Autotuning Stencil-Based Computations on GPUs

Azamat Mametjanov, Daniel Lowell, Ching-Chen Ma, Boyana Norris

LANS Performance Group
Mathematics and Computer Science Division
Argonne National Laboratory

Thanks to CACHE: Algorithms and Software for Communication Avoidance and Communication Hiding at the Extreme Scale
and SUPER: Sustained Performance, Energy and Resilience
Motivation

- Finite-difference stencils are very common in numerical modeling. They exhibit high degree of data parallelism and regular structure. However, their memory requirements hinder the performance.

- Our solution consists of
  - Exploitation of a stencil’s data access pattern
  - Automatic conversion of C loops to CUDA C host+kernel code
  - Automatic tuning of CUDA C performance parameters
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Our approach
- Results
Stencils

- Sets of neighboring discrete points in a structured grid
- Stencil pattern determines the interaction among points
  - Domain dimension: 1D, 2D, 3D
  - Stencil shape: star, box
  - Stencil width: distance from stencil center
  - Boundary condition: Dirichlet, periodic
### Grid, adjacency matrix and its compression

#### (a)

<table>
<thead>
<tr>
<th>Grid Element</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>57</td>
<td>58</td>
<td>93</td>
<td>94</td>
<td>125</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>59</td>
<td>60</td>
<td>95</td>
<td>96</td>
<td>127</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>26</td>
<td>61</td>
<td>62</td>
<td>97</td>
<td>98</td>
<td>129</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>28</td>
<td>63</td>
<td>64</td>
<td>99</td>
<td>100</td>
<td>131</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>30</td>
<td>65</td>
<td>66</td>
<td>101</td>
<td>102</td>
<td>133</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>32</td>
<td>67</td>
<td>68</td>
<td>103</td>
<td>104</td>
<td>135</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>33</td>
<td>34</td>
<td>69</td>
<td>70</td>
<td>105</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>35</td>
<td>36</td>
<td>71</td>
<td>72</td>
<td>107</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>37</td>
<td>38</td>
<td>73</td>
<td>74</td>
<td>109</td>
<td>110</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>39</td>
<td>40</td>
<td>75</td>
<td>76</td>
<td>111</td>
<td>112</td>
<td>143</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>10</td>
<td>41</td>
<td>42</td>
<td>77</td>
<td>78</td>
<td>113</td>
<td>114</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
<td>43</td>
<td>44</td>
<td>79</td>
<td>80</td>
<td>115</td>
<td>116</td>
<td>147</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>14</td>
<td>45</td>
<td>46</td>
<td>81</td>
<td>82</td>
<td>117</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>16</td>
<td>47</td>
<td>48</td>
<td>83</td>
<td>84</td>
<td>119</td>
<td>120</td>
<td></td>
</tr>
</tbody>
</table>

#### (b)

<table>
<thead>
<tr>
<th>Element</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>57</td>
<td>58</td>
<td>93</td>
<td>94</td>
<td>125</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>61</td>
<td>62</td>
<td>97</td>
<td>98</td>
<td>129</td>
<td>130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>27</td>
<td>28</td>
<td>63</td>
<td>64</td>
<td>99</td>
<td>100</td>
<td>131</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>31</td>
<td>32</td>
<td>67</td>
<td>68</td>
<td>103</td>
<td>104</td>
<td>135</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>35</td>
<td>36</td>
<td>71</td>
<td>72</td>
<td>107</td>
<td>108</td>
<td>139</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>37</td>
<td>38</td>
<td>73</td>
<td>74</td>
<td>109</td>
<td>110</td>
<td>141</td>
<td>142</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>39</td>
<td>40</td>
<td>75</td>
<td>76</td>
<td>111</td>
<td>112</td>
<td>143</td>
<td>144</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>41</td>
<td>42</td>
<td>77</td>
<td>78</td>
<td>113</td>
<td>114</td>
<td>145</td>
<td>146</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>43</td>
<td>44</td>
<td>79</td>
<td>80</td>
<td>115</td>
<td>116</td>
<td>147</td>
<td>148</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>47</td>
<td>48</td>
<td>83</td>
<td>84</td>
<td>119</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>49</td>
<td>50</td>
<td>85</td>
<td>86</td>
<td>121</td>
<td>122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>51</td>
<td>52</td>
<td>87</td>
<td>88</td>
<td>123</td>
<td>124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>53</td>
<td>54</td>
<td>89</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>55</td>
<td>56</td>
<td>91</td>
<td>92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### (c)

- (a) Grid and adjacency matrix
- (b) Adjacency matrix compression
- (c) Adjacency matrix with some elements highlighted
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Overview of the approach
- Results
Method: Code Transformation

- **Motivation**
  - Compilation: HL source code into LL portable executable code
  - Optimization: performance, energy
  - Refactoring: resiliency, maintainability, readability

- **Workflow**
  - Parse: any structured source text into abstract syntax tree
  - Analyze: common intermediate representation
  - Transform: compositions of reusable transforms
  - Generate: any structured target text

- **Challenges**
  - Create source and target domains
  - Create analysis and transformation rules
Method: Code Tuning

- **Motivation**
  - Deep component stacks
  - Each component is adjustable

- **Workflow**
  - System model: pre-specified, learned
  - Application profile: memory-/compute-bound
  - Configure: create a valid configuration of parameters
  - Select: the best performing parameter configuration

- **Challenges**
  - Auto-profile
  - Auto-modify
  - Search
  - Whole-app autotuning
Orio autotuning framework

- Code with DSL Annotations
- DSL Parser
- Sequence of (Nested) Annotated Regions
- Tuning Specification
  - Code Transformsations
- Code Generator
  - Transformed Code
  - Empirical Performance Evaluation
  - Search Engine
  - best performing version
- Optimized Code
  - CUDA
  - Fortran
  - C
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- **Overview of the approach**
- Results
Begin with reference C code

```c
for(i=0; i<=nrows-1; i++) {
    for(j=0; j<=ndiags-1; j++){
        col = i+offsets[j];
        if(col>=0&&col<nrows)
            y[i] += A[i+j*nrows] * x[col];
    }
}
```
Add a DSL annotation

/*@ begin Loop(...

for(i=0; i<=nrows-1; i++) {
  for(j=0; j<=ndiags-1; j++){
    col = i+offsets[j];
    if(col>=0&&col<nrows)
      y[i] += A[i+j*nrows] * x[col];
  }
}
) @*/

for ...

/*@ end @*/
Specify performance parameters (optional)

/*@ begin Loop(transform \textbf{CUDA}( 
  threadCount=TC, 
  blockCount=BC, 
  streamCount=SC, 
  preferL1Size=PL, 
  unrollInner=UIF, ... 
 )
 for ...
 ) @*/

for ...

/*@ end @*/
Specify parameter search ranges

/*@ begin PerfTuning(
def performance_params{
    param TC[] = range(32,1025,32);
    param BC[] = range(14,113,14);
    param SC[] = range(1,17);
    param PL[] = [16,48];
    param UIF[] = range(1,8); ...
}
) @*/
/*@ begin Loop(transform CUDA(
...
/*@ end @*/
Define empirical experiment inputs

/*@ begin PerfTuning(
    def input_params {
        param M[] = [16,32,64,128,256]; ...
    }
    def input_vars {
        decl static double A[M*N*P*NOS*DOF] = random;
        decl static double x[M*N*P*DOF] = random;
        decl static double y[M*N*P*DOF] = 0; ...
    }
    ... ...
) @*/
Define build and search parameters

/*@ begin PerfTuning(
   def build {
       arg build_command = 'nvcc -arch=sm_20 @CFLAGS';
   }
   def performance_counter {
       arg repetitions = 10;
   }
   ...
) @*/
Launch

./orcuda matVec3D.c
...

Search_Space = 1.024e+04
Number_of_Parameters = 05
Numeric_Parameters = 05
Binary_Parameters = 00
['TC', 'BC', 'UIF', 'PL', 'CFLAGS']
[[32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800, 832, 864, 896, 928, 960, 992, 1024], [14, 28, 42, 56, 70, 84, 98, 112], [1, 2, 3, 4, 5], [16, 48], ['', '-O1', '-O2', '-O3']]
Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Overview of the approach
- Results
Reduction kernels

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070

![Bar chart showing the normalized execution time for different kernels. The x-axis represents the kernels (vecDot_e5, vecDot_e6, vecDot_e7, vecNorm2_e5, vecNorm2_e6, vecNorm2_e7), and the y-axis represents the normalized execution time. The chart compares the performance of Orio, CUSP, Custom, and cuBLAS.]
**Pointwise kernels**

*Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070*

---

The chart above illustrates the normalized execution time for various pointwise kernels. The x-axis represents the different kernels, while the y-axis shows the normalized execution time. The kernels are evaluated using three different libraries: Orio, CUSP, and cuBLAS.
Example: Sparse matrix-vector product (5- and 7-point stencil) on a GPU

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070

Lower is better
Application: Bratu solid fuel ignition problem

Up to 1.5x improvement

Lower is better
Conclusion

- Workflow:
  - Functionality
  - Performance
  - Stability

- Application stack is complex
  - Dependency depth
  - Heterogeneity at each level

- Autotuning provides end-to-end integration
  - Hardware and components will continue to change
  - Application only needs to be written once
  - Programmability through portability
Thank you

http://tinyurl.com/OrioTool