TORSTEN HOEFLER

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

most work performed by TOBIAS GYSI AND TOBIAS GROSSER

Stencil computations (oh

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

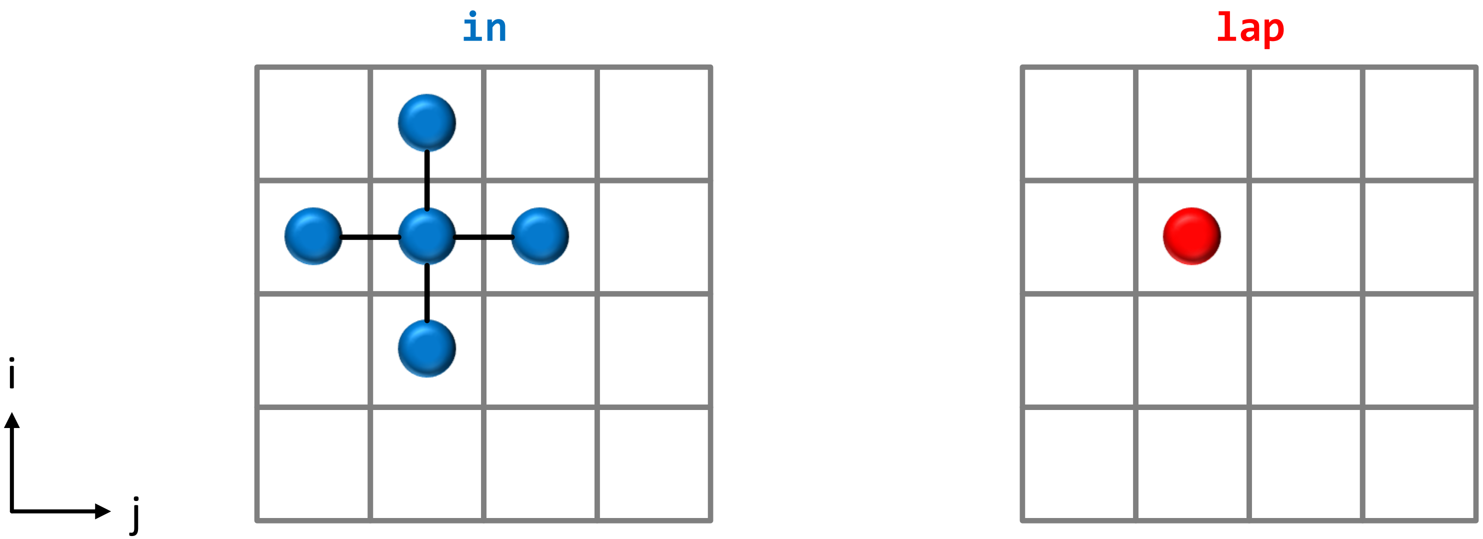
Motivation:

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

Definition:

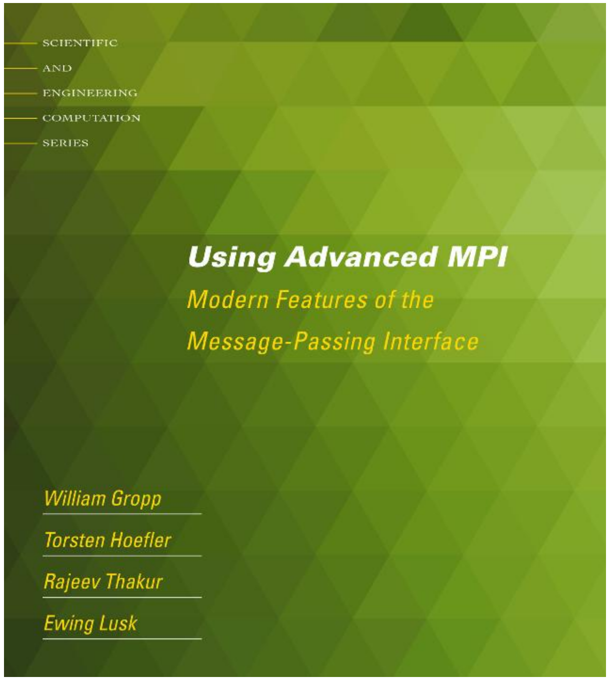
- Element-wise computation on a regular grid using a fixed neighborhood
- Typically working on multiple input fields and writing a single output field

```
lap(i,j) = -4.0 * in(i,j) + in(i-1,j) + in(i+1,j) + in(i,j-1) + 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
[2]: Matthias Christen, et al.: *PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations …, IPDPS'11*
[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)

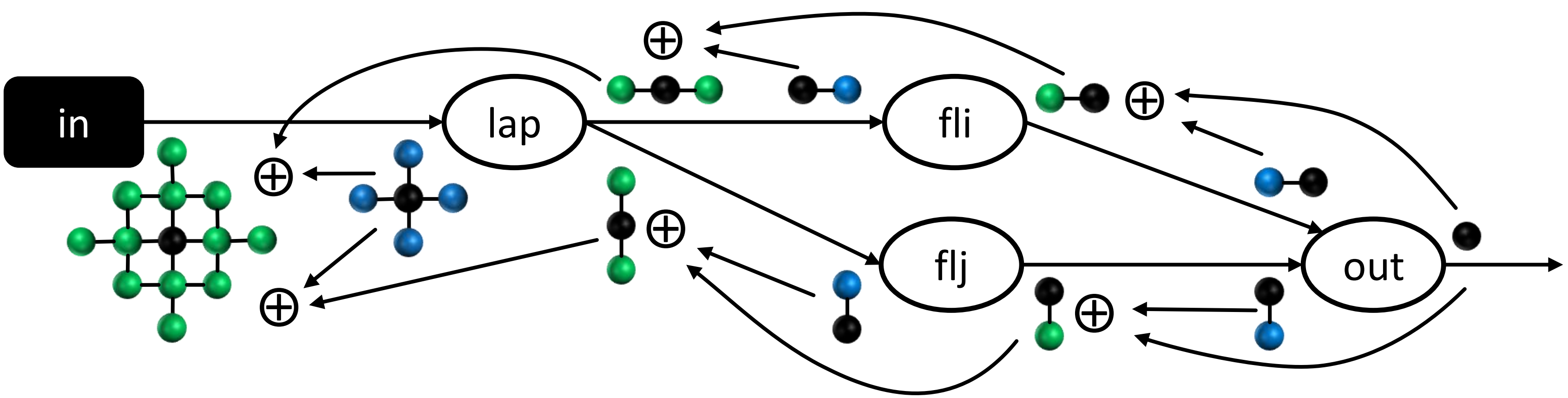
E.g., the COSMO weather code

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

Modeling stencils formally:

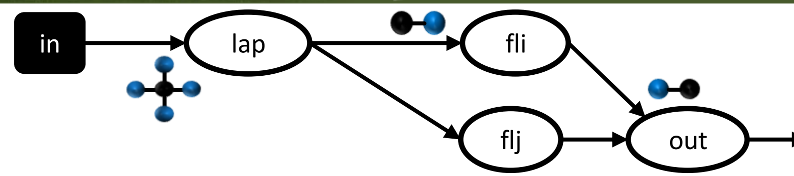
- **Represent stencils as DAGs**
  - Model stencil as nodes, data dependencies as edges

simplified horizontal diffusion example

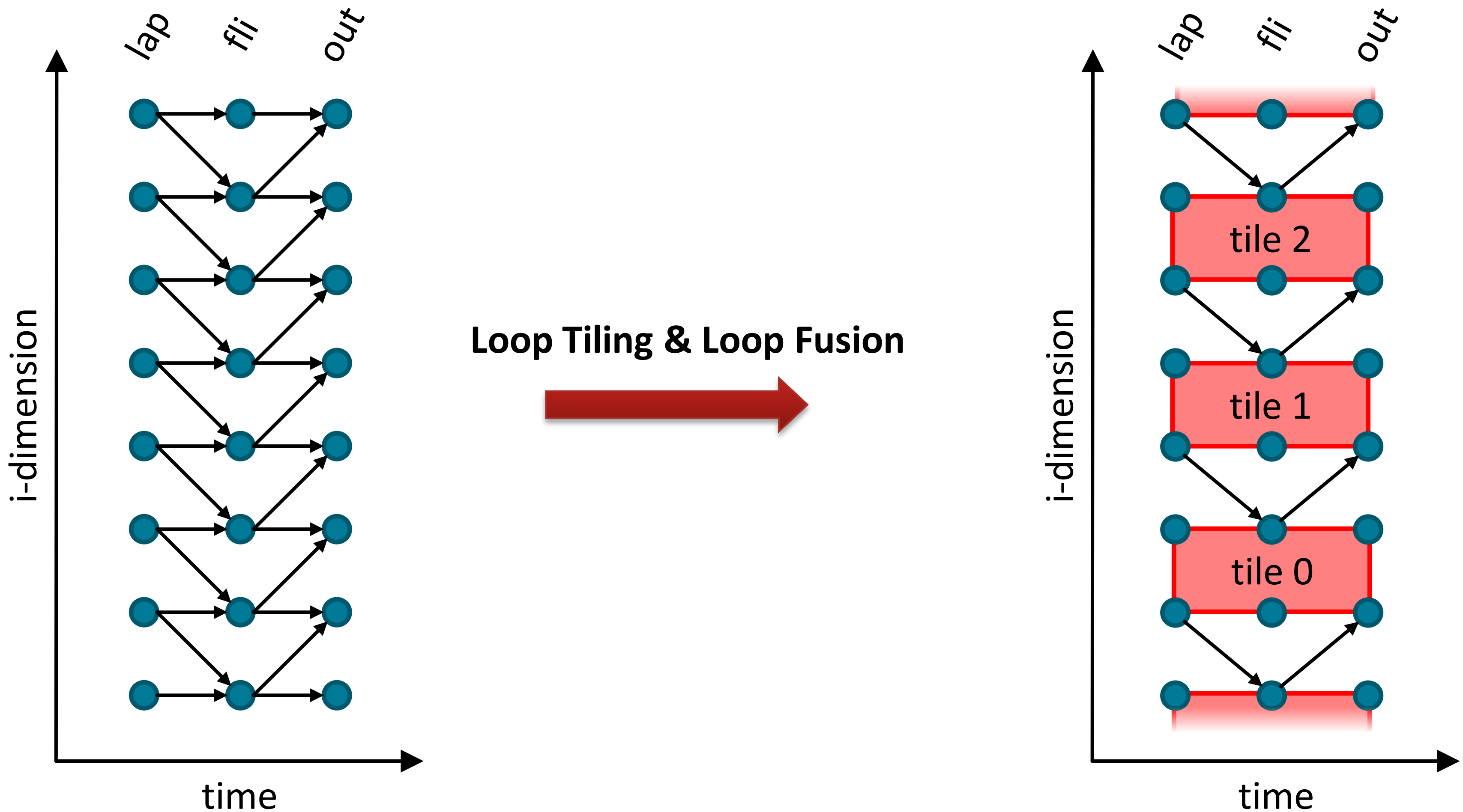


$$a \oplus b = \{a' + b' \mid a' \in a, b' \in b\}$$

Data-locality Transformations

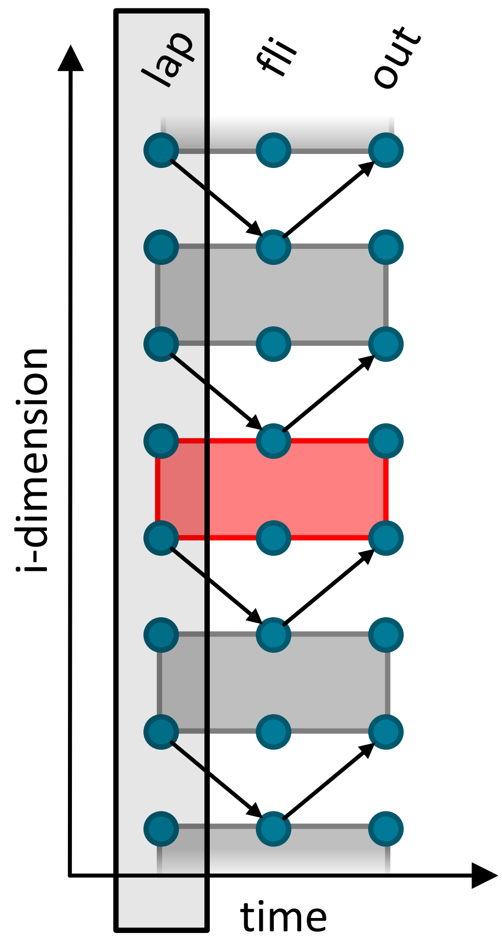


- **Consider the horizontal diffusion lap-fli-out dependency chain (i-dimension)**



How to Deal with Data Dependencies?

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



Halo Exchange Parallel (hp):

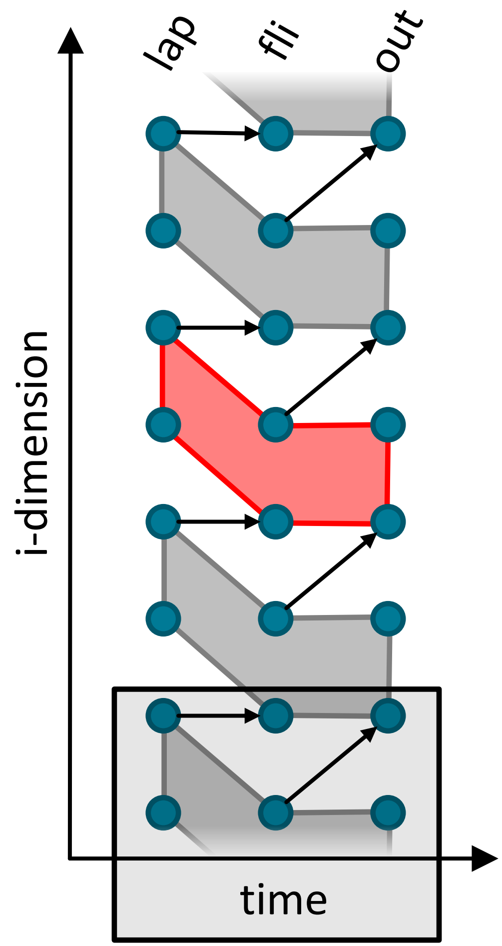
- Update tiles in parallel
- Perform halo exchange communication

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)

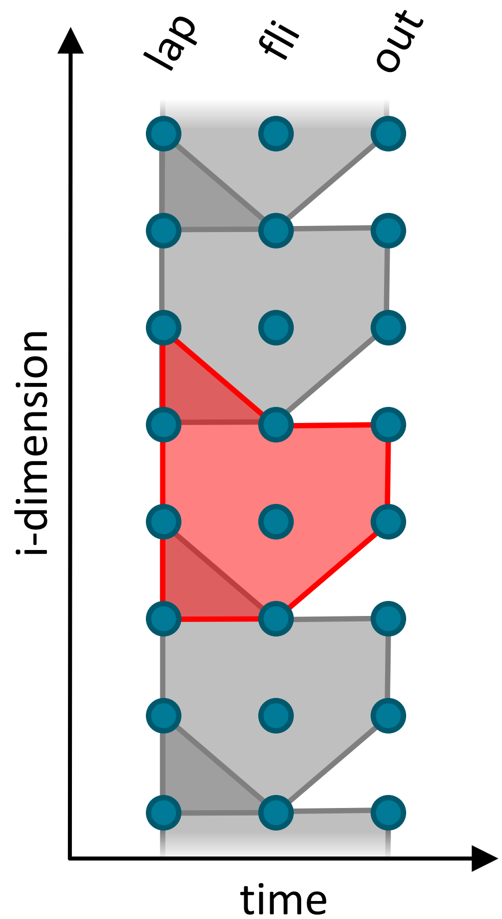


Halo Exchange Sequential (hs):
- Update tiles sequentially
- Innermost loop updates tile-by-tile

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)**

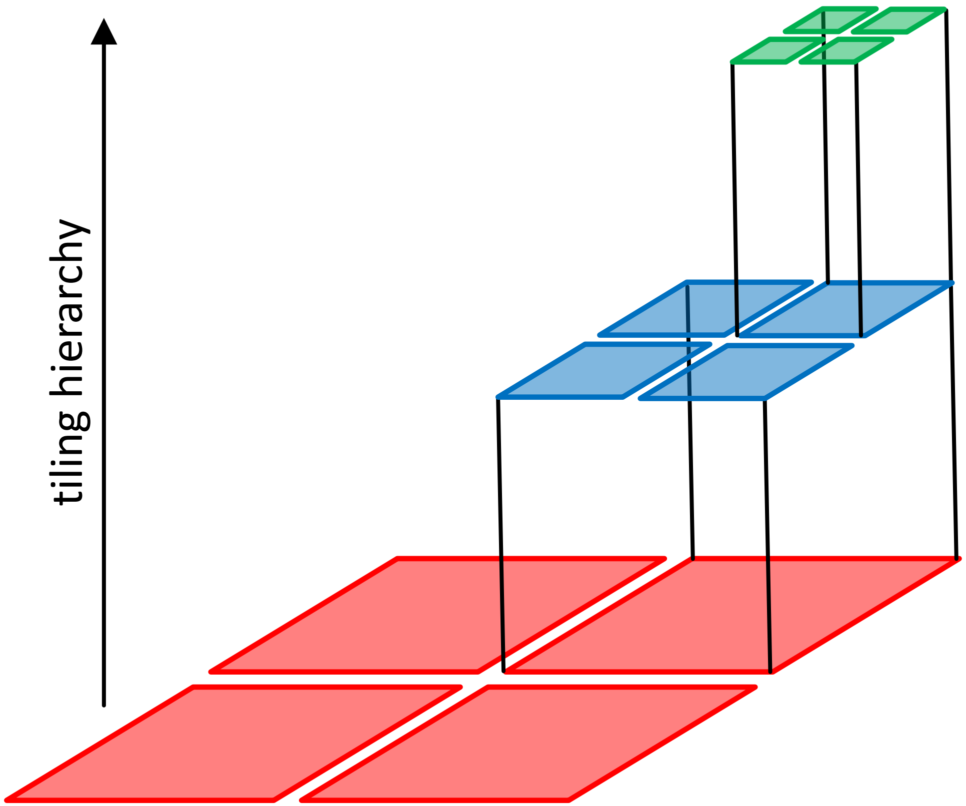


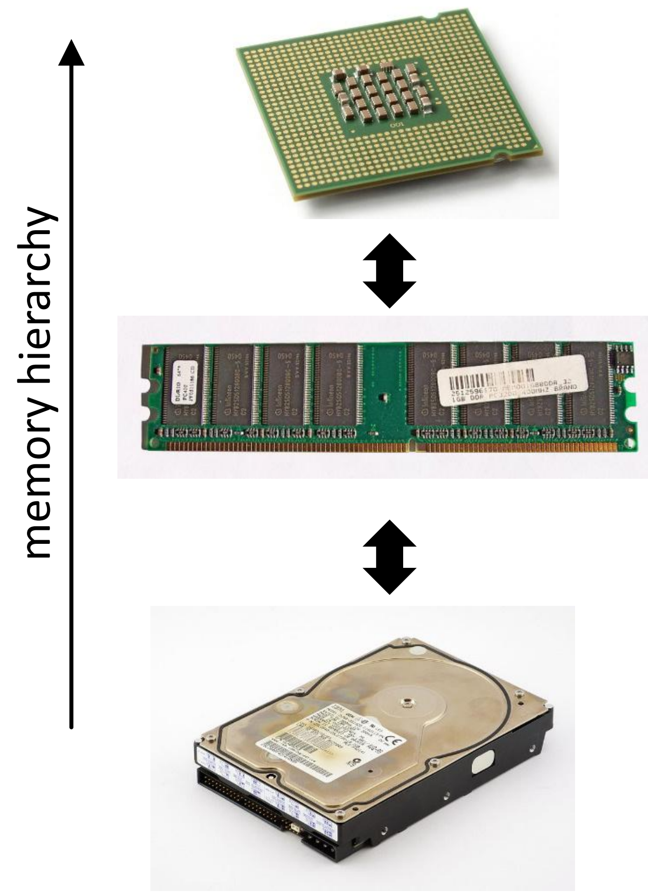
Computation on-the-fly (of):
- Compute all dependencies on-the-fly
- Overlapped tiling

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

Hierarchical Tiling

- **By tiling the domain repeatedly we target multiple memory hierarchy levels**



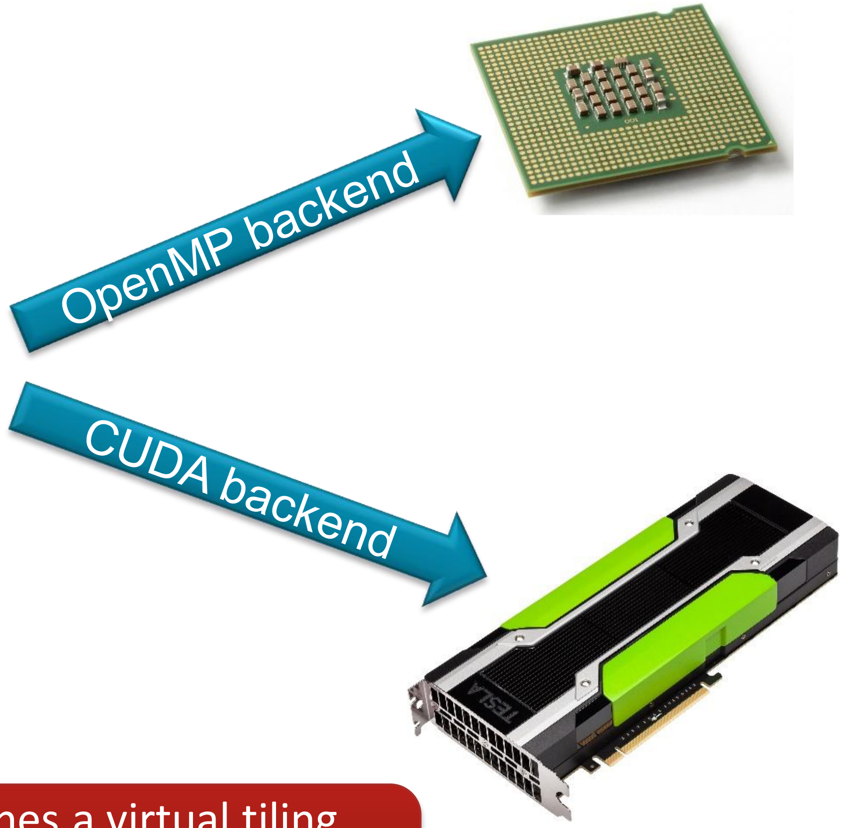


Case Study: STELLA (STEncil Loop LAnguage)

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

```
// 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();
```

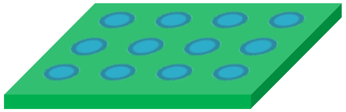
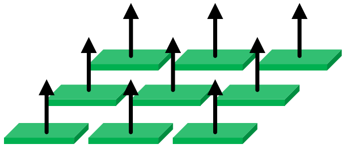
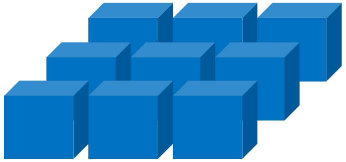
using C++ template metaprogramming:



STELLA defines a virtual tiling hierarchy that facilitates platform independent code generation

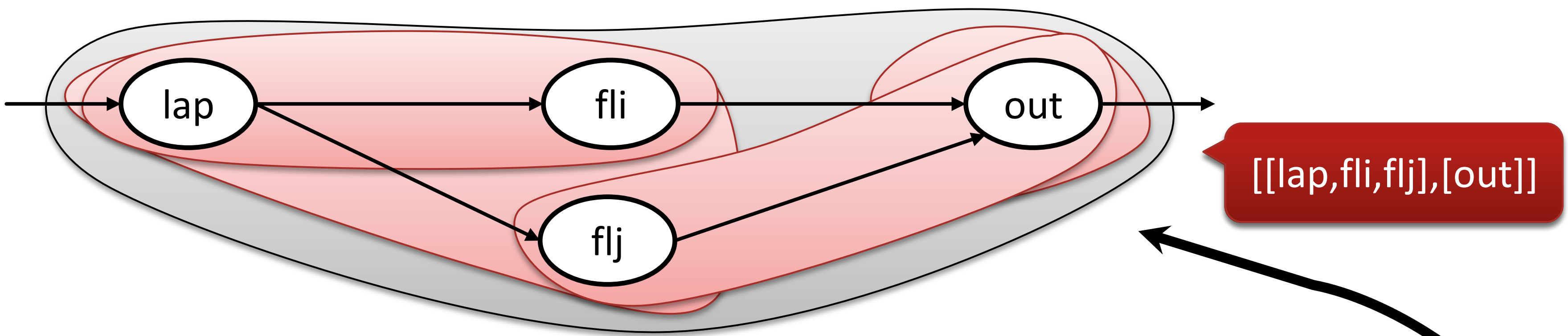
Tiling Hierarchy of STELLA's GPU-Backend

tiling hierarchy ↑

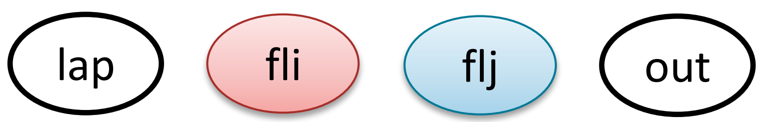
DSL	Tile Size	Strategy	Memory	Communication	
sweep	1 x 1 x 1	halo exchange parallel	registers	scratchpad	
sweep	∞ x ∞ x 1	halo exchange sequential	registers	registers	
loop	64x4x64	computation on-the-fly	GDDR	-	
stencil	∞ x ∞ x ∞	computation on-the-fly	GDDR	-	

Stencil Program Algebra

- Map stencils to the tiling hierarchy using a bracket expression



- Enumerate the stencil execution orders that respect the dependencies



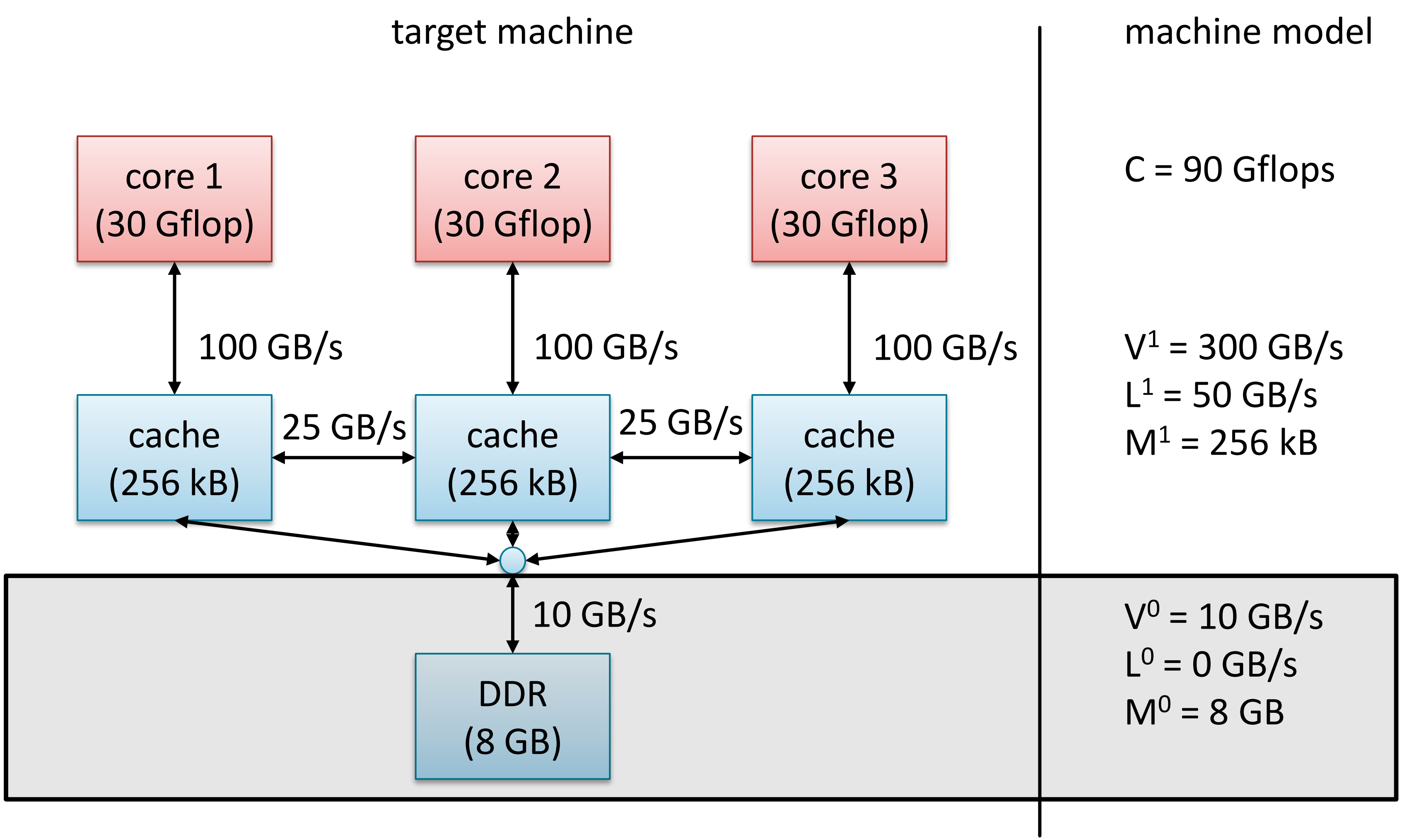
- Enumerate implementation variants by adding/removing brackets

...,lap,fli]],[[flj,out,...

Machine Performance Mo

lateral and vertical communication refer to communication within one respectively between different tiling hierarchy levels

- **Our model considers peak computation and communication throughputs**



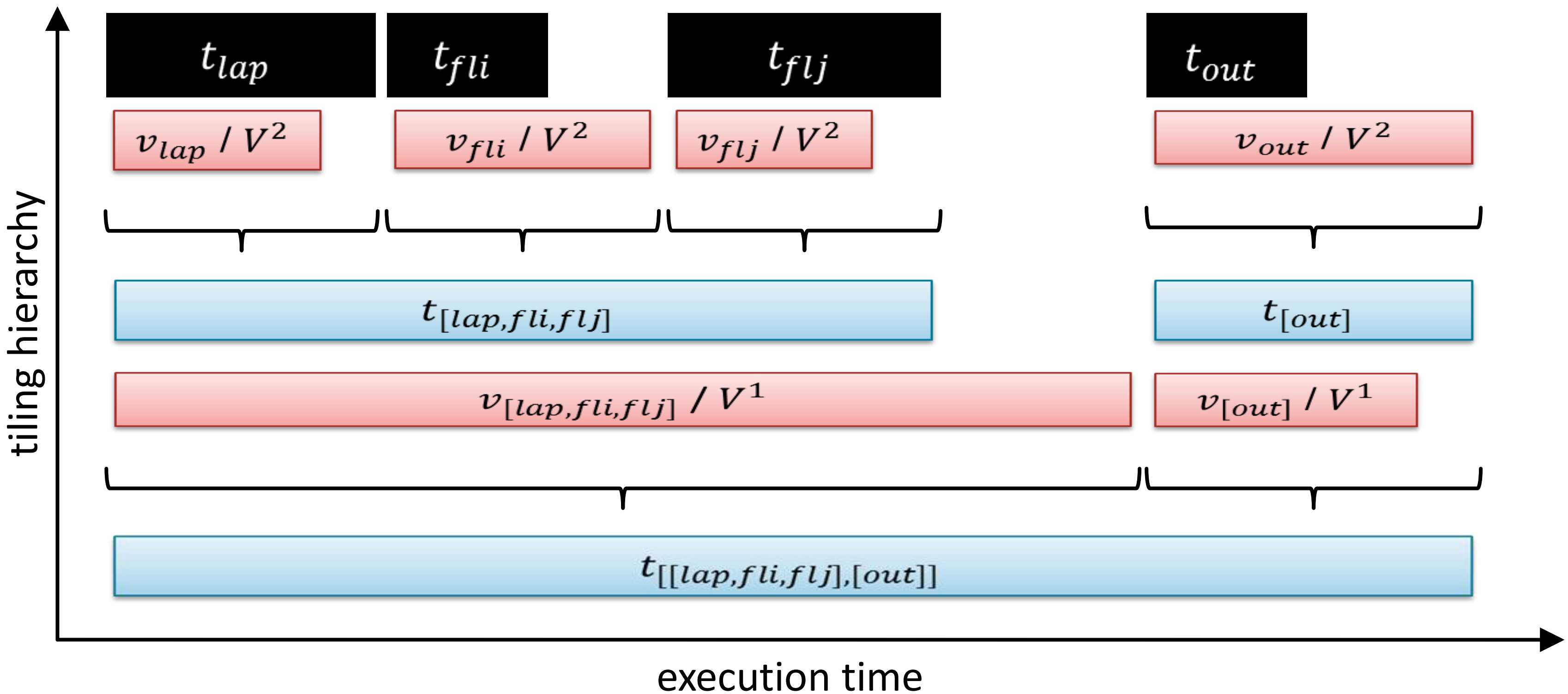
Stencil Performance Model - Overview

- Given a stencil s given and the amount of computation c_s

$$t_s = c_s / C$$

- Given a group g and the vertical and lateral communication v_c and $l_c^1, \ldots, l_c^m$

$$t_g = \sum_{c \in g.child} \max(t_c, v_c/V^m, l_c^1/L^1, \ldots, l_c^m/L^m)$$



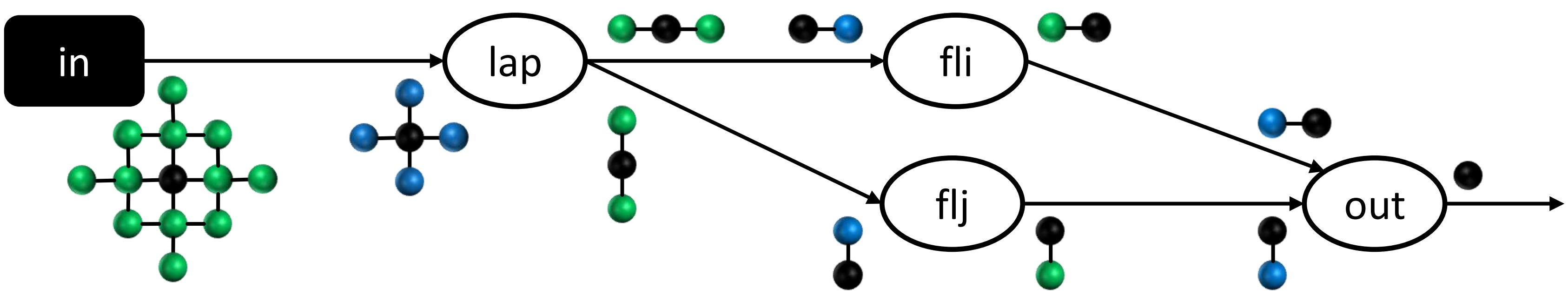
Stencil Performance Model - Affine Sets and Maps

- The stencil program analysis is based on (quasi-) affine sets and maps

$$S = \{\vec{\imath} \mid \vec{\imath} \in \mathbb{Z}^n \wedge (0, \dots, 0) < \vec{\imath} < (10, \dots, 10)\}$$
$$M = \{\vec{\imath} \rightarrow \vec{\jmath} \mid \vec{\imath} \in \mathbb{Z}^n, \vec{\jmath} \in \mathbb{Z}^n \wedge \vec{\jmath} = 2 \cdot \vec{\imath}\}$$

- For example, data dependencies can be expressed using named maps

$$D_{fli} = \{(fli, \vec{\imath}) \rightarrow (lap, \vec{\imath} + \vec{\jmath}) \mid \vec{\imath} \in \mathbb{Z}^2, \vec{\jmath} \in \{(0,0), (1,0)\}\}$$



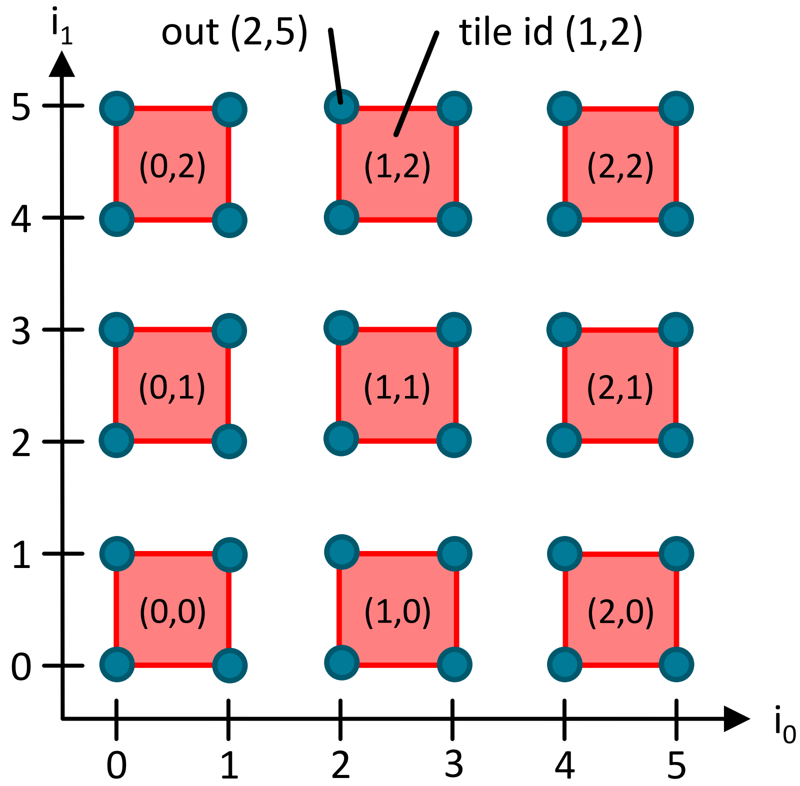
$$D = D_{lap} \cup D_{fli} \cup D_{flj} \cup D_{out}$$
$$E = \mathrm{D}^{+}(\{ (out, \vec{0}) \})$$

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**

$$T_{out} = \{(\text{out}, (i_0, i_1)) \to (\lfloor i_0/2 \rfloor, \lfloor i_1/2 \rfloor)\}$$



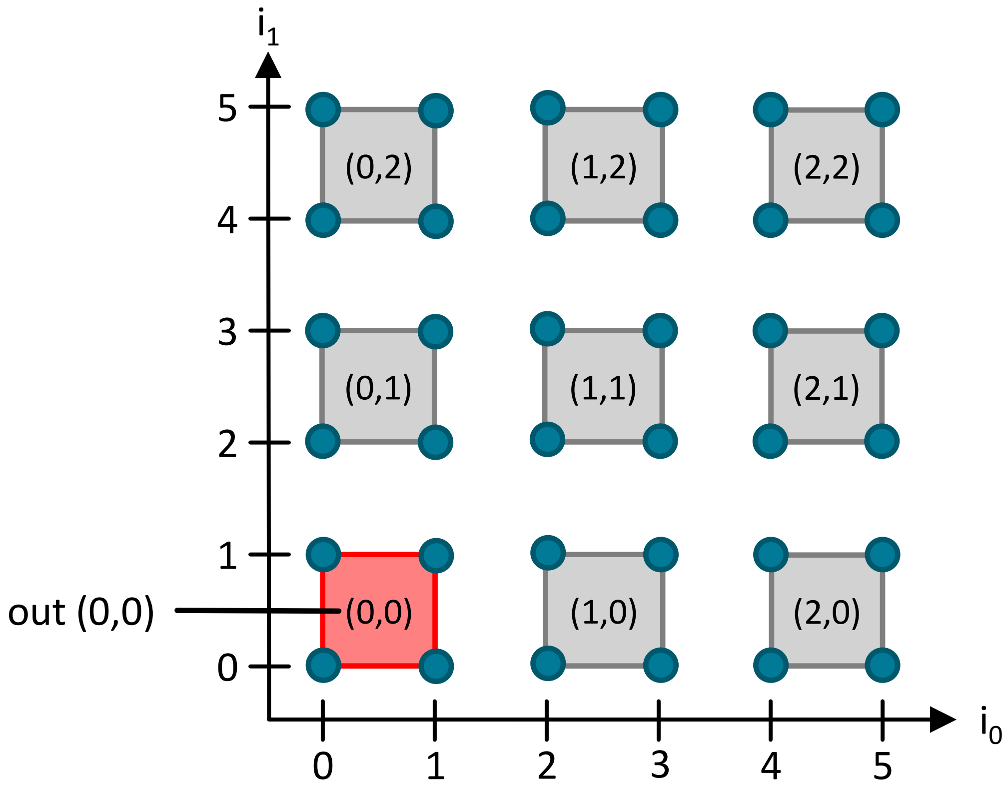
Stencil Performance Model – Comp & Comm

- **Count floating point operations necessary to update tile (0,0)**

$$c_{out} = |T_{out} \cap_{ran} \{(0,0)\}| \cdot \#flops$$

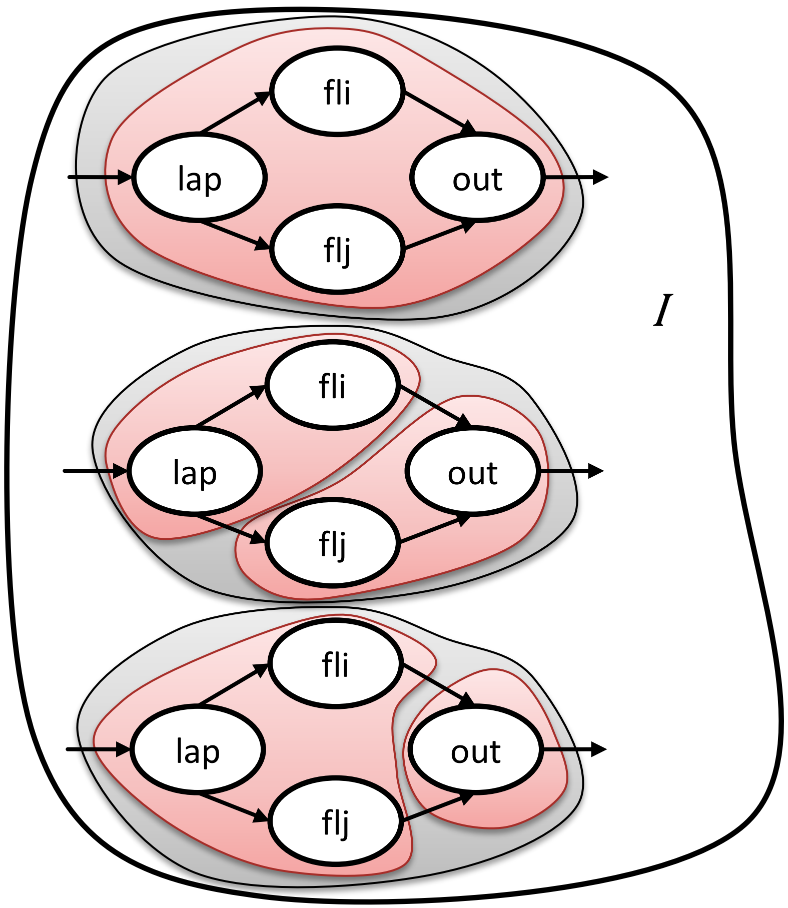
- **Count the number of loads necessary to update tile (0,0)**

$$l_{out} = |(T_{out} \circ D_{out}^{-1}) \cap_{ran} \{(0,0)\}|$$



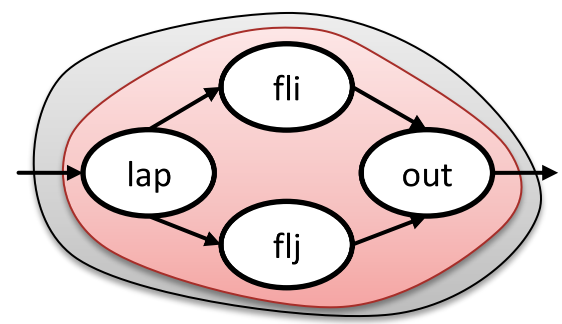
Analytic Stencil Program Optimization

- **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)



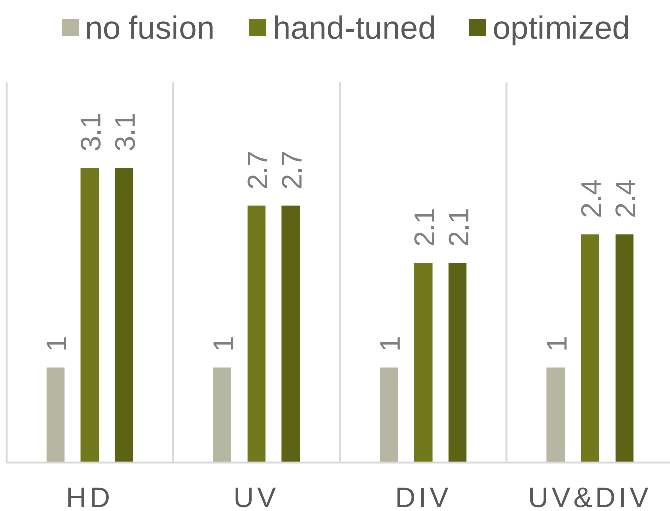
$$\underset{x \in I}{\text{minimize}}\, t(x)$$

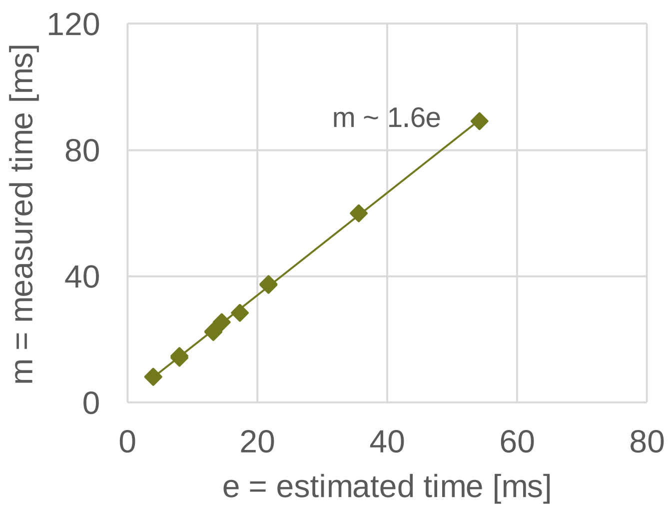
$$\text{subject to } m(x) \leq M$$



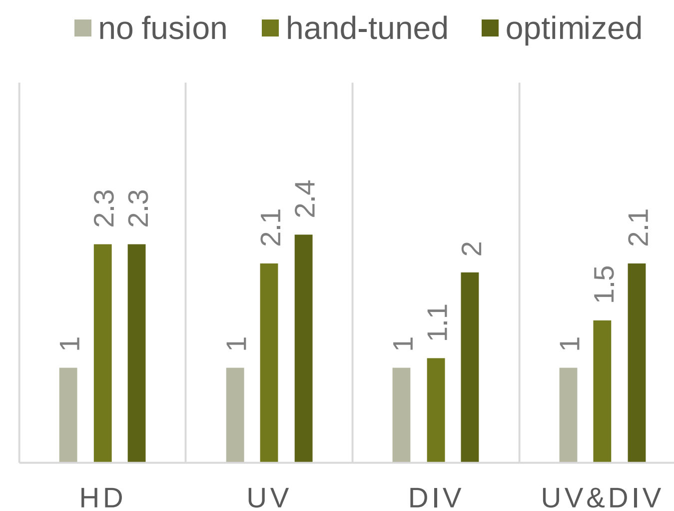
Evaluation

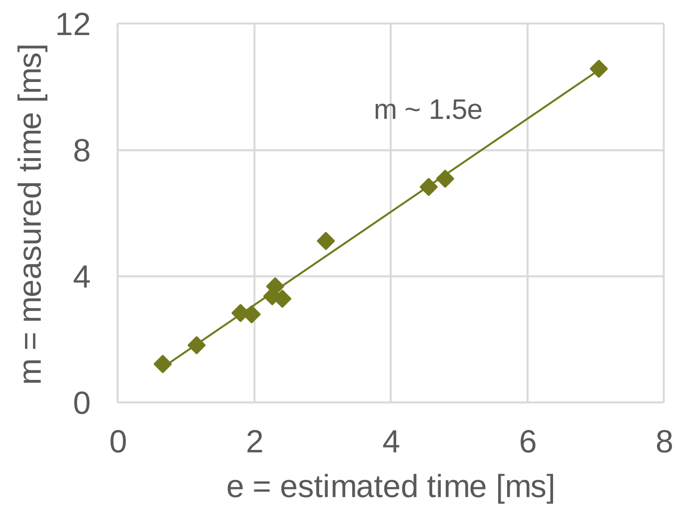
CPU Experiments (i5-3330):





GPU Experiments (Tesla K20c):





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

Sponsors: ETH-RAT · SUK·CUS Schweizerische Universitätskonferenz · PASC15 Platform for Advanced Scientific Computing Conference, Zürich Switzerland, 01-03 June 2015

Backup Slides