Experiences with Achieving Portability across Heterogeneous Architectures

Lukasz G. Szafaryn\textsuperscript{+}, Todd Gamblin\textsuperscript{++}, Bronis R. de Supinski\textsuperscript{++} and Kevin Skadron\textsuperscript{+}

\textsuperscript{+}University of Virginia
\textsuperscript{++}Lawrence Livermore National Laboratory
Outline

• Introduction
• Portability Problems
• Approach
• Molecular Dynamics Code
• Feature Tracking Code
• Conclusions
• Future Work
Introduction

• Cluster nodes are becoming heterogeneous (multi-core CPU and GPU) in order to increase their compute capability

• Codes executing at the level of a node need to be ported to new architectures/accelerators for better performance

• Most existing codes are written for sequential execution, over-optimized for a particular architecture and difficult to port

• We look at 2 representative applications, and for each one of them, we want to arrive at and maintain a single code base that can run on different architectures
Outline

• Introduction
• **Portability Problems**
• Approach
• Molecular Dynamics Code
• Feature Tracking Code
• Conclusions
• Future Work
Portability Problems

Code Structure

• Sequential form
  – Does not express parallelism well; needs effort to extract
  – Many small, independent loops and tasks that could run concurrently
  – Optimizations such as function inlining and loop unrolling

• Monolithic and coupled structure
  – Hard to identify sections that can be parallelized (and/or offloaded)
  – Hard to determine range of accessed data due to nested pointer structure

• Architecture-specific optimizations
  – Specifying native instructions to ensure desired execution
  – Memory alignment via padding
  – Orchestrating computation according to cache line size

• Prospect of significant code changes delays porting
Portability Problems
Parallel Framework

- Inconvenient and inadequate current frameworks
  - Cilk, TBB – API with runtime calls, work only with CPU
  - OpenMP - convenient annotation-based frameworks, works only with CPU
  - OpenCL - portable CPU-GPU API with runtime calls, low-level and too difficult to learn
  - CUDA, CUDAlite - convenient GPU API with runtime calls, vendor-specific
  - Cetus - OpenMP-CUDA translators, limited to OpenMP, heavily relies on code analysis
  - PGI Accelerator API – convenient annotation-based framework, works only with GPU, relies on code analysis
  - Future OpenMP with accelerator support – convenient annotation-based framework, expected to fully support CPU and GPU
Portability Problems

Summary

• Difficulties
  - Achieving portable code written in convenient framework is not possible
  - Porting across architectures requires earning a new, usually architecture-specific, language; scientists are reluctant to do that
  - Need to convert, usually over-optimized architecture-specific, serial code to parallel code; only skilled programmers can do that

• Outcomes
  - Degraded performance after initial porting, difficult to extract parallelism for improvement
  - Divergent code bases that are difficult to maintain
  - Instead, researchers will continue using current languages and corresponding architectures as long as they providing at least some incremental benefits
Outline

• Introduction
• Portability Problems
• **Approach**
  • Molecular Dynamics Code
  • Feature Tracking Code
• Conclusions
• Future Work
Approach

Code Structure

• Clearly expose parallel characteristics of the code to facilitate portability and tradeoffs across architectures

• Create consolidated loop structure that maximizes the width of available parallelism

• Organize code in a modular fashion that clearly delineates parallel sections of the code

• Remove architecture-specific optimizations

```c
... for(...) {
  (independent iter.)
}
dependentTask;
for(...) {
  (independent iter.)
}
dependentTask;
arch-specificCode;
...
```

```c
... for(...) {
  (all independent iter.)
}
dependentTask;

dependentTask;
genericCode;
...
```
Approach
Parallel Framework

• Use portable directive-based approach to
  – Be compatible with current languages
  – Minimize changes to code
  – Eliminate run-time calls in favor of annotations

• Achieve best-effort parallelization with high-level directives

• Tune performance with optional lower-level directives
Approach
Portable Implementation

• Demonstrate approach for writing single code with syntactic and performance portability
• Follow approach of PGI Accelerator API and create higher-level generic framework that can be translated to OpenMP and PGI Accelerator API

```
#define MCPU // or GPU
dependentTask;
#pragma api for parallel
for(...){
  #pragma api for vector
  for(...){
    (independent iter.)
  }
}
dependentTask;
...
```

```
dependentTask;
#pragma acc for parallel
for(...){
  #pragma acc for vector
  for(...){
    (independent iter.)
  }
}
dependentTask;
...
```

```
dependentTask;
#pragma omp parallel for
for(...){
  for(...){
    (independent iter.)
  }
}
dependentTask;
...
```

Our generic framework

Source translation

PGI Accelerator API

OpenMP
Outline

• Introduction
• Portability Problems
• Approach
• **Molecular Dynamics Code**
• Feature Tracking Code
• Conclusions
• Future Work
Molecular Dynamics Code Overview

- Particle space is divided into large boxes (or cells, could be any shape), where each box is divided into small boxes.
- Code computes interactions between particles in any given small box and its neighbor small boxes.

![Diagram of molecular dynamics code overview](image-url)
#define MCPU // or GPU
#pragma api data copyin(box[0:##_boxes-1])
    copyin(pos[0:##_par-1]) copyin(chr[0:##_par-1])
    copyout(dis[0:##_par-1]){
    #pragma api compute{
        #pragma api parallel(30) private(...)
            cache(home_box)
        for(i=0; i<##_home_boxes; i++){
            Home box setup
            #pragma api sequential
            for(j=0; j<##_neighbor_boxes; j++){
                Neighbor box setup
                pragma api vector (128) private(...)
                    cache(neighbor_box)
                for(k=0; k<##_home_particles; k++){
                    pragma api sequential
                    for(l=0; l<##_neighbor_particles;l++){
                        Calculation of interactions
                    }
                }
            }
        }
    }
}
Molecular Dynamics Code

Code Translation

```c
#define MCPU // or GPU
#pragma api data copyin(box[0:_boxes-1])
   copyin(pos[0:_par-1]) copyin(chr[0:_par-1])
   copyout(dis[0:_par-1])
#pragma api compute{
   #pragma api parallel(30) private(...)
   cache(home_box)
   for(i=0; i<_home_boxes; i++){
      Home box setup
      #pragma api sequential
      for(j=0; j<_neighbor_boxes; j++){
         Neighbor box setup
         #pragma api vector (128) private(...)
         cache(neighbor_box)
         for(k=0; k<_home_particles; k++){
            pragma api sequential
            for(l=0; l<_neighbor_particles; l++){
               Calculation of interactions
            }
         }
      }
   }
   ...}
```

Our generic framework

```
omp_set_num_threads(30);

#pragma omp parallel for
for(i=0; i<_home_boxes; i++){
   Home box setup
   for(j=0; j<_neighbor_boxes; j++){
      Neighbor box setup
      for(k=0; k<_home_particles; k++){
         Calculation of interactions
      }
   }
   ...
```

OpenMP

Source translation
Molecular Dynamics Code

Code Translation

```c
#define MCPU // or GPU
#pragma api data copyin(box[0:#_boxes-1])
    copyin(pos[0:#_par.-1]) copyin(chr[0:#_par.-1])
    copyout(dis[0:#_par.-1])
#define api compute{
    #pragma api parallel(30) private(...) \ cache(home_box)
    for(i=0; i<#_home_boxes; i++){
        Home box setup
        #pragma api sequential
        for(j=0; j<#_neighbor_boxes; j++){
            Neighbor box setup
            #pragma api vector (128) private(...) \ cache(neighbor_box)
            for(k=0; k<#_home_particles; k++){
                pragma api sequential
                for(l=0; l<#_neighbor_particles;l++){
                    Calculation of interactions
                }
            }
        }
    }
}
```

Our generic framework

```c
omp_set_num_threads(30);
#pragma omp parallel for
for(i=0; i<#_home_boxes; i++){
    Home box setup
    for(j=0; j<#_neighbor_boxes; j++){
        Neighbor box setup
        for(k=0; k<#_home_particles; k++){
            for(l=0; l<#_neighbor_particles;l++){
                Calculation of interactions
            }
        }
    }
}
```

OpenMP
Molecular Dynamics Code

Code Translation

Our generic framework

```
#define MCPU // or GPU
#pragma api data copyin(box[0:#_boxes-1])
   copyin(pos[0:#_par.-1]) copyin(chr[0:#_par.-1])
   copyout(dis[0:#_par.-1])
#pragma api compute{
   #pragma api parallel(30) private(...)\ cache(home_box)
   for(i=0; i<#_home_boxes; i++){ Home box setup
      #pragma api sequential
      for(j=0; j<#_neighbor_boxes; j++){ Neighbor box setup
         pragma api vector (128) private(...)\ cache(neighbor_box)
         for(k=0; k<#_home_particles; k++){
            pragma api sequential
            for(l=0; l<#_neighbor_particles;l++){
               Calculation of interactions
            }
         }
      }
   }
}
```

PGI Accelerator API

```
#pragma acc data region copyin(box[0:#_boxes-1])
   copyin(pos[0:#_par.-1]) copyin(chr[0:#_par.-1])
   copyout(dis[0:#_par.-1])
#pragma acc region{
   #pragma acc parallel(30) independent private(...)\ cache(home_box)
   for(i=0; i<#_home_boxes; i++){ Home box setup
      #pragma acc sequential
      for(j=0; j<#_neighbor_boxes; j++){ Neighbor box setup
         pragma acc vector (128) private(...)\ cache(neighbor_box)
         for(k=0; k<#_home_particles; k++){
            pragma acc sequential
            for(l=0; l<#_neighbor_particles;l++){
               Calculation of interactions
            }
         }
      }
   }
```

omp_set_num_threads(30);
#pragma omp parallel for
for(i=0; i<#_home_boxes; i++){ Home box setup
   for(i=0; i<#_neighbor_boxes; i++)
```
Molecular Dynamics Code

Code Translation

Our generic framework

#define MCPU // or GPU
#pragma api data copyin(box[0:#_boxes-1])
   copyin(pos[0:#_par.-1]) copyin(chr[0:#_par.-1])
   copyout(dis[0:#_par.-1])
#pragma api compute{
   #pragma api parallel(30) private(...) \ cache(home_box)
   for(i=0; i<#_home_boxes; i++){
      Home box setup
      #pragma api sequential
      for(j=0; j<#_neighbor_boxes; j++){
         Neighbor box setup
         pragma api vector (128) private(...) \ cache(neighbor_box)
         for(k=0; k<#_home_particles; k++){
            pragma api sequential
            for(l=0; l<#_neighbor_particles;l++){
               Calculation of interactions
            } // k
         } // j
      } // i
   } // Home box setup
   } // compute

#pragma api data copyin(box[0:#_boxes-1])
   copyin(pos[0:#_par.-1]) copyin(chr[0:#_par.-1])
   copyout(dis[0:#_par.-1])
#pragma api compute{
   #pragma api parallel(30) private(...) \ cache(home_box)
   for(i=0; i<#_home_boxes; i++){
      Home box setup
      #pragma api sequential
      for(j=0; j<#_neighbor_boxes; j++){
         Neighbor box setup
         pragma api vector (128) private(...) \ cache(neighbor_box)
         for(k=0; k<#_home_particles; k++){
            pragma api sequential
            for(l=0; l<#_neighbor_particles;l++){
               Calculation of interactions
            } // k
         } // j
      } // i
   } // Home box setup
   } // compute

omp_set_num_threads(30);
#pragma omp parallel for
for(i=0; i<#_home_boxes; i++){
   Home box setup
   for(j=0; j<#_neighbor_boxes; j++){
      Neighbor box setup
      for(k=0; k<#_home_particles; k++){
         for(l=0; l<#_neighbor_particles;l++){
            Calculation of interactions
         } // k
      } // j
   } // i
} ...

Our generic framework

OpenMP

define a parallel region with independent private variables

PGI Accelerator API
Molecular Dynamics Code

Workload Mapping

Boxes

vector units
1, 2, 3 … n

home box – home box
particle interactions
---------- SYNC ----------
home box – neighbor
box 1 particle
interactions
---------- SYNC ----------
home box – neighbor
box 2 particle
interactions
---------- SYNC ----------
…
home box – neighbor
box n particle
interactions

vector units
1, 2, 3 … n

home box – home box
particle interactions
---------- SYNC ----------
home box – neighbor
box 1 particle
interactions
---------- SYNC ----------
home box – neighbor
box 2 particle
interactions
---------- SYNC ----------
…
home box – neighbor
box n particle
interactions

vector units
1, 2, 3 … n

home box – home box
particle interactions
---------- SYNC ----------
home box – neighbor
box 1 particle
interactions
---------- SYNC ----------
home box – neighbor
box 2 particle
interactions
---------- SYNC ----------
…
home box – neighbor
box n particle
interactions

vector units
1, 2, 3 … n

home box – home box
particle interactions
---------- SYNC ----------
home box – neighbor
box 1 particle
interactions
---------- SYNC ----------
home box – neighbor
box 2 particle
interactions
---------- SYNC ----------
…
home box – neighbor
box n particle
interactions

global SYNC

processors

1 2 3 … 30

Time step 1

Time step 2
# Molecular Dynamics Code Performance

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-core CPU</td>
<td>Original</td>
<td>C</td>
<td>47</td>
<td>56.19</td>
<td>1</td>
</tr>
<tr>
<td>1-core CPU</td>
<td>Structured</td>
<td>C</td>
<td>34</td>
<td>73.73</td>
<td>0.76</td>
</tr>
<tr>
<td>8-core CPU</td>
<td>Structured</td>
<td>OpenMP</td>
<td>37</td>
<td>10.73</td>
<td>5.23</td>
</tr>
<tr>
<td>GPU</td>
<td>GPU-specific</td>
<td>CUDA</td>
<td>59</td>
<td>5.96</td>
<td>9.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>96</td>
<td></td>
<td>9.43</td>
</tr>
<tr>
<td>1-core CPU</td>
<td>Structured</td>
<td>Our</td>
<td>47</td>
<td>73.72</td>
<td>0.76</td>
</tr>
<tr>
<td>8-core CPU</td>
<td>Structured</td>
<td>Our</td>
<td>47</td>
<td>10.73</td>
<td>5.23</td>
</tr>
<tr>
<td>GPU</td>
<td>Structured</td>
<td>Our</td>
<td>47</td>
<td>7.11</td>
<td>7.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>47</td>
<td></td>
<td>7.91</td>
</tr>
<tr>
<td>GPU</td>
<td>Init/Trans</td>
<td>(CUDA)</td>
<td>---</td>
<td>0.87</td>
<td>---</td>
</tr>
</tbody>
</table>

Our implementation achieves over **50% reduction in code length** compared to OpenMP+CUDA

GPU performance of our implementation is within **17%** of CUDA
Outline

• Introduction
• Portability Problems
• Approach
• Molecular Dynamics Code
• Feature Tracking Code
• Conclusions
• Future Work
Feature Tracking Code
Overview

• Code tracks movement of heart walls (by tracking sample points) throughout a sequence of frames
• Movement of points between frames is determined by comparing areas around points
Feature Tracking Code

Code Translation

```
#define MCPU // or GPU
for(i=0; i<_frames; i++){
    Read frame
    #pragma api data copyin(frm[0:frm_size-1])
    copyin(ini_loc[0:_smp_points-1])
    local(con/cor[0:_pixels]) copyout(fin_loc[0:_-1])
    #pragma api compute{
        #pragma api parallel(30) independent
        for(j=0; j<_sample_points; j++){
            #pragma api vector(512) private(...)
            for(i=0; i<_pixels; i++){
                Convolving/correlating with templates
            }
        }
        #pragma api vector(512) private(...)
        for(i=0; i<_pixels; i++){
            Determining displacement
        }
    }
    ...}
```

Our generic framework
Feature Tracking Code

Code Translation

```c
#define MCPU // or GPU
for(i=0; i<#_frames; i++){
    Read frame
    #pragma api data copyin(frm[0:frm_siz-1])\n        copyin(ini_loc[0:#_smp_pnts.-1])\n        local(con/cor[0:#_pixels]) copyout[fin_loc[0:#._-1]){
        #pragma api compute{
            #pragma api parallel(30) independent
            for(j=0; j<#_sample_points; j++){
                #pragma api vector(512) private(...)
                for(i=0; i<#_pixels; i++){
                    Convolving/correlating with templates
                }
                #pragma api vector(512) private(...)
                for(i=0; i<#_pixels; i++){
                    Determining displacement
                }
            }
        }
    }
}
```

Our generic framework

```c
omp_set_num_threads(30);
for(i=0; i<#_frames; i++){
    Read frame
    #pragma omp parallel for
    for(j=0; j<#_sample_points; j++){
        for(i=0; i<#_pixels; i++){
            Convolving/correlating with templates
        }
        for(i=0; i<#_pixels; i++){
            Determining displacement
        }
    }
}
```

OpenMP

Source translation
Feature Tracking Code

Code Translation

```
#define MCU // or GPU
for(i=0; i<#_frames; i++){  
    Read frame
    #pragma api data copyin(frm[0:frm_siz-1])
    copyin(ini_loc[0:#_smp_pnts.-1])
    local(con/cor[0:#_pixels]) copyout(fin_loc[0:#_.-1])
    #pragma api compute{
        #pragma api parallel(30) independent
        for(j=0; j<#_sample_points; j++){  
            #pragma api vector(512) private(...)
            for(i=0; i<#_pixels; i++){  
                Convolving/correlating with templates
            }
            #pragma api vector(512) private(...)
            for(i=0; i<#_pixels; i++){  
                Determining displacement
            }
            ...
        }
    }
}
```

Our generic framework

```
for(i=0; i<#_frames; i++){  
    Read frame
    omp_set_num_threads(30);
    #pragma omp parallel for
    for(j=0; j<#_sample_points; j++){  
        Convolving/correlating with templates
    }
    for(i=0; i<#_pixels; i++){  
        Determining displacement
    }
    ...
}
```

OpenMP
#define MCPU // or GPU
for(i=0; i<#_frames; i++){  
  Read frame
  #pragma api data copyin(frm[0:frm_siz-1])\  
    copyin(ini_loc[0:#_smp_pnts.-1])\  
    local(con/cor[0:#_pixels]) copyout[fin_loc[0:#_.-1]]{
    #pragma api compute{
      #pragma api parallel(30) independent
      for(j=0; j<#_sample_points; j++){  
        #pragma api vector(512) private(...)  
        for(i=0; i<#_pixels; i++){  
          Convolving/correlating with templates
        }
      #pragma api vector(512) private(...)  
      for(i=0; i<#_pixels; i++){  
        Determining displacement
      }
    }
  }
}
...
Feature Tracking Code

Our generic framework

```c
#define MCPU // or GPU
for(i=0; i<#_frames; i++){
    Read frame
    #pragma api data copyin(frm[0:frm_siz-1])
      copyin(ini_loc[0:#_smp_pnts.-1])
      local(con/cor[0:_pixels]) copyout(fin_loc[0:_..-1])
    #pragma api compute{
        #pragma api parallel(30) independent
        for(j=0; j<#_sample_points; j++){
            #pragma api vector(512) private(...)
            for(i=0; i<#_pixels; i++){
                Convolving/correlating with templates
            }
        }
    }
    for(i=0; i<#_pixels; i++){
        Determining displacement
    }
}

Our generic framework

```
Feature Tracking Code
Workload Mapping

Frame 1
- Convolution
  - SYNC
- Correlation
  - SYNC
- Shifting
  - SYNC
- Reduction
  - SYNC
- Statistical
  - SYNC
- Determine
displacement

Frame 2
- Convolution
  - SYNC
- Correlation
  - SYNC
- Shifting
  - SYNC
- Reduction
  - SYNC
- Statistical
  - SYNC
- Determine
displacement
# Feature Tracking Code Performance

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-core CPU</td>
<td>Original</td>
<td>C</td>
<td>132</td>
<td>117.21</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1-core CPU</td>
<td>Structured</td>
<td>C</td>
<td>124</td>
<td>112.44</td>
<td>1.04</td>
</tr>
<tr>
<td>3</td>
<td>8-core CPU</td>
<td>Structured</td>
<td>OpenMP</td>
<td>138</td>
<td>15.84</td>
<td>7.40</td>
</tr>
<tr>
<td>4</td>
<td>GPU</td>
<td>GPU-specific</td>
<td>CUDA</td>
<td>156</td>
<td>6.54</td>
<td>17.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>294</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1-core CPU</td>
<td>Structured</td>
<td>Our</td>
<td>144</td>
<td>112.44</td>
<td>1.14</td>
</tr>
<tr>
<td>7</td>
<td>8-core CPU</td>
<td>Structured</td>
<td>Our</td>
<td>144</td>
<td>15.84</td>
<td>8.09</td>
</tr>
<tr>
<td>8</td>
<td>GPU</td>
<td>Structured</td>
<td>Our</td>
<td>144</td>
<td>7.21</td>
<td>16.25</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>GPU</td>
<td>Init/Trans Overhead</td>
<td>(CUDA)</td>
<td>---</td>
<td>1.13</td>
<td>---</td>
</tr>
</tbody>
</table>

Our implementation achieves over **50%** reduction in code length compared to OpenMP+CUDA  

GPU performance of our implementation is within **10%** of CUDA
Outline

• Introduction
• Portability Problems
• Approach
• Molecular Dynamics Code
• Feature Tracking Code
• Conclusions
• Future Work
Conclusions

• A common high-level directive-based framework can support efficient execution across architecture types
• Correct use of this framework with the support of the current state-of-the-art parallelizing compilers can yield comparable performance to custom, low-level code
• Our approach results in increased programmability across architectures and decreased code maintenance cost
Outline

- Introduction
- Portability Problems
- Approach
- Molecular Dynamics Code
- Feature Tracking Code
- Conclusions
- Future Work
Future Work

• Test our solution with a larger number of applications

• Propose several annotations that allow:
  – Assignment to particular devices
  – Concurrent execution of loops, each limited to a subset of processing elements

• Extend translator with code analysis and code translation to lower-level language (CUDA) to implement these features for GPU

• Source code for the translator will be released shortly at lava.cs.virginia.edu
Thank you