MODESTO: Data-centric Analytic Optimization of Complex Stencil Programs on Heterogeneous Architectures

most work performed by TOBIAS GYSI AND TOBIAS GROSSER
Stencil computations (oh no, another stencil talk)

due to their low arithmetic intensity, stencil computations are typically heavily memory bandwidth limited!

Motivation:
- Important algorithmic motif (e.g., finite difference method)

Definition:
- Element-wise computation on a regular grid using a fixed neighborhood
- Typically working on multiple input fields and writing a single output field

$$\text{lap}(i,j) = -4.0 \times \text{in}(i,j) + \text{in}(i-1,j) + \text{in}(i+1,j) + \text{in}(i,j-1) + \text{in}(i,j+1)$$
How to tune such stencils (most other stencil talks)

- LOTS of related work!
  - Compiler-based (e.g., Polyhedral such as PLUTO [1])
  - Auto-tuning (e.g., PATUS [2])
  - Manual model-based tuning (e.g., Datta et al. [3])
  - Saday’s tricks from his talk after lunch 😊
  - ... essentially every micro-benchmark or tutorial, e.g.:

- Common features
  - Vectorization tricks (data layout)
  - Advanced communication (e.g., MPI neighbor colls)
  - Tiling in time, space (diamond etc.)

- Much of that work DOES NOT compose well with practical complex stencil programs

---

[1]: Uday Bondhugula, A. Hartono, J. Ramanujan, P. Sadayappan. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer , PLDI’08
[3]: Kaushik Datta, et al., Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors, SIAM review
What is a “complex stencil program”? (this stencil talk)

E.g., the COSMO weather code

- is a regional climate model used by 7 national weather services
- contains hundreds of different complex stencils

Modeling stencils formally:

- Represent stencils as DAGs
  - Model stencil as nodes, data dependencies as edges

simplified horizontal diffusion example

\[ a \oplus b = \{ a' + b' | a' \in a, b' \in b \} \]
Data-locality Transformations

- Consider the horizontal diffusion lap-fli-out dependency chain (i-dimension)

Loop Tiling & Loop Fusion
How to Deal with Data Dependencies?

- Consider the horizontal diffusion lap-fli-out dependency chain (i-dimension)

Halo Exchange Parallel (hp):
- Update tiles in parallel
- Perform halo exchange communication

Pros and Cons:
- Avoid redundant computation
- At the cost of additional synchronization
How to Deal with Data Dependencies?

- Consider the horizontal diffusion lap-fli-out dependency chain (i-dimension)

Halo Exchange Sequential (hs):
- Update tiles sequentially
- Innermost loop updates tile-by-tile

Pros and Cons:
- Avoid redundant computation
- At cost of being sequential
How to Deal with Data Dependencies?

- Consider the horizontal diffusion lap-fli-out dependency chain (i-dimension)

Computation on-the-fly (of):
- Compute all dependencies on-the-fly
- Overlapped tiling

Pros and Cons:
- Avoid synchronization
- At the cost of redundant computation
Hierarchical Tiling

- By tiling the domain repeatedly we target multiple memory hierarchy levels
Case Study: STELLA (STEncil Loop LAnguage)

- STELLA is a C++ stencil DS(e)L of COSMO’s dynamical core (50k LOC, 60% RT)

```cpp
// define stencil functors
struct Lap { ... };
struct Fli { ... };
...
// stencil assembly
Stencil stencil;
StencilCompiler::Build(
    stencil,
    pack_parameters( ... ),
    define_temporaries(
        StencilBuffer<lap, double>(),
        StencilBuffer<fli, double>(),
        ... ),
    define_loops(
        define_sweep(
            StencilStage<Lap, IJRange<-1,1,-1,1> >(),
            StencilStage<Fli, IJRange<-1,0,0,0> >(),
            ...
        )));
// stencil execution
stencil.Apply();
```

using C++ template metaprogramming:

STEELA defines a virtual tiling hierarchy that facilitates platform independent code generation
# Tiling Hierarchy of STELLA’s GPU-Backend

<table>
<thead>
<tr>
<th>DSL</th>
<th>Tile Size</th>
<th>Strategy</th>
<th>Memory</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>sweep</td>
<td>1 x 1 x 1</td>
<td>halo exchange parallel</td>
<td>registers</td>
<td>scratchpad</td>
</tr>
<tr>
<td>sweep</td>
<td>∞ x ∞ x 1</td>
<td>halo exchange sequential</td>
<td>registers</td>
<td>registers</td>
</tr>
<tr>
<td>loop</td>
<td>64x4x64</td>
<td>computation on-the-fly</td>
<td>GDDR</td>
<td>-</td>
</tr>
<tr>
<td>stencil</td>
<td>∞ x ∞ x ∞</td>
<td>computation on-the-fly</td>
<td>GDDR</td>
<td>-</td>
</tr>
</tbody>
</table>
Stencil Program Algebra

- Map stencils to the tiling hierarchy using a bracket expression

- Enumerate the stencil execution orders that respect the dependencies

- Enumerate implementation variants by adding/removing brackets

... lap, fli, flj, out, ...

[[lap, fli, flj], [out]]
Our model considers peak computation and communication throughputs.

### Machine Performance Model

- **target machine**
  - core 1 (30 Gflop)
  - core 2 (30 Gflop)
  - core 3 (30 Gflop)
  - cache (256 kB)
  - cache (256 kB)
  - cache (256 kB)

- **machine model**
  - $C = 90$ Gflops
  - $V^1 = 300$ GB/s
  - $L^1 = 50$ GB/s
  - $M^1 = 256$ kB

### Communication Throughputs
- Lateral and vertical communication refer to communication within one respectively between different tiling hierarchy levels.

- Lateral communication:
  - 100 GB/s between cores
  - 25 GB/s between caches

- Vertical communication:
  - 10 GB/s to DDR
  - 100 GB/s to cache
  - 100 GB/s to core

- DDR:
  - 25 GB/s to cache
  - 10 GB/s to machine

- Machine:
  - 8 GB DDR
  - 256 kB cache
Stencil Performance Model - Overview

- Given a stencil $s$ given and the amount of computation $c_s$
  \[ t_s = \frac{c_s}{C} \]

- Given a group $g$ and the vertical and lateral communication $v_c$ and $l^1_c, ..., l^m_c$
  \[ t_g = \sum_{c \in g.\text{child}} \max(t_c, v_c/V^m, l^1_c/L^1, ..., l^m_c/L^m) \]
Stencil Performance Model - Affine Sets and Maps

- The stencil program analysis is based on (quasi-) affine sets and maps
  \[ S = \{ \vec{i} | \vec{i} \in \mathbb{Z}^n \land (0, ..., 0) < \vec{i} < (10, ..., 10) \} \]
  \[ M = \{ \vec{i} \rightarrow \vec{j} | \vec{i} \in \mathbb{Z}^n, \vec{j} \in \mathbb{Z}^n \land \vec{j} = 2 \cdot \vec{i} \} \]
- For example, data dependencies can be expressed using named maps
  \[ D_{fli} = \{ (fli, \vec{i}) \rightarrow (lap, \vec{i} + \vec{j}) | \vec{i} \in \mathbb{Z}^2, \vec{j} \in \{(0,0), (1,0)\} \} \]

\[ D = D_{lap} \cup D_{fli} \cup D_{flj} \cup D_{out} \]
\[ E = D^{+}(\{(out, \vec{0})\}) \]

apply the out origin vector to the transitive closure of all dependencies
Stencil Performance Model - Tiling Transformations

- Define a tiling using a map that associates stencil evaluations to tile ids

\[ T_{out} = \{(\text{out}, (i_0, i_1)) \rightarrow ([i_0/2], [i_1/2])\} \]
Stencil Performance Model – Comp & Comm

- Count floating point operations necessary to update tile (0,0)
  \[ c_{out} = |T_{out} \cap ran \{(0,0)\}| \cdot \#flops \]

- Count the number of loads necessary to update tile (0,0)
  \[ l_{out} = |(T_{out} \circ D_{out}^{-1}) \cap ran \{(0,0)\}| \]
Analytic Stencil Program Optimization

- **Put it all together (stencil algebra, performance model, stencil analysis)**
  1. Optimize the stencil execution order (brute force search)
  2. Optimize the stencil grouping (dynamic programming / brute force search)

\[
\minimize_{x \in I} t(x)
\]

subject to \( m(x) \leq M \)
Evaluation

**CPU Experiments (i5-3330):**

- no fusion
- hand-tuned
- optimized

**GPU Experiments (Tesla K20c):**

- no fusion
- hand-tuned
- optimized

**Graphs:**

- HD
- UV
- DIV
- UV&DIV

**Equations:**

- $m = 1.6e$
- $m = 1.5e$
Not just your basic, average, everyday, ordinary, run-of-the-mill, ho-hum stencil optimizer

- **Complete performance models for:**
  - Computation (very simple)
  - Communication (somewhat tricky, using sets and Minkowski sums, parts of the PM)
- **Established a stencil algebra**
  - Complete enumeration of all program variants
- **Navigate the performance space analytically**
  - Find the best program variant for a given system
  
  *Very different for CPU and GPU!*
- **Automatic tuning of stencil programs (using the STELLA DS(e)L)**
  - 2.0-3.1x speedup against naive implementations
  - 1.0-1.8x speedup against expert tuned implementations
Backup Slides