CPU, using OpenMP parallelism if possible.

#### 6.3.1.1 Data Structures

The data structures used in the Chrono::Rigid implementation must be different if GPU or CPU computing is used. If the flag for GPU computing is set, all applicable data structures are defined as `thrust::device_vector<>` objects. If CPU execution is desired, the data structures are instead defined as `thrust::host_vector<>` objects. This behavior can be seen in the following code listing. Note that all variables of vector type are defined as `custom_vector<>`. Therefore, if the
line `#define SIM_ENABLE_GPU_MODE` is included, device vectors will be used. Otherwise, host vectors will be used for all vector-type variables.

### Listing 6.1: Selection of GPU or CPU Data Structures

```c
#ifdef SIM_ENABLE_GPU_MODE
#defineTHRUST_DEVICE_SYSTEMTHRUST_DEVICE_SYSTEM_CUDA
#definecustom_vectorthrust::device_vector
#else
#defineTHRUST_DEVICE_SYSTEMTHRUST_DEVICE_SYSTEM_OMP
#definecustom_vectorthrust::host_vector
#endif
```

#### 6.3.1.2 Functions and Kernels

As with data structures, different functions and syntax are needed depending on whether GPU or CPU computing is used. The same flag can be used to select which code is executed as was used to select data structures. As an example, consider the code in the following listings. The purpose of this code is to increment the speed of each body given the force acting on each body. If GPU computing is used, this should be performed with a kernel with one thread per body in the system. If CPU computing is used, this same functionality would be achieved by looping over all bodies in the system; OpenMP could also be used through the inclusion of `#pragma omp parallel for`.

First, a function is written to perform the desired operation. In this case, the function should increment the velocity of a given body based on the mass, inertia, force, torque, and time step. Note that this function is defined with `__host__ __device__`, which means it is compiled to be executed on both the CPU (host) and GPU (device).

### Listing 6.2: Function to Increment the Velocity of a Body

```c
__host__ __device__ voidfunction_addForces(uint& index,bool*active,
real*inv_mass,real3*inv_inertia,real3*forces,real3*torques,
real3*vel,real3*omega) {
    if (active[index] != 0) {
        // v += m_inv * h * f
```
vel[index] += forces[index] * inv_mass[index];
// w += J_inv * h * c
omega[index] += torques[index] * inv_inertia[index];
}
}

The above function should be called in different ways if CPU or GPU is used. If the GPU is used, a GPU kernel should call the function. Recall that a GPU kernel is the code which is executed by each GPU thread. The GPU kernel for this case is shown in the listing below. Note the inclusion of __global__, which signifies a kernel function, and the macro INIT_CHECK_THREAD_BOUNDED, which ensures that the thread IDs executing the kernel are appropriate.

Listing 6.3: GPU Kernel to Increment the Velocity of All Bodies in the System

```c
__global__ void device_addForces ( bool * active , real * mass , real3 * inertia , real3 * forces , real3 * torques , real3 * vel , real3 * omega ) {
    INIT_CHECK_THREAD_BOUNDED ( INDEX1D , number_of_objects_const );
    function_addForces ( index , active , mass , inertia , forces , torques ,
                        vel , omega );
}
```

If CPU computing is used, the code should loop over all bodies in the system using an OpenMP for-loop, calling the function on each body as in the following listing.

Listing 6.4: Host Loop to Increment the Velocity of All Bodies in the System

```c
void ChLcpSolverGPU::host_addForces ( bool * active , real * mass , real3 * inertia , real3 * forces , real3 * torques , real3 * vel , real3 * omega ) {
    #pragma omp parallel for
    for ( uint index = 0; index < number_of_objects; index++ ) {
        function_addForces ( index , active , mass , inertia , forces ,
                            torques , vel , omega );
    }
}
```
Finally, the same flag as before is used to call the appropriate version. If the flag is defined, the GPU kernel is called. If the flag is not defined, the host function is called.

Listing 6.5: Chrono Call to Increment Velocities, Selecting Between CPU and GPU Execution

```c
#ifdef SIM_ENABLE_GPU_MODE
    COPY_TO_CONST_MEM(number_of_objects);
    device_addForces <<<BLOCKS(number_of_objects), THREADS>>>(
        CASTB1(gpu_data.device_active_data),
        CASTR1(gpu_data.device_mass_data),
        CASTR3(gpu_data.device_inr_data),
        CASTR3(gpu_data.device_frc_data),
        CASTR3(gpu_data.device_trq_data),
        CASTR3(gpu_data.device_vel_data),
        CASTR3(gpu_data.device_omg_data));
#else
    host_addForces(
        gpu_data.device_active_data.data(),
        gpu_data.device_mass_data.data(),
        gpu_data.device_inr_data.data(),
        gpu_data.device_frc_data.data(),
        gpu_data.device_trq_data.data(),
        gpu_data.device_vel_data.data(),
        gpu_data.device_omg_data.data());
#endif
```

This same structure is used for all parallel code sections which require custom code. A single function is written to perform the necessary operations, capable of executing on the CPU and GPU. One wrapper function is written for the GPU, where a kernel calls the function with each thread. Another wrapper function is written for the CPU, where the function is called from within an OpenMP for-loop. Finally, the flag is used to call the appropriate wrapper function in the Chrono::Rigid code.

Note that primitive functions are handled by Thrust directly. For example, sorting a Thrust vector can be accomplished by calling `thrust::sort(myVec.begin(), myVec.end());`. If
myVec is a host vector, Thrust will use OpenMP to sort it. If myVec is a device vector, Thrust will use the GPU.

Perhaps the most important computation in the Chrono::Rigid implementation is the function which evaluates the product $N\gamma$. This operation is used repeatedly in the various iterative methods. It is challenging because it represents a matrix-vector multiplication, which can scale like $O\left(N_{c}^{2}\right)$ in the general case. In the case of the problem of interest, the sparsity of $N$ leads to a matrix-multiplication which should scale like $O\left(N_{c}\right)$. In fact, this sparse matrix-vector multiplication is still the main computational bottleneck in the implementation. However, another challenge emerges due to the fact that the Chrono::Rigid implementation is matrix-free and never explicitly forms the matrix $N$.

Therefore, the product $N\gamma$ can be computed by a two-stage process. First, compute the term

$$x = M^{-1}D\gamma.$$  \hfill (6.1)

Due to the matrix free approach, this computation must be performed as a loop. Recall that the matrix $D = [D_1, \ldots, D_{N_c}]$, and $D_i = [D_{i,n}, D_{i,u}, D_{i,w}] \in \mathbb{R}^{6n_b \times 3}$ (see Equation 2.16). Note that each $D_i$ has two $(6 \times 3)$ blocks of non-zeros, corresponding to the two bodies which participate in contact $i$. In the implementation, the contacts are stored in a vector, and each contact stores its associated data. Therefore, each contact $i$ stores $D_{i,A}$, the 18 non-zero entries for $D_i$ associated with body $A$, and $D_{i,B}$, the 18 non-zero entries for $D_i$ associated with body $B$. Finally, the first stage can be written as

$$\text{for } i = 1 \text{ to } N_c$$

$$x_A = x_A + M^{-1}_AD_{i,A}\gamma_A$$ \hfill (6.2)

$$x_B = x_B + M^{-1}_BD_{i,B}\gamma_B$$ \hfill (6.3)

$$\text{end.}$$

In the preceding, note that the vector $x$ should be initialized to all zeros before the loop. Also, note that the iterations of the loop are not completely independent. In fact, race conditions may
occur because multiple contacts may involve the same body. For example, if contact 1 is between bodies 3 and 7 and contact 2 is between bodies 3 and 11, the iterations for $i = 1$ and $i = 2$ may yield a race condition. To resolve this issue, one of two different approaches is employed depending on whether the CPU or GPU is used. In the CPU case, the loop is performed serially, without OpenMP acceleration. Without parallelism, the race condition is not applicable. In the GPU case, the loop iterations are launched as parallel threads on the GPU. In this case, the additions in Equations 6.2 and 6.3 are done with atomic operators which allow only one thread to update a variable at a time. There is some performance penalty for this, but it ensures accuracy and deterministic behavior.

Finally, the second stage is used to compute the final result as

$$ y = D^T x. \quad \text{(6.4)} $$

Once again, this computation must be performed in a loop due to the matrix-free approach. The loop-form of the computation can be written as:

$$ \text{for } i = 1 \text{ to } N_c $$

$$ y_i = D_{i,A}^T x_A + D_{i,B}^T x_B \quad \text{(6.5)} $$

$$ \text{end} $$

In the second stage, the iterations of the loop are completely independent. Therefore, there are no possible race conditions in the parallelization of this loop. In the CPU case, OpenMP can be used to parallelize the loop using \texttt{omp parallel for}. In the GPU case, each iteration of the loop can be launched as an independent parallel thread.

### 6.3.2 MPI Support for DEM Approach

The DEM approach is also supported in the \texttt{Chrono::Rigid} framework in both serial and parallel implementations. The serial implementation is summarized by the following pseudo-code:

\texttt{PSEUDOCODE DEM()}

\begin{enumerate}
\item Create \texttt{ChSystem} object
\end{enumerate}
(2) Create ChBodyDEM objects
(3) Create ChContactContainerDEM object
(4) Set parameters and initial conditions
(5) \textbf{while} \ t < t_{\text{end}}
(6) \quad \text{Perform collision detection}
(7) \quad \text{Compute collision forces}
(8) \quad \text{Compute body forces}
(9) \quad \text{Solve CCP problem (if system has constraints)}
(10) \quad \text{Perform integration to update positions}
(11) \quad \ t \leftarrow t + h
(12) \textbf{endwhile}

The main differences between the DEM implementation and the DVI implementation is the use of an alternative contact container and body objects. Bodies of type ChBodyDEM store their DEM parameter values in addition to their rigid-body data. The ChContactContainerDEM object handles the contacts differently by computing the associated DEM forces and applying them to the correct bodies instead of computing the contact terms used in the DVI approach.

The parallel implementation is based on the domain decomposition approach described in Chapter 5. The Chrono framework is leveraged by using several specialized classes. For example, each MPI process creates and manages an object of type ChSystemMPI to represent the sub-domain, which extends the base class ChSystem. This object stores the usual system data and additional information about the extents of the sub-domain and which sub-domains are neighbors. Similarly, bodies are represented by objects of type ChBodyMPI, which extends the base class ChBody and provides functionality to determine and keep track of the list of sub-domains which share the given body.

Each MPI process manages the simulation for its associated sub-domain. See Section 5.1 for the pseudocode describing the operations that each sub-domain executes. In the time-stepping loop, each MPI process makes a call like \texttt{mySys->DoStepDynamics(h)} to advance the state of the
sub-domain by the time step $h$. During that call, each MPI process can advance independently to the first barrier, where intermediate communication occurs. Once the intermediate communication is finished, the MPI processes can again advance independently to the second barrier, where final communication occurs.

In terms of communication, non-blocking MPI sends and receives [46] are used. For example, refer to Equations 5.5 and 5.6, which represent the intermediate communication. In particular, the implementation uses `MPI_Isend` and `MPI_Irecv` for non-blocking sends. In practice, MPI process $i$ loops over all other processes, posting an `MPI_Isend` to each to send the forces in the appropriate lists. Note that each `MPI_Isend` returns immediately as they are non-blocking. Then, process $i$ again loops over all other processes, posting an `MPI_Irecv` to each to receive the forces in the appropriate lists. Again, these calls return immediately as they are non-blocking. Finally, process $i$ calls `MPI_Wait` on each communication path to ensure that all sending and receiving is complete before the net force can be computed (see Equation 5.7).
Chapter 7

Validation

The purpose of this chapter is to describe the outcome of the validation effort related to the numerical methods introduced in previous sections. Two types of validation will be presented. First, simulations will be validated against analytical solutions for simple scenarios such as a block sliding on an incline, a sphere on an incline which transitions from sliding to pure rolling, and a stack of spheres. Second, simulations will be validated against experimental data for more complex scenarios. In the first experimental validation, the mass flow rate of granular material flowing out of a container will be measured and compared to a corresponding simulation. The second experimental validation will study the penetration of a large spherical body as it impacts a bed of granular material.

7.1 Analytical Validation

This section provides a series of numerical experiments to validate the developed numerical solution methods against analytical results for simple systems. Specifically, the experiments include a block sliding down an incline, a sphere sliding down an incline which transitions to pure rolling, and a static stack of spheres.

7.1.1 Block on Incline

In this test, a block slides from rest down an inclined plane. The block is a cube, with side length $l$, mass $m$, and friction coefficient $\mu$. The plane is inclined at $\theta$ to the horizontal.

The equation of motion for the block in the translational direction can be written as
Figure 7.1: The block is released from rest on an inclined plane.
\[ m \ddot{x}_t = mg \sin(\theta) - f_f, \quad (7.1) \]

where \( x_t \) is the translational displacement down the slope and \( f_f \) is the frictional force.

Note that Equation 7.1 indicates that the block will slide down the slope when \( f_f < mg \sin(\theta) \). Additionally, the Coulomb friction model requires that \( f_f \leq \mu mg \cos(\theta) \). These conditions can be combined to determine the critical friction coefficient, \( \mu_c \), for a given slope angle.

\[
mg \sin(\theta) = \mu mg \cos(\theta) \quad (7.2)
\]

\[
\implies \mu_c = \tan(\theta) \quad (7.3)
\]

For a given slope angle \( \theta \), the block will slide if \( \mu < \mu_c \), with \( f_f = \mu mg \cos(\theta) \). The block will stick in place if \( \mu \geq \mu_c \), with \( f_f = mg \sin(\theta) \).

The velocity and position for the block are given by the following expressions, assuming the block starts from rest at the origin.

\[
\dot{x}_t = (mg \sin(\theta) - f_f) t \quad (7.4)
\]
\[
x_t = 0.5 (mg \sin(\theta) - f_f) t^2 \quad (7.5)
\]

Two simulations have been performed to compare simulation results with the analytical solution. In these tests, the slope angle was \( \theta = 20^\circ \), resulting in a critical friction coefficient of approximately \( \mu_c = 0.36397 \). The simulations were performed with the iterative Seidel solver and the APGD solver, with time step \( h = 0.001 \) s, tolerance \( 10^{-6} \), and maximum of 100 iterations per time step.

In the first simulation, the friction coefficient was set below the critical value at \( \mu = 0.36 \), meaning that the block slid down the slope. Figures 7.2, 7.3, and 7.4 show the position and velocity results in the \( x-, y-, \) and \( z- \) directions respectively.

In Figures 7.2-7.4, it can be observed that the APGD results are much closer to the analytical results than are the Seidel results. There is significant drift in both position and velocity when using the iterative Seidel method.
Figure 7.2: X-direction position and velocity results for block sliding on incline ($\mu = 0.36 < \mu_c$).
Figure 7.3: Y-direction position and velocity results for block sliding on incline ($\mu = 0.36 < \mu_c$).
Figure 7.4: Z-direction position and velocity results for block sliding on incline ($\mu = 0.36 < \mu_c$).
The same scenario was simulated a second time, with the friction coefficient set to $\mu = 0.37$, which is above the critical value, indicating that the block should remain motionless in stiction. Figures 7.5, 7.6, and 7.7 show the position and velocity results in the $x$-, $y$-, and $z$-directions respectively for the sticking case.

In Figure 7.5-7.7, it can be observed that the APGD results are much closer to the analytical results than are the Seidel results. Note that both solvers allow some small amount of drift down the slope, with the iterative Siedel method allowing larger drift in both position and velocity than APGD.

In this test, both methods performed fairly well in the stiction case, with errors on the order of $10^{-3}$ in position and velocity. In the sliding case, when the friction coefficient was just below the critical value, the APGD solver was significantly more accurate than the Seidel solver for a given number of iterations.

Next, the same scenario was used to analyze the accuracy of the methods as the time step changed. To this end, five values were used for the time step, 0.01 s, 0.005 s, 0.0025 s, 0.001 25 s, and 0.000 625 s. For each time step value, simulations were performed with both the APGD and Seidel solvers and for both friction coefficients of $\mu = 0.36$ and $\mu = 0.37$. Therefore, a total of 20 simulations were performed. For each simulation, the position and velocity results were compared to the analytical solutions at each time step. Figures 7.8-7.10 show the largest magnitude position and velocity errors for the $x$-, $y$-, and $z$-directions respectively for each simulation. The results demonstrate first-order behavior for both solvers and both sliding and sticking cases, as expected. The APGD results generally show smaller errors than the Seidel solver for all cases.

### 7.1.2 Sphere on Incline

In this test, a sphere moves on an inclined plane. The sphere has radius $r$, mass $m$, and friction coefficient $\mu$. The plane is inclined at $\theta$ to the horizontal. The sphere is given an initial velocity parallel to the inclined plane and zero initial rotational velocity such that the sphere slides over the plane initially.
Figure 7.5: X-direction position and velocity results for block sticking on incline ($\mu = 0.37 > \mu_c$).
Figure 7.6: Y-direction position and velocity results for block sticking on incline ($\mu = 0.37 > \mu_c$).
Figure 7.7: Z-direction position and velocity results for block sticking on incline ($\mu = 0.37 > \mu_c$).
Figure 7.8: Largest magnitude error in x-direction position and velocity results for block on incline.
Figure 7.9: Largest magnitude error in y-direction position and velocity results for block on incline.
Figure 7.10: Largest magnitude error in z-direction position and velocity results for block on incline.
Figure 7.11: The sphere is given an initial velocity parallel to the inclined plane. The sphere slides and eventually begins rolling without slip.
At the transition time, $t = t_t$, the sphere begins rolling without slip. While the sphere is slipping on the plane, the equations of motion can be written as

\begin{align*}
    m\ddot{x}_t &= mg\sin(\theta) - \mu mg \cos(\theta) \quad (7.6) \\
    \frac{2}{5}mr^2\ddot{\phi} &= \mu mg r \cos(\theta) \quad (7.7)
\end{align*}

where $x_t$ is the position of the sphere along the slope, $\phi$ is the orientation of the sphere, and $g = 9.81 \text{ m/s}^2$.

These equations can be simplified to the following form.

\begin{align*}
    \ddot{x}_t &= g \sin(\theta) - \mu g \cos(\theta) \quad (7.8) \\
    \ddot{\phi} &= \frac{5\mu g \cos(\theta)}{2r} \quad (7.9)
\end{align*}

By integrating Equations 7.8 and 7.9 each by time, the following expressions are obtained for the translational and rotational velocity, where $v_{t,0}$ and $\omega_0$ are respectively the translational and rotational velocity at time $t = 0$.

\begin{align*}
    \dot{x}_t(t) &= g \left( \sin(\theta) - \mu \cos(\theta) \right) t + v_{t,0} \quad (7.10) \\
    \dot{\phi} &= \frac{5\mu g \cos(\theta)}{2r} + \omega_0 \quad (7.11)
\end{align*}

At the transition time, $t = t_t$, the sphere rolls without slip and therefore satisfies $\dot{x}_t = r \dot{\phi}$. By combining this constraint with Equations 7.10 and 7.11, the following expression can be obtained for $v_{t,0}$, the initial translational velocity required for the transition to pure rolling to occur at $t = t_t$.

\begin{align*}
    v_{t,0} &= \left( \frac{7}{2} \mu g \cos(\theta) - g \sin(\theta) \right) t_t + \omega_0 r \quad (7.12)
\end{align*}

By integrating Equations 7.10 and 7.11 with respect to time, the following expressions can be obtained for $x_t$ and $\phi$. 
\[
x_t = 0.5g \left( \sin(\theta) - \mu \cos(\theta) \right) t^2 + v_{t,0} t \tag{7.13}
\]
\[
\phi = 0.5 \frac{5\mu g \cos(\theta)}{2r} t^2 + \omega_0 t \tag{7.14}
\]

In the pure rolling regime, for \( t > t_t \), the equations of motion are as follows.

\[
m \ddot{x}_t = mg \sin(\theta) - f_f \tag{7.15}
\]
\[
\frac{2}{5} mr^2 \ddot{\phi} = rf_f \tag{7.16}
\]
\[
r \dddot{\phi} = \ddot{x}_t \tag{7.17}
\]

Solving Equations 7.15 through 7.17 for \( \ddot{x}_t, \ddot{\phi}, \) and \( f_f \) yields the following.

\[
\ddot{x}_t = \frac{5}{7} g \sin(\theta) \tag{7.18}
\]
\[
\ddot{\phi} = \frac{5g \sin(\theta)}{7r} \tag{7.19}
\]
\[
f_f = \frac{2}{7} mg \sin(\theta) \tag{7.20}
\]

Integrating Equations 7.18 and 7.19 with respect to time yields the expressions for \( \dot{x}_t \) and \( \dot{\phi} \), where \( v^*_t \) and \( \omega^* \) respectively are the translational and rotational velocities at \( t = t_t \).

\[
\dot{x}_t = \frac{5}{7} g \sin(\theta)t + v^*_t \tag{7.21}
\]
\[
\dot{\phi} = \frac{5g \sin(\theta)}{7r}t + \omega^* \tag{7.22}
\]

Equations 7.10, 7.11, 7.21, and 7.22 can be solved simultaneously at \( t = t_t \) for \( v^*_t \) and \( \omega^* \).

\[
v^*_t = \frac{2}{7} gt_s \sin(\theta) - g \mu t_s \cos(\theta) + v_{t,0} \tag{7.23}
\]
\[
\omega^* = \frac{5\mu g \cos(\theta)}{2r} t_t - \frac{5g \sin(\theta)}{7r} t_t + w_0 \tag{7.24}
\]
Finally, Equations 7.21 and 7.22 can be integrated to obtain expressions for $x_t$ and $\phi$ valid for $t > t_t$.

\[ x_t = 0.5 \frac{5}{7} g \sin(\theta) t^2 + v_t^* t + x_t^* \] (7.25)

\[ \phi = 0.5 \frac{5g \sin(\theta)}{7r} t^2 + \omega^* t + \phi^* \] (7.26)

Equations 7.13, 7.14, 7.25 and 7.26 can be solved simultaneously at $t = t_t$ for $x_t^*$ and $\phi^*$.

\[ x_t^* = 0.5 g \mu \cos(\theta) t_t^2 - \frac{1}{7} g \sin(\theta) t_t^2 \] (7.27)

\[ \phi^* = -0.5 \frac{5 \mu g \cos(\theta)}{2r} t_t^2 + \frac{5g \sin(\theta)}{14r} t_t^2 \] (7.28)

Finally, the evolution of the system is expressed as follows:

\[ x_t(t) = \begin{cases} 
0.5 g (\sin(\theta) - \mu \cos(\theta)) t^2 + v_{t,0} t & : t <= t_t \\
0.5 \frac{5}{7} g \sin(\theta) t^2 + v_t^* t + x_t^* & : t > t_t 
\end{cases} \] (7.29)

\[ \phi(t) = \begin{cases} 
0.5 \frac{5 \mu g \cos(\theta)}{2r} t^2 + \omega_0 t & : t <= t_t \\
0.5 \frac{5g \sin(\theta)}{7r} t^2 + \omega^* t + \phi^* & : t > t_t 
\end{cases} \] (7.30)

\[ \dot{x}_t(t) = \begin{cases} 
g (\sin(\theta) - \mu \cos(\theta)) t + v_{t,0} & : t <= t_t \\
\frac{5}{7} g \sin(\theta) t + v_t^* & : t > t_t 
\end{cases} \] (7.31)

\[ \dot{\phi}(t) = \begin{cases} 
\frac{5 \mu g \cos(\theta)}{2r} + \omega_0 & : t <= t_t \\
0.5 \frac{5g \sin(\theta)}{7r} t^2 + \omega^* t + \phi^* & : t > t_t 
\end{cases} \] (7.32)

The expressions in Equations 7.29 through 7.32 are the analytical velocities, positions, and orientations for the sphere which transitions from slip to pure rolling. Simulation results using the iterative methods described in Chapter 4 will be compared to these analytical results. For this comparison, the transition time was selected to be $t_t = 0.5$ s, the time step was $h = 0.001$ s, and $t_{end} = 3.0$ s. The mass was $m = 1$ kg, the radius was $r = 1$ m, the friction coefficient was $\mu = 0.3$, ...
Figure 7.12: X-direction position and velocity results for sphere on inclined plane.
Figure 7.13: Y-direction position and velocity results for sphere on inclined plane.
Figure 7.14: Z-direction position and velocity results for sphere on inclined plane.
and the incline was $\theta = 20^\circ$. The simulation was performed with both the Iterative Seidel solver and the APGD solver.

Figures 7.12, 7.13, and 7.14 show the position and velocity results in the $x$-, $y$-, and $z$-directions respectively for sphere on inclined plane experiment. For all components, the absolute errors can be seen to be on the order of $10^{-3}$ or smaller. Overall, position and velocity results for both solvers show good agreement with the analytical solution, and very similar behavior with each other.

7.1.3 Stack of Spheres

In this test, a set of 10 spheres are stacked in a vertical column resting on a ground plane. The spheres are released from rest, and the system is allowed to evolve dynamically for 50 seconds. Analytically, the spheres should remain motionless throughout the simulation. However, numerical errors may accumulate and cause the stack to buckle and collapse. The simulation parameters used are $r = 1$ m, $m = 1$ kg, $\mu = 0.3$, and $h = 0.001$ s. The iterative Seidel and APGD methods were both used to perform the same simulation.

Both solvers produced position and velocity results exactly equal to zero for the $x$ and $z$ coordinates for all bodies for the duration of the simulation, as expected. Some difference could be seen in the $y$-coordinate.

Figure 7.15 shows the $y$-coordinate position error for each sphere. Note that the APGD solver matched the expected results exactly, while the Seidel solver had error on the order of $10^{-5}$ m.

Figure 7.16 shows the $y$-coordinate velocity for each sphere. Recall that the analytical solution is exactly zero for each body. From Figure 7.16, one can see that the APGD solver has a slightly larger spread than the Seidel solver.

The simulation results demonstrate that both methods are stable in terms of position. The stack remains standing throughout the entire 50 s simulation using both solvers. It is interesting that the APGD solver has slightly larger errors in terms of velocity, but smaller errors in terms of position.
Figure 7.15: Y-Position error results for Seidel and APGD solvers for stack of 10 spheres.
Figure 7.16: Y-Velocity results for Seidel and APGD solvers for stack of 10 spheres.
7.2 Experimental Validation

The purpose of this section is to validate the developed numerical methods and the implementation thereof for large many-body dynamics problems. Two different cases are considered. In the first, granular material flows out of a container and the mass of the collected material is measured over time. A corresponding simulation is performed, and the time evolution of the mass of the collected material is compared to the same data from the physical experiments. In the second, a large spherical object impacts a bed of granular material. A set of simulations is performed, with various impact velocities and ball masses, in which the penetration depth is measured. The simulation results are compared to an empirical expression based on experimental data.

7.2.1 Mass Flow Rate

In this section, simulation results with the APGD solver are compared to experimental results for the flow of granular material. Because of the inherent difficulty in experimentally observing/validating the motion of individual particles, the validation effort focuses on bulk properties of the granular material. This validation experiment has been previously performed for the SEIDEL and JACOBI methods [77, 35, 76]. Note that the same experimental data is used here, yet it is compared instead to simulation results obtained with the APGD solver to validate this new numerical solution method.

In the experiment, the granular material was composed of approximately 39,000 spherical glass particles. The glass spheres were assumed to be identical with diameter of $500 \mu$m and mass of $1.6315 \times 10^{-7}$ kg based on a material density of 2500 kg/m$^3$. The experimental setup was composed of two machined components, a translational stage, a linear actuator, and a scale/collector. The boundary geometry was composed of the two machined components. The fixed component had a slot with a dimension of 0.009525 m. The moving component was angled at 45° and fit inside the slot. The gap at the bottom of the slot could be precisely controlled with the linear actuator. Full details of the experimental setup can be found in [77, 35]. Experiments were performed for gap sizes of 2.0 mm, 2.5 mm, and 3.0 mm.
Figure 7.17: Schematic of mass flow rate validation experiment in closed (upper) and open (lower) configurations.
Figure 7.17 shows a 2D schematic of the experiment in the open and closed positions. The trough was filled with a fixed amount of granular material. The translating wall was moved to open the desired gap size, and the granular material flowed out onto the scale where the mass was measured over time.

To perform the corresponding simulations, a model was constructed with the same dimensions. The position and velocity of the body representing the translating wall was set at each time step to match the motion in the experiment. The APGD solver was used with a tolerance of $10^{-3}$, and a time step of $10^{-3}$ s. Note that there was no scale in the simulation model, but rather the number of spheres below a specified measuring line were simply counted and the associated weight was computed.

The friction coefficient was determined via a tuning process. To this end, a set of simulations was performed with various value for the friction coefficient at a gap size of 2.0 mm. The values considered were $\mu = 0.15$, $\mu = 0.3$, and $\mu = 0.4$. The simulation results can be seen in Figure 7.18 compared to five experimental runs for the same gap size. As can be seen in the figure, the
friction value which corresponds best to the experimental results is $\mu = 0.3$. This value is then selected and used for the remaining tests. Note that in [35] and [76], a similar tuning process was undertaken when validating the Jacobi and Seidel implementations. In that work, the selected value was $\mu = 0.15$.

Results of this validation effort can be seen in Figures 7.19, 7.20, and 7.21 for gap sizes of 3.0 mm, 2.5 mm, and 2.0 mm, respectively. Each plot shows the results of five experimental runs, and one corresponding simulation. As can be seen in the results, the simulation results correspond favorably with experimental results.

7.2.2 Ball Drop

This section seeks to validate the developed method for even larger systems than those considered in previous sections. Here, a spherical projectile is simulated as it impacts a bed of granular material. The results are compared to an empirical expression determined by a comprehensive set of physical experiments [78, 79, 80].
Figure 7.20: Weight of collected granular material vs. time for 2.5 mm gap size.

Figure 7.21: Weight of collected granular material vs. time for 2.0 mm gap size.
Specifically, this experiment studied the low-speed impact of spheres into granular material and the resulting crater formation. The scenarios considered were limited to the case where the penetration depth was less than or equal to the diameter of the impacting sphere.

In each experiment, a ball of diameter $D_b$ and density $\rho_b$ was dropped from a height $h$ above the surface of a bed of granular material. The granular material was assumed to have bulk density $\rho_g$ and a friction coefficient $\mu$ corresponding to the characteristic angle of repose $\theta_r$ as $\mu = \tan(\theta_r)$. A 2D schematic of the experiments can be seen in Figure 7.22.

It was shown in [78] and confirmed in [80] that the penetration depth $d$ had the following scaling, where the total drop distance was $H = h + d$:

$$d = \frac{0.14}{\mu} \left( \frac{\rho_b}{\rho_g} \right)^{1/2} D_b^{2/3} H^{1/3} \quad (7.33)$$

Note that in [80], all experiments were performed using a granular medium composed of glass spheres with a diameter range of 0.25 mm-0.35 mm and material density of 2.5 g/cm$^3$. In [78], a variety of granular materials were used, ranging from popcorn kernels of size 4 mm $\times$ 6 mm $\times$ 7 mm to beach sand of size $(0.5 \pm 0.4)$ mm. The same scaling was observed for all considered granular materials. In fact, results provided in [78] state that identical cratering was observed for 0.2 mm and 1.0 mm diameter glass beads, indicating that the grain size is not important while the grain density and friction coefficient is important in the penetration depth.

For the purposes of validation, a collection of simulations was performed with the APGD solver with varying parameter values. All simulations used the same granular medium, which represented glass spheres of 1.0 mm diameter, material density of 2.5 g/cm$^3$, and friction coefficient of $\mu = 0.3$. The drop height $h$ was varied from 5 cm to 20 cm, and the density of the ball was varied from 0.28 g/cm$^3$ to 2.2 g/cm$^3$, while the diameter was held fixed at 2.54 cm. The considered scenarios are described in Table 7.1.

The simulation procedure involved two main steps. First, a settling simulation was performed to obtain the configuration of loosely-packed granular material. Four walls and a floor were created, resulting in a box with an interior length and width of 9 cm and a depth of 14.5 cm. A total
Figure 7.22: Schematic of ball drop validation experiment showing initial and final positions of projectile.
Table 7.1: Parameter values for various simulation parameters.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>h [cm]</th>
<th>$\rho_b$ [g/cm$^3$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0.28 (hollow polypropylene)</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.28</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.28</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>0.7 (wood)</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.7</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>0.7</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>2.2 (teflon)</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>2.2</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>2.2</td>
</tr>
</tbody>
</table>
of 500,400 spherical bodies with 1.0 mm diameter and material density of 2.5 g/cm$^3$ were initialized in random positions above the box and allowed to settle due to gravity. When the bodies had reached steady state positions, the simulation was halted and the state of each body was stored to a file. When settled, the spheres occupied a depth of 5.38 cm in the box. Therefore, the packing fraction of the settled spheres was computed to be 0.6012, and the bulk density of the granular material was then $\rho_g = 1.51$ g/cm$^3$. Note that the experiments reported in [78] and [80] used preparations with packing fractions of 0.64 and 0.59, respectively, and the same scaling behavior was observed in both cases. This indicates that the same scaling should be expected in the simulations, given the packing fraction used therein.

The second step consisted of performing the simulations of the impact into the granular material. For each simulation, the granular material was initialized based on the state data from the settling simulation. This way, each simulation started with the same initial conditions. The large spherical impactor was created just above the surface of the settled bodies, with an initial velocity in the $y$-direction of $\dot{y}_0 = -\sqrt{-2gh}$, where $g = -9.81$ m/s$^2$ and $h$ was the drop height.

Each simulation used a time step of $2 \times 10^{-5}$ s, a tolerance of $10^{-3}$, and a maximum of 1,000 iterations at each time step. The simulations were each performed with 8 OpenMP processes.

The penetration depth $d$ was measured by monitoring the vertical position of the center of the impacting sphere. When the vertical position reached its minimum value, $y_{\text{min}}$, the penetration depth was calculated as

$$d = y_{\text{sf}} - y_{\text{min}} + r_s,$$

where $y_{\text{sf}}$ was the position of the top surface of the undisturbed granular material and $r_s$ was the radius of the impacting sphere. After the settling simulation it was determined that $y_{\text{sf}} = 0.0588$ m.

Figure 7.23 shows four snapshots from the impact simulation with $h = 20$ cm and $\rho_s = 0.7$ g/cm$^3$. The four snapshots come at $t = 0.0008$ s, $t = 0.0128$ s, $t = 0.0248$ s, and $t = 0.0368$ s respectively. Note that the last snapshot represents the instant of deepest penetration. In the images, each granular body is colored by its velocity magnitude, where blue represents low velocity and red represents
high velocity. Additionally, note that the images show a cut-away through the center of the projectile so that the profile of the crater can be seen.

Figure 7.24 shows the time evolution of the vertical position of the bottom of the penetrating sphere relative to the surface of the granular material for the considered simulation tests. This quantity is the negative of the penetration distance defined in Equation 7.34. Figure 7.24 shows some important properties of these granular impact simulations. First, penetration depth increases with increasing drop height (and associated impact velocity) and fixed impactor mass. Second, penetration depth increases with impactor mass for fixed drop height. Both of these results are as expected. Finally, note that the deepest penetration for each test is marked by the square marker on each line in Figure 7.24. It is interesting to note that for fixed impactor mass, tests with higher impact velocities achieve larger penetrations in less time than those with lower impact velocities.

The simulation results for all tests are summarized in Figure 7.25, which shows penetration depth vs. scaled total drop distance as in Equation 7.33. The result of each simulation is given by a marker on the plot, where drop height is specified by the marker shape and impactor density is specified by marker color. The blue line on the plot is the result of a linear regression, with the corresponding fit equation and $R^2$ value given on the plot. The pink line represents the linear fit from [80], which has a slope of $0.14/\mu = 0.4666$.

The linear fit based on the simulation results suggests that the penetration depth satisfies the following scaling:

$$d = \frac{0.1221}{\mu} \left( \frac{\rho_b}{\rho_g} \right)^{1/2} D_b^{2/3} H^{1/3}$$

(7.35)

Note that the scaling observed in the simulation results is the same as that observed experimentally (see Equation 7.33, [80]), except for the empirical constant. Note that because all considered simulations used the same granular material (and therefore the same friction coefficient), this data is not sufficient to argue the correctness of the empirical constant. For these tests, the empirical constants determined from simulation and experiments show 13.7% difference. However, the other parameters showed the same scaling behavior in both the simulation and experimental tests.
Figure 7.23: Snapshots from impact simulation with $h = 20 \text{ cm}$ and $\rho_s = 0.7 \text{ g/cm}^3$
Figure 7.24: Penetration depth vs. time for a variety of simulation experiments
Figure 7.25: Penetration depth vs. scaled total drop distance for a variety of simulation experiments
Figure 7.26: Snapshot of the instant of deepest penetration from each simulation
For completeness, Figure 7.26 shows the instant of deepest penetration for each considered simulation.
Chapter 8

Demonstration of Technology

The purpose of this chapter is to further demonstrate the capabilities of the domain-decomposition approach applied to the discrete element method (see Chapter 5). Note that the simulations described here are done for demonstration purposes and do not solve a specific engineering problem. However, the systems considered here are representative of real-world engineering problems and could be adapted for use in the design process with knowledge of realistic parameters, for example. The capabilities demonstrated here are state of the art in terms of many-body dynamics at the time of this thesis.

8.1 Mars Rover

The model considered in this section is that of a wheeled vehicle operating on granular terrain. The vehicle is modeled after the Mars Exploration Rover Opportunity. Dimensions were estimated from available references and images. A schematic representation of the rover, as used in the simulation, can be seen in Figure 8.1. The associated dimensions used in the simulation are given in Table 8.1. The mass of the rover chassis was set to 100 kg, and the mass of each wheel was 3 kg. Note that the wheels were cylinders with radius 10 cm and width 12.5 cm. Each wheel has 30 grousers distributed evenly around the circumference. Each grouser extends across the entire width of the wheel, has a thickness of 0.5 cm, and extends away from the wheel by 0.5 cm.
Figure 8.1: Schematic of Mars Rover geometry: (top) top view, (bottom) side view
Table 8.1: Dimensions of Mars Rover

<table>
<thead>
<tr>
<th>Dimension</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$y_1$</th>
<th>$z_1$</th>
<th>$z_2$</th>
<th>$z_3$</th>
<th>$H_1$</th>
<th>$L_1$</th>
<th>$W_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value [m]</td>
<td>0.441</td>
<td>0.038</td>
<td>0.679</td>
<td>0.409</td>
<td>0.484</td>
<td>0.398</td>
<td>0.381</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
</tr>
</tbody>
</table>

8.2 Simulation 1: Two Million Terrain Bodies

In this simulation, the terrain was composed of 2,016,000 rigid spherical bodies with radius 5 mm and mass 4.5 g. The terrain bodies were initialized in random positions above the simulation domain and allowed to settle under gravity. The simulation domain was rectangular, with length of 3 m in the driving direction and width of 1.5 m. All six wheels of the rover were driven with constant angular velocity of $\pi$ rad/s, or 30 rpm.

In terms of simulation parameters, the time step was $10^{-5}$ s, and 6 s of dynamics were simulated, requiring about 298 hr of computation time. The DEM stiffness and damping parameters (see Equations 2.95-2.98) had the values $k_n = k_t = 2 \times 10^5$, $\gamma_n = 750$, and $\gamma_t = 0$.

Further, this simulation was performed in the Domain Decomposition framework with the Discrete Element Method. To this end, the simulation domain was divided into 64 sub-domains by dividing both the length and the width evenly into 8 sub-domains in each direction. The resulting sub-domains were rectangular, with length 0.375 m and width 0.1875 m. The depth direction was not divided, so each sub-domain extended infinitely in the depth direction.

A snapshot from the simulation can be seen in Figure 8.2. Note that terrain particles are colored by sub-domain, and shared bodies are colored white. Further, the wheels of the rover are checkered with the color of the master sub-domain. In Figure 8.2 the wheels are pink, which indicates that the pink sub-domain has the master copy of the entire rover vehicle.

Figure 8.3 shows the “footprint” of the rover. The terrain particles are colored uniformly, hiding the sub-domain division, and the rover and its wheels are not shown. In this manner, the interaction of the wheels with the terrain can be directly observed.
Figure 8.2: Snapshot from Mars Rover simulation at $t = 4.41$ s, with two million terrain bodies.
Figure 8.3: Snapshot from Mars Rover simulation at $t = 4.41$ s, showing “footprint” of rover wheels with grousers in granular terrain composed of two million rigid bodies.
8.3 Simulation 2: Eight Million Terrain Bodies

A similar simulation was performed again [81], this time with the terrain composed of 8,257,536 rigid spherical bodies with radius 3.125 mm and mass 1.099 g. Most aspects of this simulation were the same as in the previous simulation. For example, the same procedure was used to initialize the terrain bodies, the simulation domain had the same dimensions, and the same driving constraints were used. Additionally, the same time step and DEM force model parameters were used.

In this case, the simulation domain was divided into 225 sub-domains by dividing both the length and the width evenly into 15 sub-domains in each direction, resulting in rectangular sub-domains with length 0.2 m and width 0.1 m. The simulation represented 2.1 s of dynamics, which required about 168 hr of computation time using 225 cores split between four nodes of the cluster.

A snapshot from the simulation can be seen in Figure 8.4, where once again the terrain particles are colored by sub-domain, and shared bodies are colored white.

Note that this simulation contained 4.096 times the number of bodies as the previous simulation, and represented 0.35 times the length of dynamics. Therefore, based on the first simulation it could be estimated that a 2.1 s long simulation with 8,257,536 bodies would take approximately 298 hr × 4.096 × 0.35 = 427.2 hr using 64 cores. Note that the second simulation actually took 168 hr using 225 cores. Therefore, it is estimated that a speedup factor of 2.54 was achieved by using 3.51 times as many cores. This sub-optimal scaling is expected due to the communication overhead. Additionally, note that this performance compares well to the scaling experiment in Section 5.2.1, where a speedup of 3.122 was achieved by using 4 times as many cores. This demonstrates that the implementation scales well even for large problem sizes and large numbers of cores. Finally, based on these observations we note that the implementation is likely latency-bound, not bandwidth-bound.
Figure 8.4: Snapshot from Mars Rover simulation at $t = 1.0$ s, with over eight million terrain bodies.
Chapter 9

Conclusions

This thesis describes the development, implementation, testing, and validation of improved methods for the simulation of many-body systems with friction and contact. Specifically, the two most commonly used approaches for handling friction and contact in many-body dynamics are improved to enable the simulation of larger systems of rigid bodies. The discrete element method (DEM) is extended through a domain decomposition approach to support the use of distributed computing hardware. The differential variational inequality (DVI) approach is enhanced through the use of new numerical methods with improved convergence properties. Throughout this work, care is taken to support emerging computational hardware via parallel computing. Specifically, the DEM approach supports large computational clusters in a distributed memory architecture. The DVI approach supports both GPU and multi-core architectures through CUDA and OpenMP, respectively. In this way, the developed implementations should scale well as computing hardware continues to improve.

Specific contributions of the author include:

- Developed new numerical methods for the Differential Variational Inequality formulation which demonstrate improved convergence properties:
  - Developed Gradient-Projected Minimum Residual (GPMINRES) method for frictionless problems
  - Developed Accelerated Projected Gradient Descent (APGD) method for frictional problems
  - Implemented APGD to run in parallel with OpenMP or GPU programming
– Demonstrated potential of Interior Point methods for future work

• Implemented domain decomposition approach for Discrete Element Method formulation:
  – Developed domain decomposition, inter-domain communication, and synchronization methods to perform distributed many-body dynamics simulations on computing cluster
  – Demonstrated scaling behavior of the implementation

• Validated APGD method against analytical solutions for simple scenarios

• Validated APGD method against experimental data for two scenarios

• Demonstrated capabilities of new approaches through simulation of Mars Rover vehicle

These developments provide an important step towards larger, more realistic simulations. The APGD method demonstrates speedups of 2-20 times when compared to existing, commonly used methods in the DVI approach. Further, APGD scales better as problems become larger or more challenging. For example, APGD demonstrates improved relative performance for problems with deeper stacks, smaller tolerances, and higher friction coefficients. Additionally, the APGD method was validated against both analytical and experimental tests. The domain decomposition approach has been successfully used to simulate a problem with 8 million terrain particles. Such capability begins to approach the scale needed to perform realistic simulations of vehicles interacting with granular terrain, as is demonstrated in the Mars Rover example.

Combined, this work advances the state of the art in both the DVI and DEM formulations for handling friction and contact. Simulation speed and problem size are increased in both methods, allowing more realistic simulations to be performed. Finally, the new DVI method has been validated for two large, complex scenarios, indicating that simulation can be used predictively for future tests.

9.1 Future Work

The work carried out here suggests several interesting directions for continued or future work. Perhaps the most promising technical issue left under-developed by this work is the use of Interior
Point (IP) methods for solving the cone complementarity problem (CCP) at each time step in the DVI approach. The IP method was implemented in this work in MATLAB for solving a single CCP, but not in C++ in the Chrono framework for solving dynamic problems. However, even the preliminary single-step numerical experiments show that the IP method can solve the CCP with very few iterations. Note that the major deterrent to using an IP method in this context is the required solution of a large sparse linear system. However, recent work suggests that solving such linear systems may be possible while leveraging parallel computing. For example, it may be possible to re-order the large sparse matrix such that it has a dense but narrow band of non-zeros, a case which is well-handled by a recently developed algorithm [82].

Another technical issue which deserves further consideration is warm-starting in the iterative solution process. In the current implementation, each time step is essentially solved independently. For this reason, even in a quasi-static simulation of settled granular material, the solution of each time step is equally difficult. However, the configuration of the bodies in the system may be changing very little, and one would expect both the velocity and the contact impulse solutions to be very similar from one time step to the next. Therefore, improved performance of the code may be achieved by warm-starting the iterative solution process with an initial guess based on the solution from the previous time step. The implementation of such an approach is challenging due to the need to match-up contacts from the current and previous time steps. An alternative approach may be to use a coarse-grained, easy to compute representation of the granular material to initialize contact impulses based on their spatial location.

An implementation task which may provide significant benefit is the development of an MPI-OpenMP hybrid approach to the distributed DEM code developed in Chapter 5. The code developed there uses a single MPI process for each sub-domain in the domain decomposition approach. Emerging many-core architectures may have over 60 CPU cores on a single compute node. In this case, higher performance may be achieved by using the shared-memory OpenMP parallelism. In the current implementation, for example, a simulation on 64 cores on a single node spends a significant amount of time passing messages between MPI ranks. However, because the cores are on a single node they have access to the same memory, and it would likely be advantageous to
instead use the cores to parallelize the execution of for-loops via OpenMP. In fact, a hybrid approach would likely provide the best performance. The same MPI-enabled domain decomposition approach should be used at a high level, with each MPI rank managing a collection of OpenMP threads. MPI would only be used to communicate between compute nodes, while OpenMP parallelism would be used within a node.

Continued validation work would also be valuable. For example, the granular impact validation of Section 7.2.2 could be continued to verify the effect of friction on penetration depth and resolve the differences observed in the scaling behavior between the experimental and simulation results. With the validated simulation set-up, it may be possible to use the simulation framework to gain further insight into granular dynamics. For example, the effect of particle geometry could be studied very accurately via computer simulation.
LIST OF REFERENCES


APPENDIX
Using MATLAB to Compare Iterative Methods

This appendix provides a description of the use of MATLAB to compare iterative methods for solving problems in the DVI formulation. Specifically, this section relates to the solution of problems in the form of Equation 2.46. For conciseness, the problem of interest is restated here.

\[
\begin{align*}
\min q(\gamma) &= \frac{1}{2} \gamma^{(l+1)T} N \gamma^{(l+1)} + r^{T} \gamma^{(l+1)} \\
\text{subject to } &\gamma^{(l+1)}_{i} \in \Upsilon_{i} \text{ for } i = 1, 2, \ldots, N_c \\
\text{where } N &= D^{T} M^{-1} D \in \mathbb{R}^{3n_c \times 3n_c} \\
r &= b + D^{T} M^{-1} k \\
k &= f + M v^{(l)}
\end{align*}
\] (A.1)

A.1 Extracting Terms from Chrono

The terms required by Equation A.1 can be extracted from the Chrono simulation software. To this end, the simulation of interest should be set up in the Chrono framework. At the desired point in the simulation, function calls can be made to save each of the desired terms to a file. The terms which will be saved include the following: \( M, D, E, f, b, \mu, v^{(l)} \)

In the Chrono framework, note that the state of the physical system is advanced by one time step by making a call as follows.

Listing A.1: Performing a Single Time Step in Chrono

```c
mphysicalSystem->DoStepDynamics(time_step);
```
Due to the internal data structures used in Chrono, the term $v^{(l)}$ must be saved before the step is taken. Because the Chrono implementation is matrix-free and the matrix terms are never explicitly formed, the other terms are only available after the time step is complete. Therefore, the following code demonstrates the calls needed to extract the required terms and save them to data files.

Listing A.2: Extracting Matrix and Vector Terms from Chrono

```cpp
try {
    chrono::ChMatrixDynamic<double> mdV0;
    mphysicalSystem->GetLcpSystemDescriptor()->FromVariablesToVector(mdV0);
    chrono::ChStreamOutAsciiFile file_V0("Matrices\dump_V_old.dat");
    mdV0.StreamOUTdenseMatlabFormat(file_V0);
}
catch(chrono::ChException myexc) {
    chrono::GetLog() << myexc.what();
}
mphysicalSystem->DoStepDynamics(time_step);
try {
    chrono::ChSparseMatrix mdM;
    chrono::ChSparseMatrix mdCq;
    chrono::ChSparseMatrix mdE;
    chrono::ChMatrixDynamic<double> mdf;
    chrono::ChMatrixDynamic<double> mdb;
    chrono::ChMatrixDynamic<double> mdfric;
    mphysicalSystem->GetLcpSystemDescriptor()->ConvertToMatrixForm(&mdCq,
    &mdM, &mdE, &mdf, &mdb, &mdfric);
    chrono::ChStreamOutAsciiFile file_M("Matrices\dump_M.dat");
    mdM.StreamOUTsparseMatlabFormat(file_M);
    chrono::ChStreamOutAsciiFile file_Cq("Matrices\dump_Cq.dat");
    mdCq.StreamOUTsparseMatlabFormat(file_Cq);
    chrono::ChStreamOutAsciiFile file_V0("Matrices\dump_V_old.dat");
    mdV0.StreamOUTdenseMatlabFormat(file_V0);
}
```
The function `StreamOUTdenseMatlabFormat` simply writes each entry of the vector to a file, one entry per line. The function `StreamOUTsparseMatlabFormat` writes one line to the file for each entry in the sparse matrix. Each line has three entries, the row index, the column index, and the value, separated by spaces. Note that row and column indices are 1-based in MATLAB, not 0-based as in C++.

### A.2 Importing and Working in MATLAB

The saved matrix/vector terms can be loaded into MATLAB using the following function. Note that the input argument is the path to the folder containing the saved files.

**Listing A.3: Function to Load Terms into MATLAB**

```matlab
function [M,D,fh,b,v0,fric] = create_system_from_dumps(path)
M=spconvert(load(strcat(path,'dump_M.dat')));
D=spconvert(load(strcat(path,'dump_Cq.dat')));
fh=load(strcat(path,'dump_f.dat'));
b=load(strcat(path,'dump_b.dat'));
E=spconvert(load(strcat(path,'dump_E.dat')));
v0=load(strcat(path,'dump_v_old.dat'));
```

Finally, the following MATLAB code can be used to compute the matrix $N$ and vector $r$ required in Equation A.1.

Listing A.4: Computing $N$ and $r$ in MATLAB

```matlab
file_loc = 'C:\DATA\Matrices\'
[M, D, fh, b, E, v_old, fric] = create_system_from_dumps(file_loc);
k = fh + M * v_old;
r = b + D' * (M \ k);
N = D' * (M \ D);
```

A.3 Summary of Numerical Experiments

MATLAB code was written to perform the single-step convergence experiments from Section 4.7.1. The tests five tests reported there represent frictionless systems with 1000, 2000, and 4000 bodies, and frictional systems with 1000 and 2000 bodies. The MATLAB code for these experiments is available at [83].