Exploiting Architectural Capabilities using Higher Semantics

Nabeeh Jumah, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

NEC User Group Meeting
Germany, Kiel
22-05-2019
Introduction

Goals

- Optimal use of architectural features
  - cores, vector units, caches, memory bandwidth, ...
- Portability to different architectures and machines.
- Scalability over multiple nodes.

Challenges

- Needed expertise for optimizing code for hardware features.
- Different approaches and designs in different architectures.
### Different Architectures ... Different Features

#### Broadwell processor
- 18 cores (36 threads); 45 MB shared SmartCache for L3
- Max memory bandwidth is 76.8 GB/s
- Intel(R) AVX2 instruction set extensions
  - Registers of length 256 bits
  - Vector operations are applied with those vector lengths.

#### SX-Aurora vector engine
- 8 cores; 16 MB shared last level cache
- Max memory bandwidth is 1.2 TB/s
- Each register holds 256 entries (64 bits).
- Three FMA pipes per core
  - Each handles 32 double precision FP operations per cycle
Development Using General-Purpose Languages

- The semantical nature of the languages limits the compilers' ability to exploit some optimization opportunities.
- Scientists need to manually optimize code.
- Challenging effort:
  - The complexity of the architectural features
  - The diversity of the architectures
  - Various tools and programming models
- Code optimized for one architecture is suboptimal on another.
- Code quality:
  - Code duplication for different architectures
  - Code maintainability
Advanced Computation and I/O Methods for 
Earth-System Simulations

- Enhance programmability and performance-portability
- Overcome storage limitations
- Shared benchmark for icosahedral models

Funded within the DFG priority programme
Moving to Higher-Level Semantics

Modeling Language Extensibility

- Bypass the shortcomings of the general-purpose languages
- Still use the preferred modeling language
- Extend the modeling language
  - Based on scientific concepts
  - Hiding lower level details (e.g., architecture, memory layout)
- The semantical nature of the extensions allows optimization

Projected Benefits

- Performance-portability
- Code readability and maintainability
- Developers productivity
Application-Level Challenges

Earth system models are representative stencil computations, however, many challenges face developers

- Different modeling approaches
  - Grid structure: regular vs. icosahedral grids
  - Field Localization: staggered vs. collocated grids

- Optimal use of resources is essential to run simulations
  - e.g. memory bandwidth use is a key optimization

a) Triangular grid

b) Hexagonal grid
Approach

Separation of Concerns

- Domain scientists
  - Application source code
  - Scientific perspective
  - Machine-independent (free of machine semantics)

- Scientific programmers
  - Configuration files (guide optimization)
  - Technical perspective
  - Target machine specific

- Higher-level code translation
  - Flexible tools use configuration information to transform high-level code into optimized code
Higher-Level Coding with GGDML

GGDML

- **GGDML**: *General Grid Definition and Manipulation Language*
- Grid definition
- Field declaration
- Field data access/update
  - Iterators
  - Access operators
- Stencil operations

*GGDML: Icosahedral Models Language Extensions (Nabeeh Jumah et. al)*

*DOI: 10.15379/2410-2938.2017.04.01.01*

- Hides memory locations and access details
- Hides connectivity and grid structure
GGDML Code Example

```c
foreach c in grid {
    float df = (f_F[c.east_edge()] - f_F[c.west_edge()]) / dx;
    float dg = (f_G[c.north_edge()] - f_G[c.south_edge()]) / dy;
    f_HT[c] = df + dg;
}

Sample generated C code: ________________________________________________

... handle domain decomposition and halo management
for (size_t blk_start = (0); ... blocking
    size_t blk_end = ...
    #pragma omp parallel for
    for (size_t YD_index = 0; YD_index < local_Y_Cregion; YD_index++) {
        #pragma omp simd
        for (size_t XD_index = blk_start; XD_index < blk_end; XD_index++) {
            float df = (f_F[YD_index][XD_index +1] -
                        f_F[YD_index][XD_index]) / dx;
            float dg = (f_G[YD_index +1][XD_index] -
                        f_G[YD_index][XD_index]) / dy;
            f_HT[YD_index][XD_index] = df + dg;
        }
    }
```
Translation Configurations

- Scientific programmers
  - Define language extensions
    - e.g. define access operators
      right(): XD=$XD+1
      => Allows access to the neighboring cell to the right
  - Control optimization procedures
- Different options allow to exploit hardware
  - Memory layout & abstract index translation, loop order, parallelization, blocking, vector units
- Configurations define grids and how they will be processed
  - Problem domain
  - Grids relationships and connectivity
    - Simplifies specifying stencils
    - Very beneficial for unstructured grids
  - Halo patterns under multi-node runs
Translation Process

Higher-level code translation

- A source-to-source translation tool is used
  - A lightweight tool
  - Easily ships with code repositories
  - Simply fits within build procedures, e.g. make
- Optimization procedures are applied during translation
  - to exploit features of target-machine

Translation Process Drivers

- The semantics of the language extensions
  - Extracted from the source code
- Configuration information
Translation Process

Performance Portability of Earth System Models with User-Controlled GGDM code Translation (Jumah & Kunkel)
DOI: 10.1007/978-3-030-02465-9_50
Some experimental results:
Vectorization and Memory Throughput

- **Multi-core experiments environment**
  - Dual socket Broadwell nodes
  - Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz
  - Intel C compiler (ICC 17.0.5 20170817)

- **Vector engine experiments environment**
  - NEC SX-Aurora TSUBASA vector engine
  - NEC NCC (1.3.0) C compiler

- **Measurement tools**
  - Likwid on Broadwell
  - Ftrace on Aurora

Experiments done and published under: Automatic Vectorization of Stencil Codes with the GGDML Language Extensions (Jumah & Kunkel) DOI: http://doi.acm.org/10.1145/3303117.3306160
Vectorization and Memory Throughput

- Test code
  - Shallow water equations
  - Structured grid
  - Explicit time stepping scheme
  - Finite difference method
  - Eight kernels
    - Flux components
    - Tendencies of the two velocity components
    - Surface level tendency
    - Velocity components
    - Surface level

- Tested configurations
  - Contiguous unit stride arrays
  - AoS emulation: Constant short distance (4 byte distance) separating consecutive elements
  - Scattered (distant) data elements
Results on Broadwell Multi-core Processor

- Max memory bandwidth is 76.8 GB/s
- Achieved throughput around 62 GB/s (~80% of max.)
- Unit stride code is performing well taking into account the arithmetic intensity, all kernels are completely vectorized
- In constant short distance version some kernels are vectorized
- In scattered data version code is not vectorized

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Time (s)</th>
<th>AVX GFLOPS</th>
<th>Time (s)</th>
<th>AVX GFLOPS</th>
<th>Time (s)</th>
<th>AVX GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>flux1</td>
<td>250</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>27</td>
<td>11</td>
</tr>
<tr>
<td>flux2</td>
<td>248</td>
<td>0</td>
<td>54</td>
<td>0</td>
<td>27</td>
<td>11</td>
</tr>
<tr>
<td>compute_U_tendency</td>
<td>431</td>
<td>0</td>
<td>80</td>
<td>21</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>update_U</td>
<td>158</td>
<td>0</td>
<td>39</td>
<td>0</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>compute_V_tendency</td>
<td>432</td>
<td>0</td>
<td>94</td>
<td>18</td>
<td>47</td>
<td>37</td>
</tr>
<tr>
<td>update_V</td>
<td>158</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>compute_H_tendency</td>
<td>251</td>
<td>0</td>
<td>55</td>
<td>0</td>
<td>28</td>
<td>11</td>
</tr>
<tr>
<td>update_H</td>
<td>158</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Application Level</td>
<td>2,103</td>
<td>3</td>
<td>466</td>
<td>13</td>
<td>244</td>
<td>25</td>
</tr>
</tbody>
</table>

Automatic Vectorization of Stencil Codes with the GGDML Language Extensions (Jumah & Kunkel)
DOI: http://doi.acm.org/10.1145/3303117.3306160
Introduction

Higher-Level Language Extensions

Experiments

Conclusion

Results on Aurora Vector Engine

- Max memory bandwidth is 1.2 TB/s
- Achieved throughput around 960 GB/s (~80% of max.)
- Unit stride code is performing well taking into account the arithmetic intensity, all kernels are optimally vectorized
- In the other code versions the vector units still work, but as efficiently as the unit stride code version

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Scattered</th>
<th>Constant short distance</th>
<th>Contiguous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td>GFLOPS</td>
<td>Time (s)</td>
</tr>
<tr>
<td>flux1</td>
<td>5.37</td>
<td>56</td>
<td>3.96</td>
</tr>
<tr>
<td>flux2</td>
<td>5.36</td>
<td>56</td>
<td>4.08</td>
</tr>
<tr>
<td>compute_U_tendency</td>
<td>20.67</td>
<td>92</td>
<td>8.26</td>
</tr>
<tr>
<td>update_U</td>
<td>3.82</td>
<td>52</td>
<td>2.44</td>
</tr>
<tr>
<td>compute_V_tendency</td>
<td>20.66</td>
<td>97</td>
<td>9.12</td>
</tr>
<tr>
<td>update_V</td>
<td>3.82</td>
<td>52</td>
<td>2.43</td>
</tr>
<tr>
<td>compute_H_tendency</td>
<td>6.88</td>
<td>73</td>
<td>4.26</td>
</tr>
<tr>
<td>update_H</td>
<td>3.82</td>
<td>52</td>
<td>2.44</td>
</tr>
<tr>
<td>Application level</td>
<td>70.40</td>
<td>80</td>
<td>37.17</td>
</tr>
</tbody>
</table>

Automatic Vectorization of Stencil Codes with the GGDML Language Extensions (Jumah & Kunkel)
DOI: http://doi.acm.org/10.1145/3303117.3306160

Nabeeh Jumah
Universität Hamburg
Inter-Kernel Optimization

Further experiments were done to improve data reuse

- Using loop fusions
- Reducing data loading from memory
- Loading a field once from memory for multiple stencils
- Tuning per architecture is important
  - For optimal use of caching
- Results show better use of memory bandwidth
- Performance ratios reflect architecture memory bandwidths

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Memory bandwidth (GB/s)</th>
<th>Before merge</th>
<th>GFLOPS</th>
<th>After merge</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Measured memory throughput (GB/s)</td>
<td></td>
<td>Measured memory throughput (GB/s)</td>
<td></td>
</tr>
<tr>
<td>Broadwell</td>
<td>77</td>
<td>62</td>
<td>24</td>
<td>60</td>
<td>31</td>
</tr>
<tr>
<td>P100 GPU</td>
<td>500</td>
<td>380</td>
<td>149</td>
<td>389</td>
<td>221</td>
</tr>
<tr>
<td>NEC Aurora</td>
<td>1,200</td>
<td>961</td>
<td>322</td>
<td>911</td>
<td>453</td>
</tr>
</tbody>
</table>
Conclusion

- Model development is improved with GGDML semantics
  - Performance portability, code quality, productivity
- A single code can be used for all architectures
- Architecture resources can still be used
  - Optimal use of memory bandwidth of an architecture is the key to maximize performance
- Instead of source code, configuration files guide optimization
Acknowledgement

- DFG (German Research Foundation)
- Erlangen regional computing center (RRZE) at Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
- NEC Deutschland