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

Project AIMES

Advanced Computation and I/O Methods for Earth-System Simulations

AIMES

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

Funded within the DFG priority programme SPPEXA

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

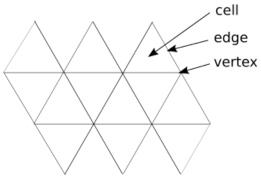
Projected Benefits

- Performance-portability
- Code readability and maintainability
- Developers productivity

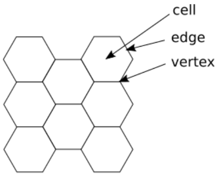
Application-Level Challenges

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**:***G**eneral **G**rid **D**efinition and **M**anipulation **L**anguage*
- 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

```
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 mangagement
  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 benefecial for unstructured grids
  - Halo patterns under multi-node runs

Translation Process

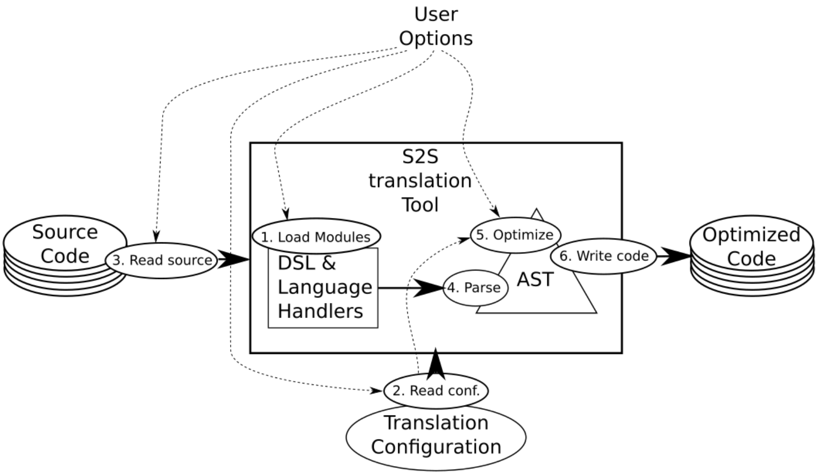
Higher-level code translation

- 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 GGDML 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) seperating 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

Kernel	Scattered		Constant short distance		Contiguous	
	Time (s)	AVX GFLOPS	Time (s)	AVX GFLOPS	Time (s)	AVX GFLOPS
flux1	250	0	52	0	27	11
flux2	248	0	54	0	27	11
compute_U_tendency	431	0	80	21	41	41
update_U	158	0	39	0	20	10
compute_V_tendency	432	0	94	18	47	37
update_V	158	0	40	0	20	10
compute_H_tendency	251	0	55	0	28	11
update_H	158	0	40	0	20	10
Application Level	2,103	3	466	13	244	25

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

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 effeciently as the unit stride code version

	Scattered		Constant short distance		Contiguous	
Kernel	Time (s)	GFLOPS	Time (s)	GFLOPS	Time (s)	GFLOPS
flux1	5.37	56	3.96	76	1.30	230
flux2	5.36	56	4.08	74	1.51	199
compute_U_tendency	20.67	92	8.26	230	5.29	359
update_U	3.82	52	2.44	82	1.21	166
compute_V_tendency	20.66	97	9.12	220	5.22	384
update_V	3.82	52	2.43	82	1.21	165
compute_H_tendency	6.88	73	4.26	117	1.52	330
update_H	3.82	52	2.44	82	1.20	167
Application level	70.40	80	37.17	161	18.63	322

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

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

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

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