Automatic Vectorization of Stencil Codes with the GGDML Language Extensions

Nabeeh Jum'ah, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

WPMVP 2019 Workshop on Programming Models for SIMD/Vector Processing
Washington DC, USA 16-02-2019

Goals

- Enable unified high-level source code of earth system models (e.g. climate and numerical weather prediction)
- Optimally use vector units & memory bandwidth besides to other hardware features of the different architectures.

Project AIMES

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

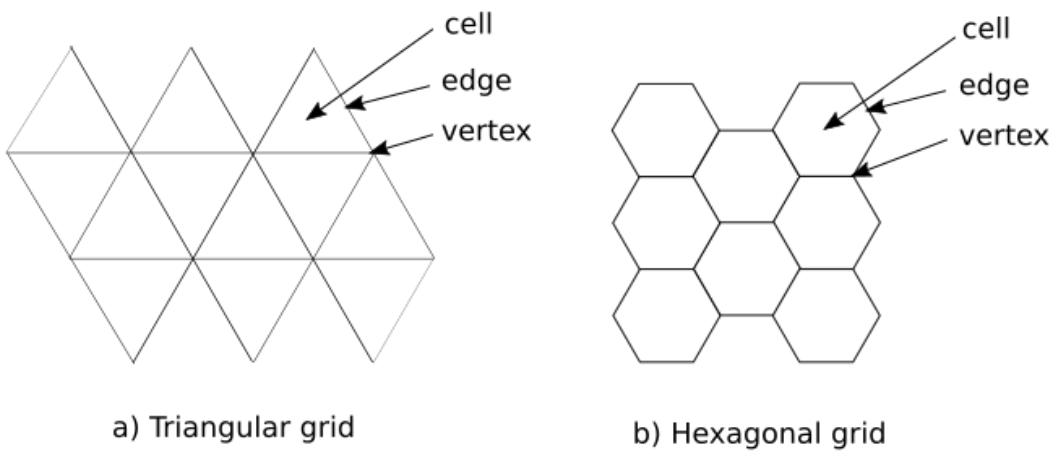
- Enhance programmability and performance-portability
- Overcome storage limitations
- Shared benchmark for icosahedral models

Architecture-dependant Features

- Architecture features drive code optimization
- Broadwell processor
  - 18 cores (36 threads)
  - 45 MB shared SmartCache for L3 caching
  - 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

Earth-System Modeling

- Earth system models are representative stencil computations
- Optimal use of resources is essential to run simulations
  - Structure of stencil codes fits vectorization
  - Code should be written to exploit vector units
  - Memory access optimization is a key to optimal code
- Grid choice, e.g. regular/icosahedral, affects stencil definition
- Stencils could access cell centers, edges, vertices ...



a) Triangular grid

b) Hexagonal grid

Model Development

Modeling 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
  - Model's maintainability

Model Development

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

Approach

Separation of Concerns

- Domain scientists formulate scientific logic in source code
- Scientific programmers write target configurations

- Model development with extended language
  - Scientific perspective
    - not machine perspective
  - Code is developed once
    - performance is achieved for different configurations
- Configurations define software performance
  - A configuration corresponds to a target run environment
  - Written by programmers with more experience in platform

Approach

- 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 semantical nature of the language extensions
  - Extracted from the source code
- Configuration information

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, data iteration
- Abstract higher concepts of grids, hiding connectivity details

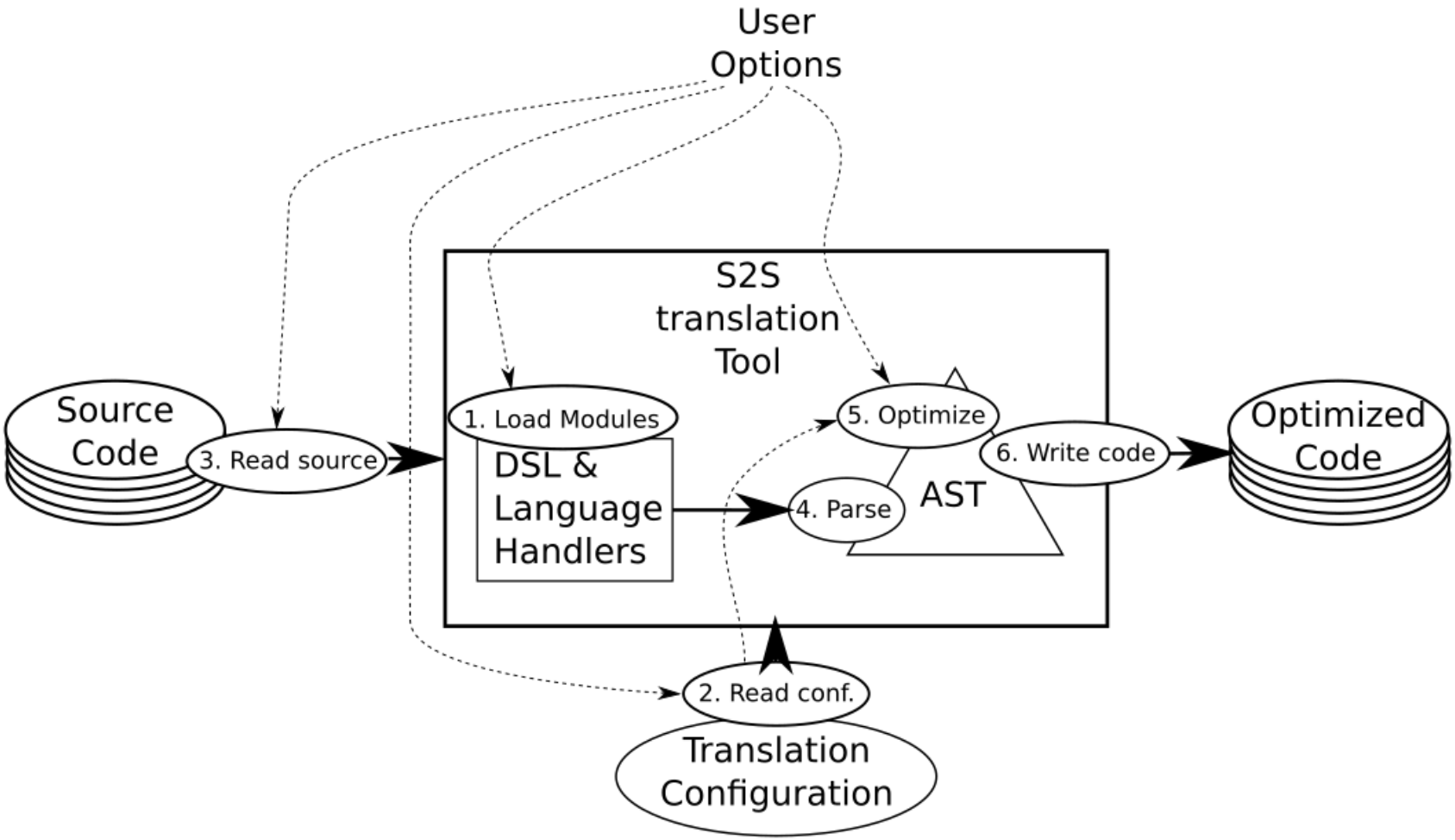
GGDML Code Example

```
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 process



Performance Portability of Earth System Models with User-Controlled GGDML code Translation (Jumah & Kunkel)-DOI: 10.1007/978-3-030-02465-9_50

Translation Configurations

- Loop optimization is essential for
  - Optimal memory access
  - Applicability of vector operations
- Suitable transformations are applied
  - Memory layout & abstract index translation
  - Loop order
  - Parallelization

Translation Configurations

Memory Layout

- Controlled by users
  - Memory allocation
  - Array Indices
- The translation tool generates the needed memory layout of a field based on
  - The semantical information used to declare a field
  - The user-provided memory allocation configuration
- The indices are completely controlled by the user
  - Index reordering
  - More complicated formulae to apply mathematical transformations, e.g. a filling curve
- Loop order should match memory layout

Translation Configurations

Access operators

- The user defines
  - The syntax
  - The behavior
- Define grids relationships and connectivity
  - Simplify references to neighborhoods
  - Abstracts the machine notion of array indices with domain concepts, e.g. above, below, neighbor, right, edge...
- Example definition:
  - above(): height=$height+1
  - =>Allows access to the element directly above the current
- Improves semantics & code quality

Translation Configurations

Parallelization

- Parallelizaion should allow optimal vectorization
- Parallelizaion on different architectures tested
  - multi-core processors (using OpenMP)
  - GPUs (using OpenACC)
  - Multiplr-node MPI(+OpenMP/OpenACC)
- The parallelization on multiple-node configurations is possible
  - The user controls the communication library initialization
  - The user controls the halo exchange code
- The translation tool uses the semantics of the field access to generate the halo exchange code

Experiments

- 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

Experiments

- 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

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

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

Conclusion

- GGDML improves model development
  - Performance portability, code quality, productivity
- Memory layout is an important factor for vectoriztion
- Memory bandwidth use efficiency is also affected
- Loop structure should match the memory layout
  - Scheduling, caching, loop order
- Bad memory layouts harm seriously the performance
- The tools allowed estimating AoS performance (and vectorization) without really implementing it
- Some applications need both unit-stride and distant element access, this study allowed better understanding of performance

Future Work

- Explore vectorization with inter-kernel optimization
- Explore automatic optimal memory layout detection for applications with different access patterns

Acknowledgement

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