Optimizing Memory Bandwidth Efficiency with User-Preferred Kernel Merge

Nabeeh Jumah, Julian Kunkel

Scientific Computing
Department of Informatics
University of Hamburg

COLOC: 3rd workshop on data locality/Euro-Par 2019
Göttingen, Germany
26-08-2019

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

Application Domain

Earth system modeling

- Models apply numerical methods to simulate system
  - Hundreds or thousands of stencils are executed
  - A sequence of stencils is applied each time step
- A stencil has a pattern of data elements
  - Low arithmetic intensity
  - Memory bound
- Each stencil should be optimized
  - Efficient use of memory bandwidth is critical
  - Memory access should be minimized
  - Caches are a key consideration

Optimization

Stencil Optimization

- Stencils comprise multiple grid points
- Field data is accessed multiple times

```
a[i][j] =  c[i][j] * 0.6 + 0.1 *
    (c[i-1][j] + c[i+1][j] + c[i][j-1] + c[i][j+1]);
```

- Neighboring points => data locality
- Exploiting data locality reduces memory access

Inter-kernel Optimization

- Optimal stencil code is not all what can be done
- Optimization across kernels improves application performance

Inter-kernel Optimization

- A sequence of stencils is subject to optimization
  - A set of fields are accessed
  - Some fields are updated
  - Computation consistency is a main objective
    - Stencil precedence is an issue
    - Data dependencies should be held with any transformation

```
a[i][j] = (c[i][j] + c[i  ][j-1]) /2.0;

b[i][j] = (c[i][j] + c[i-1][j  ]) /2.0;
```

- Solutions have been applied with loop fusions
  - Manually by scientists
    - Additional effort by scientists
    - Complicated loops and code structure
  - Automatic by tools

Our Approach

- Use GGDML to write code
- Use tools to translate GGDML code and apply optimization
- Exploit inter-kernel optimization opportunities
- Allow scientists to control the process
- Automatize the time consuming and complicated parts
  - Tools analyze code
  - Prepare a list of possible fusions
  - Apply fusions selected by scientists
- Maximize possibilities by inter-module optimization
  - Calls are analyzed across code files by tools
  - A list of possible call inlinings is prepared
  - Tools inline calls selected by scientists

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

An Example GGDML Code

```
foreach e in grid {

    f_F[e] = f_U[e] * (f_H[e.east_cell()] +
                       f_H[e.west_cell()]) / 2.0;
}

foreach e in grid {

    f_G[e] = f_V[e] * (f_H[e.north_cell()] +
                       f_H[e.south_cell()]) / 2.0;
}
```

Resulting Transformations

```
for (size_t blk_start = (0); blk_start < (GRIDX + 1);
    blk_start += 20000) {
...
#pragma omp parallel for num_threads(36)
  for (size_t YD_index = (0); YD_index < (local_Y_Eregion);
      YD_index++) {
#pragma omp simd
    for (size_t XD_index = blk_start; XD_index < blk_end;
        XD_index++) {
      f_F[YD_index][XD_index] =
          f_U[YD_index   ][XD_index   ] * (
          f_H[YD_index   ][XD_index   ] +
          f_H[YD_index   ][XD_index -1]) /2.0;
      f_G[YD_index][XD_index] =
          f_V[YD_index   ][XD_index   ] * (
          f_H[YD_index   ][XD_index   ] +
          f_H[YD_index -1][XD_index   ]) /2.0;
    }
  }
}
```

Experiments

Multi-core experiments environment

- Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz
- Intel C compiler (ICC 17.0.5)

GPU experiments environment

- Tesla P100 GPUs, 16 GB memory, PCIe interconnect
- PGI (17.7.0) C compiler

Vector engine experiments environment

- SX-Aurora TSUBASA
- NCC (1.3.0) C compiler

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

Experiments on Broadwell

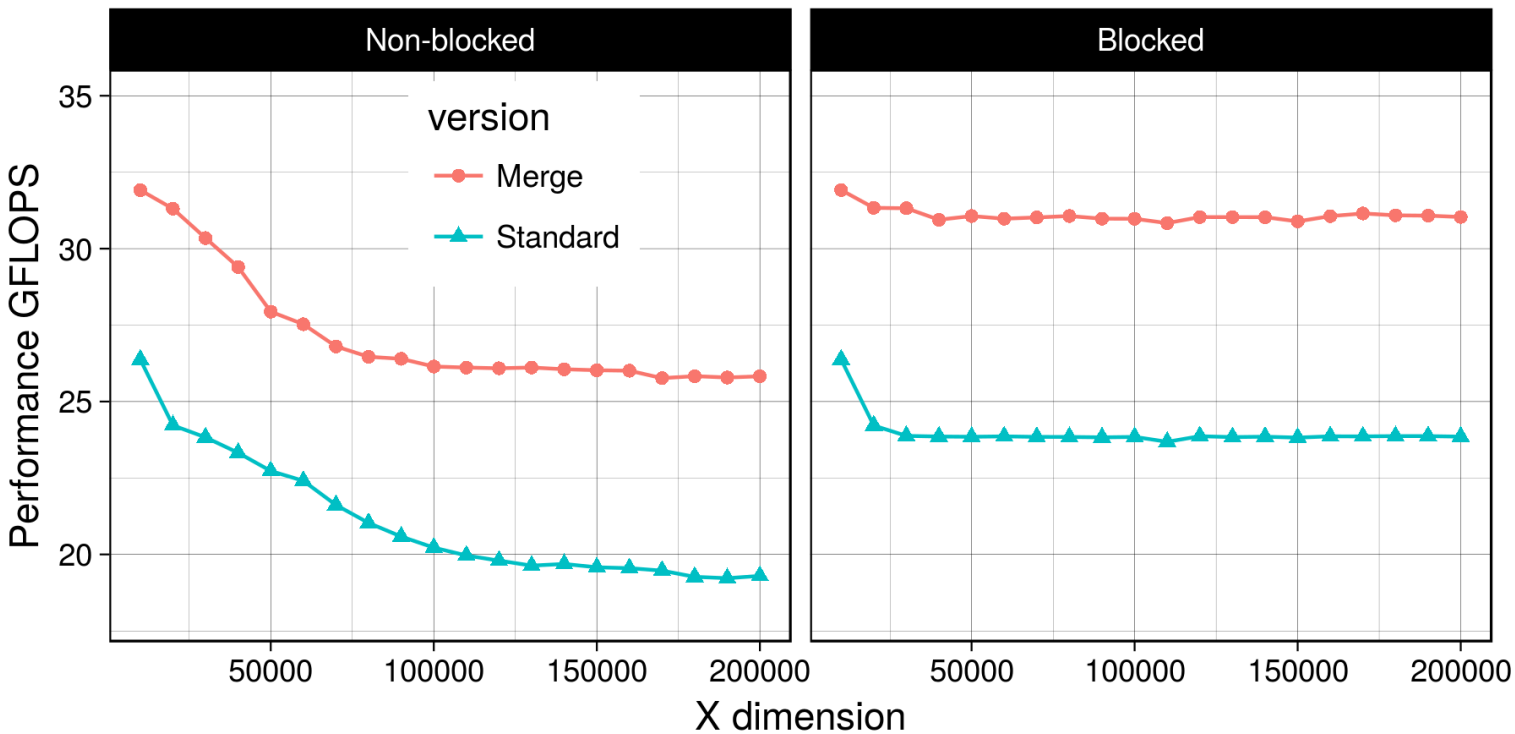

Figure: Variable grid width with/o blocking on Broadwell

Experiments on Broadwell

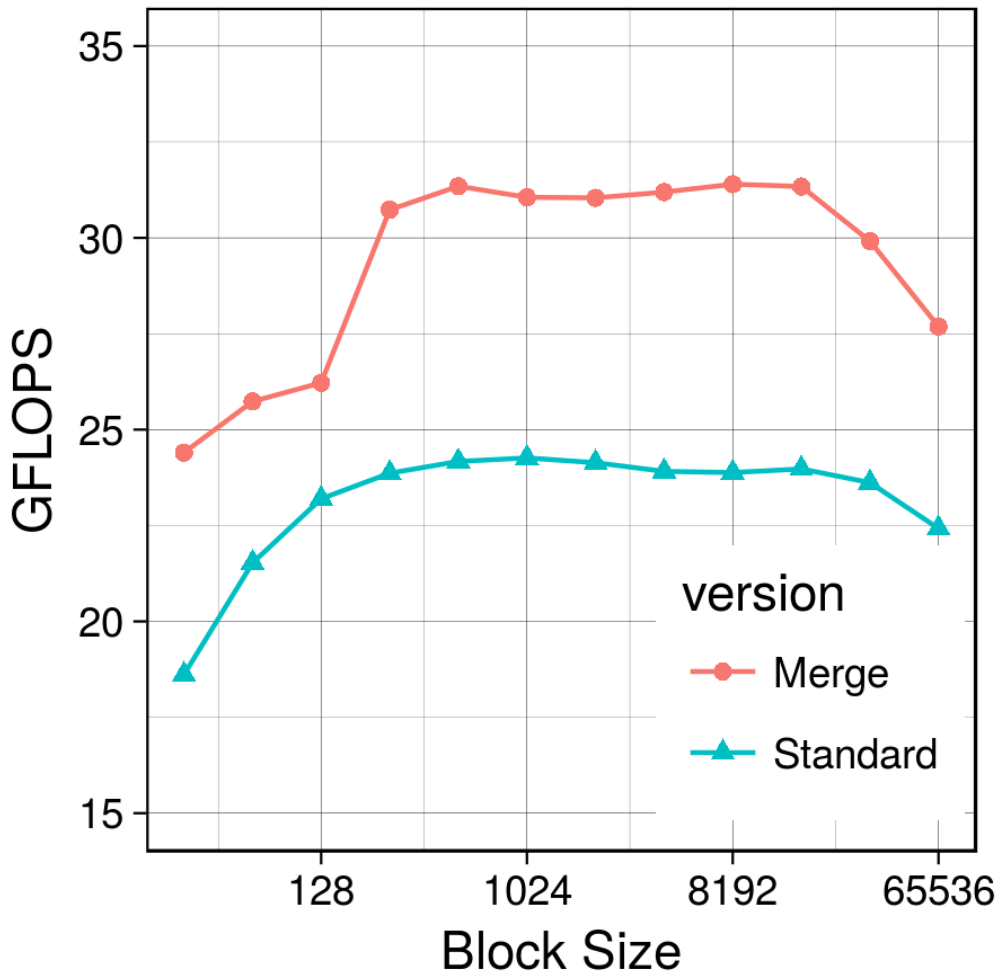

Figure: Different block sizes on Broadwell

Experiments on Broadwell

Kernel	Time (s)	GFLOPS	Memory Bandwidth (GB/s)
flux1	26.9	11.2	59.8
flux2	26.6	11.3	62.8
compute_U_tendency	41.3	41.2	62.3
update_U	19.5	10.3	62.8
compute_V_tendency	46.4	36.7	61.8
update_V	19.3	10.3	63.3
compute_H_tendency	26.6	11.3	62.9
update_H	19.8	10.1	62.4
Standard_code	226.3	23.8	62.2
flux_and_tendencies	96.9	41.3	59.5
velocities	39.6	10.1	61.3
compute_surface	40.6	12.3	60.7
Merged_code	177.0	31.0	60.2

Table: Likwid profiles on Broadwell for all kernels and both code versions

Experiments on P100 GPUs

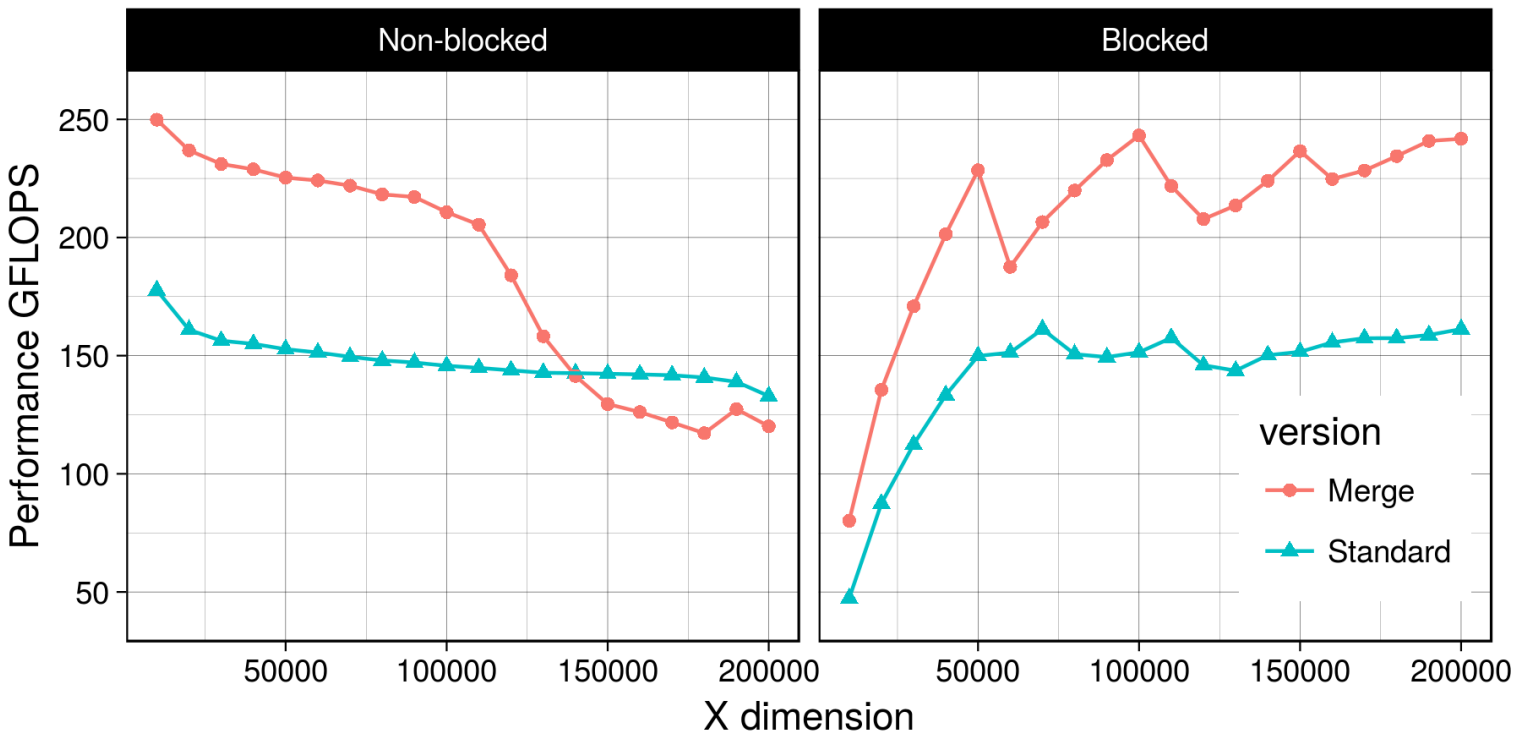

Figure: Different grid widths on P100 GPU

Experiments on P100 GPUs

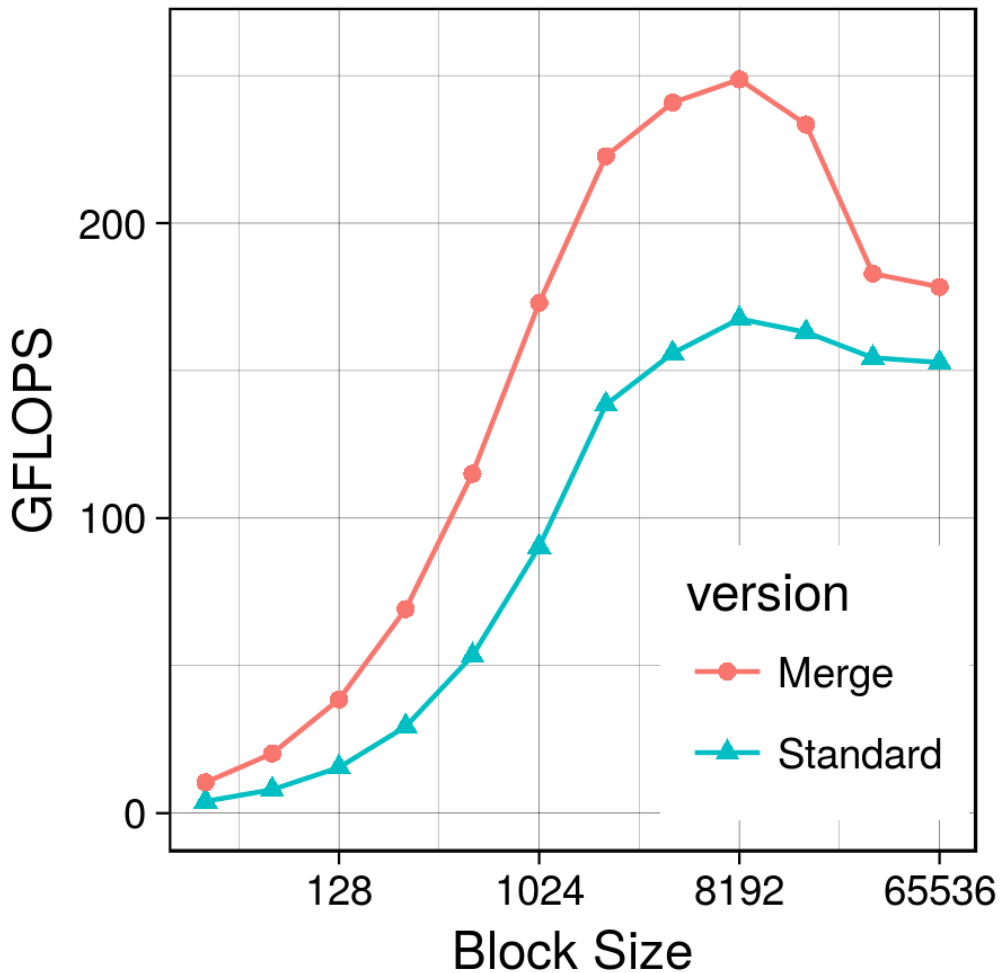

Figure: Different block sizes on P100 GPU

Experiments on P100 GPUs

Kernel	Memory Throughput (GB/s)	Data Volume (GB)	Kernel Time (s)	GFLOPS
flux1	447	1,175	2.63	114
flux2	478	1,570	3.29	91
compute_u_tendency	358	3,338	9.33	225
update_u	376	1,126	2.99	67
compute_v_tendency	374	4,195	11.22	196
update_v	376	1,126	3.00	67
compute_h_tendency	333	1,588	4.77	105
update_h	387	1,126	2.91	69
Standard_code	380	15,244	40.13	149
flux_and_tendencies	396	5,970	15.08	325
velocities	360	2,268	6.31	63
compute_surface	403	2,303	5.71	123
Merged_code	389	10,542	27.11	221

Table: Kernels measurements in both code versions on P100 GPU

Experiments on Aurora Tsubasa vector engine

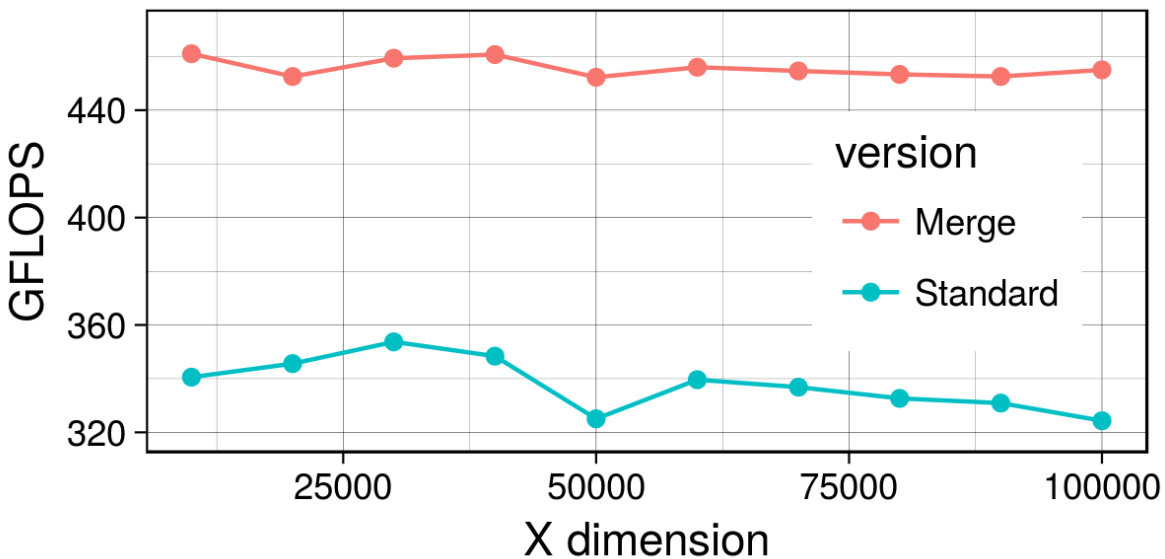

Figure: Different grid widths on NEC Aurora vector engine

Experiments on Aurora Tsubasa vector engine

Kernel	Time (s)	GFLOPS	Memory Throughput (GB/s)
flux1	1.30	230	858
flux2	1.51	199	989
compute_U_tendency	5.29	359	986
update_U	1.21	166	927
compute_V_tendency	5.22	384	1,001
update_V	1.21	165	924
compute_H_tendency	1.52	330	984
update_H	1.20	167	934
Standard_code	18.63	322	961
flux_and_tendencies	8.40	500	911
velocities	2.43	165	922
compute_surface	2.31	303	940
Merged_code	13.25	453	911

Table: Kernel measurements of both code versions on the NEC Aurora

Conclusion

- Stencil-level optimization allows to improve use of hardware
- Optimization across stencils improves performance
  - Resources are used more efficiently
  - Achieved improvement about 30-48% for different architectures
- GGDML and its tool enable inter-kernel optimization
- Developers can control inter-kernel optimization
  - Tools handle lengthy and complex tasks of analysis
  - Users choose optimizations
  - Tools apply the optimizations
- Analysis is done across files
  - Allows inlining calls across code files
  - Uncovers further fusion opportunities

Acknowledgement

- DFG (German Research Foundation)
- Swiss National Supercomputing Center (CSCS)
- Erlangen regional computing center (RRZE) at Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
- NEC Deutschland
- Prof. John Thuburn, University of Exeter