An Interactive Visual User Iterface for Program Optimization Using Loopy

James Stevens

April 7, 2020

Acknowledgements

Loopy-UI:

- Summer Xia
- Eunsun Lee
- Juefei Chen
- Feng Hou
- Andreas Klöckner
- Bogdan Enache

Loopy:

- Andreas Klöckner
- Matt Wala
- Kaushik Kulkarni

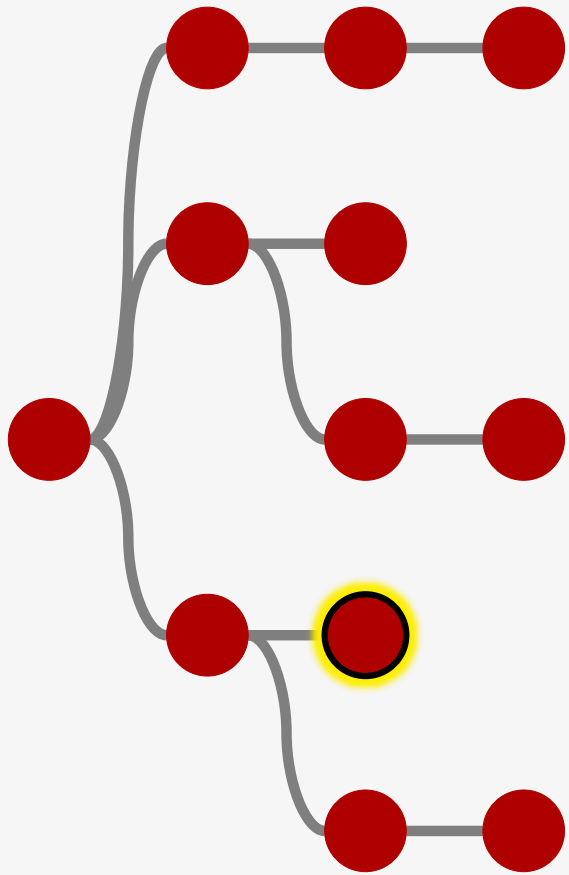
Question

Can we build a system that assists a user in:

- Applying program transformations for optimization
- Exploring search space of mathematically equivalent program variants
  - Including evaluation of performance stats

Subject to the constraints that:

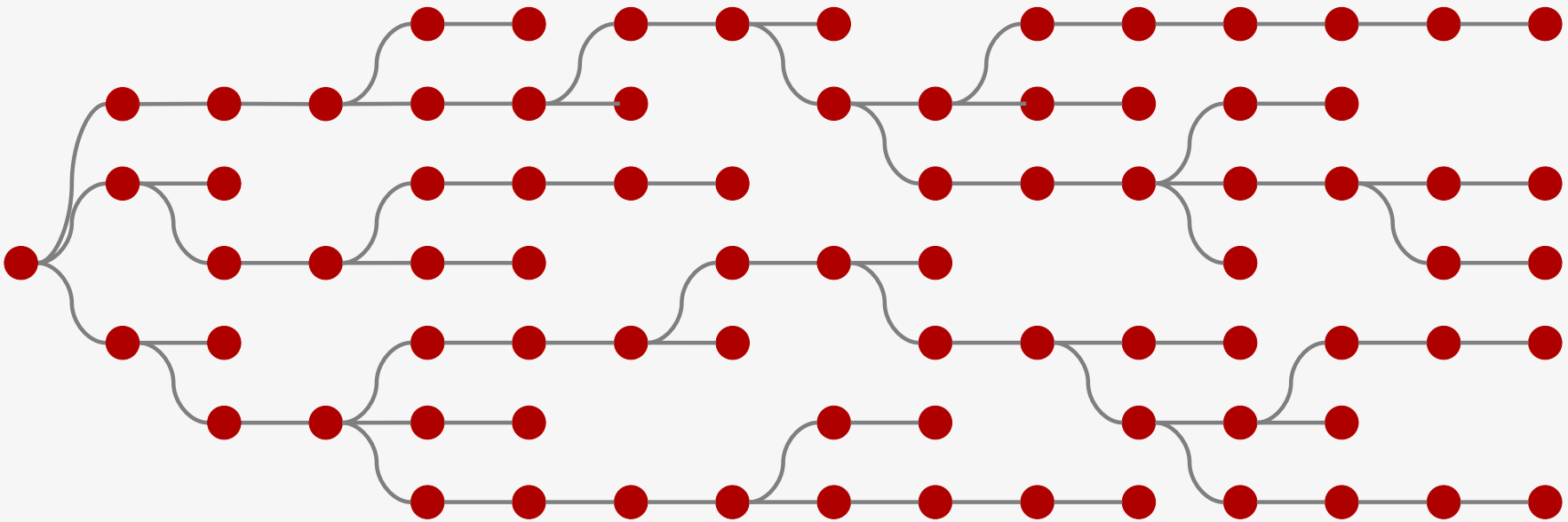
- Expression of mathematical intent is invariant to code transformations
- Source code is not manually written/modified
- Program variants can be easily reproduced



Question

In building this system, can we answer the following additional questions:

- Which program variants are reachable?
- Can system scale with program size while meeting desired criteria?



Primary Components

- LOOPY: Program abstraction and code generation engine
- LOOPY-UI: Visual interface for optimizing LOOPY programs

Motivation for Program Abstraction and Code Generation

Parallel (GPU) code dev:	Hand-written	Compiler directives	Goal
Low development time	✗	✓	✓
Low code modification time	✗	✓	✓
High P(optimal performance)	✓	✗	✓
Low P(coding errors)	✗	✓	✓
Modify source at runtime	✗	✗	✓

P(X): probability of X

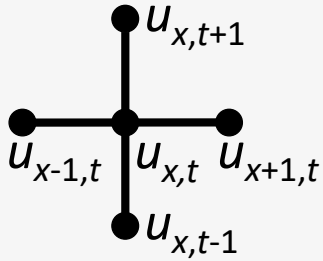
Loopy Example

- Programming system for array computations providing GPU/CPU code generation
- Separates mathematical intent from computational minutiae

Example Loopy program: solve wave equation

```
knl = lp.make_kernel(
  "[nx,nt] -> {[x, t]: 1<=x<nx-1 and 0<=t<nt}",
  """
  u[x, t+2] = (
   dt**2/dx**2 * (u[x+1, t+1] - 2*u[x, t+1] + u[x-1, t+1])
   + 2*u[x, t+1] - u[x, t])
  """)
knl = lp.prioritize_loops(knl, ("t", "x"))
```

$$u_{tt} = c \cdot u_{xx}$$



[Klöckner et al.(2016)],[Klöckner(2015)],[Klöckner(2014)]

Loopy Example

Loopy-generated code:

```
__kernel void [...] wave_equation(float const dt, float const dx, int
    const nt, int const nx, __global float *__restrict__ u)
{
  for (int t = 0; t <= -1 + nt; ++t)
    for (int x = 1; x <= -2 + nx; ++x)
      u[(2 + nt) * x + 2 + t] =
        2.0f * u[(2 + nt) * x + 1 + t] +
        ((dt * dt) / (dx * dx)) * (u[(2 + nt) * (1 + x) + 1 + t] + -1.0
            f * 2.0f * u[(2 + nt) * x + 1 + t] + u[(2 + nt) * (-1 + x)
            + 1 + t]) +
        -1.0f * u[(2 + nt) * x + t];
}
```

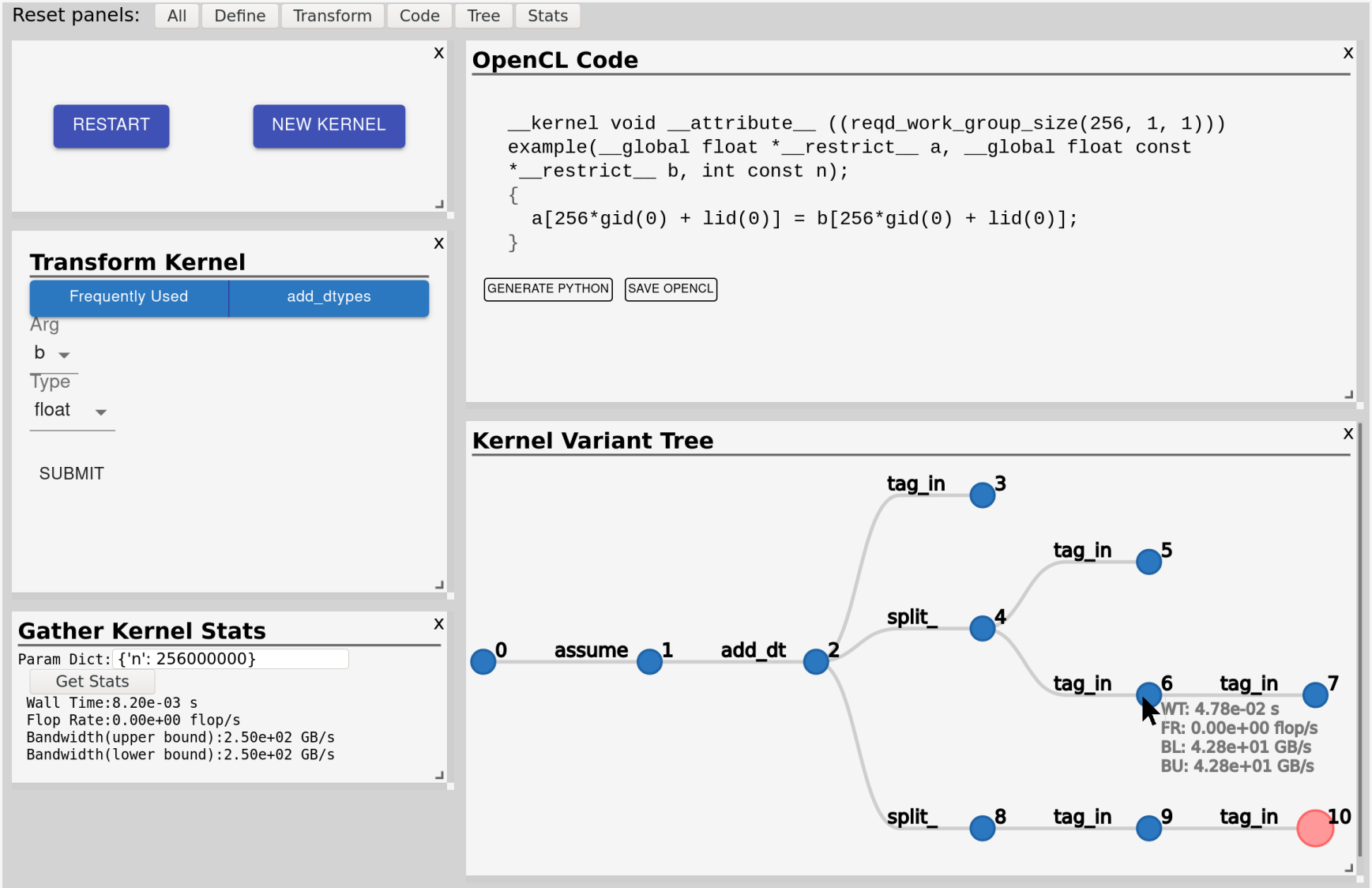
Loopy program transformations:

- Loop splitting, unrolling, vectorization, parallelization, prefetching, ...

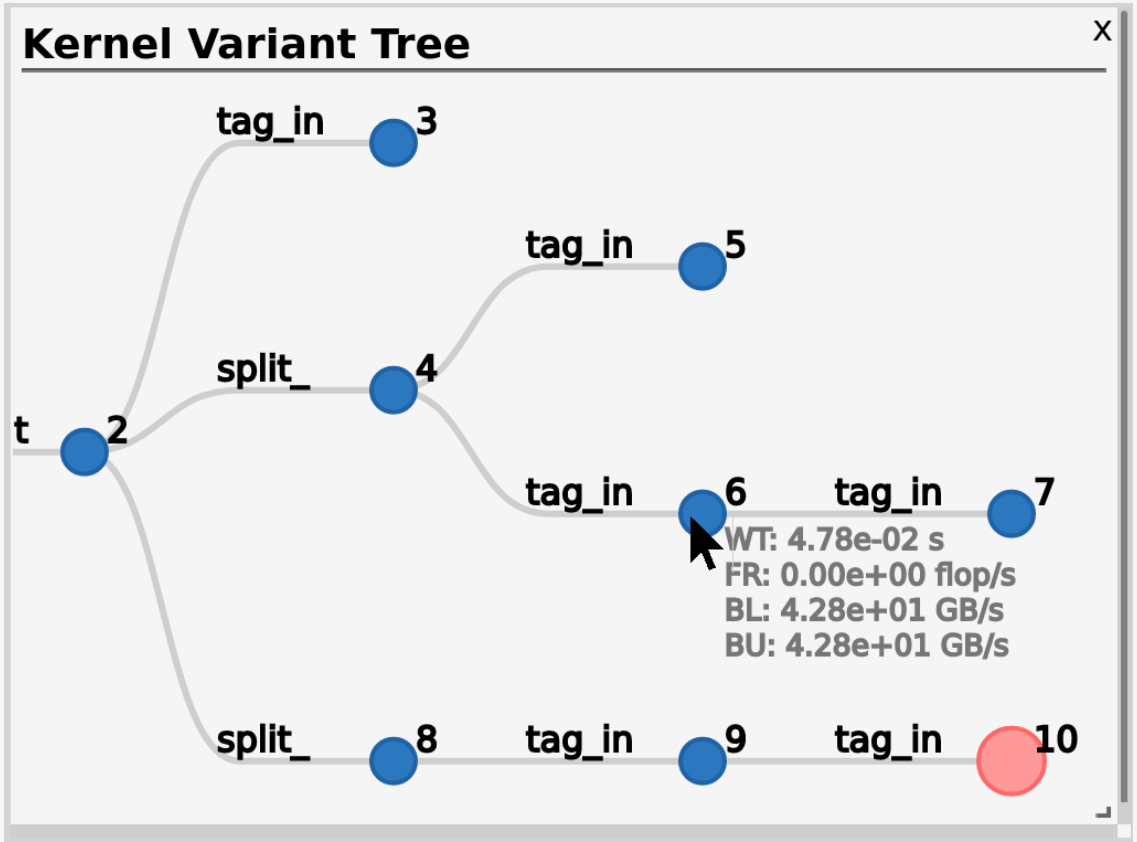
Desired System Capabilities

- (☑ Loopy capability)
  - ☑ Describe mathematical intent independently of implementation
  - ☐ Optimize algorithm without manually writing/modifying
    - ☑ source code or
    - ☐ higher level code that generates source (e.g., Python)
  - ☐ Visualize + interactively explore search space of optimization strategies
    - ☐ with immediate evaluation of performance and other stats
  - ☐ Without rewriting code, recreate
    - ☑ program variant source code
    - ☐ interactive search space of multiple program variants

Demos

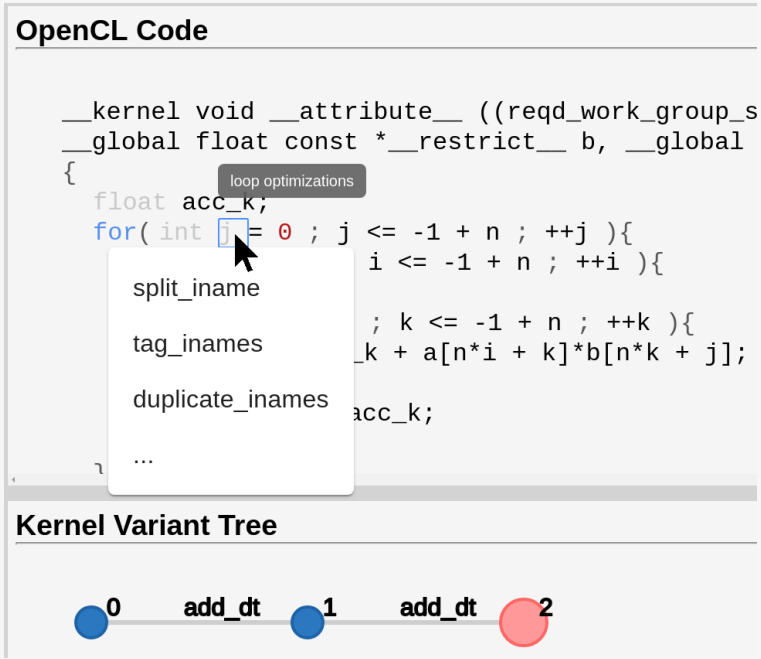

Reset panels:
All
Define
Transform
Code
Tree
Stats
RESTART
NEW KERNEL
Transform Kernel
Frequently Used
add_dtypes
Arg
b
Type
float
SUBMIT
Gather Kernel Stats
Param Dict: {'n': 256000000}
Get Stats
Wall Time:8.20e-03 s
Flop Rate:0.00e+00 flop/s
Bandwidth(upper bound):2.50e+02 GB/s
Bandwidth(lower bound):2.50e+02 GB/s
OpenCL Code
__kernel void __attribute__ ((reqd_work_group_size(256, 1, 1)))
example(__global float *__restrict__ a, __global float const
*__restrict__ b, int const n);
{
a[256*gid(0) + lid(0)] = b[256*gid(0) + lid(0)];
}
GENERATE PYTHON
SAVE OPENCL
Kernel Variant Tree
0
assume
1
add_dt
2
tag_in
3
split_
4
tag_in
5
tag_in
6
tag_in
7
WT: 4.78e-02 s
FR: 0.00e+00 flop/s
BL: 4.28e+01 GB/s
BU: 4.28e+01 GB/s
split_
8
tag_in
9
tag_in
10

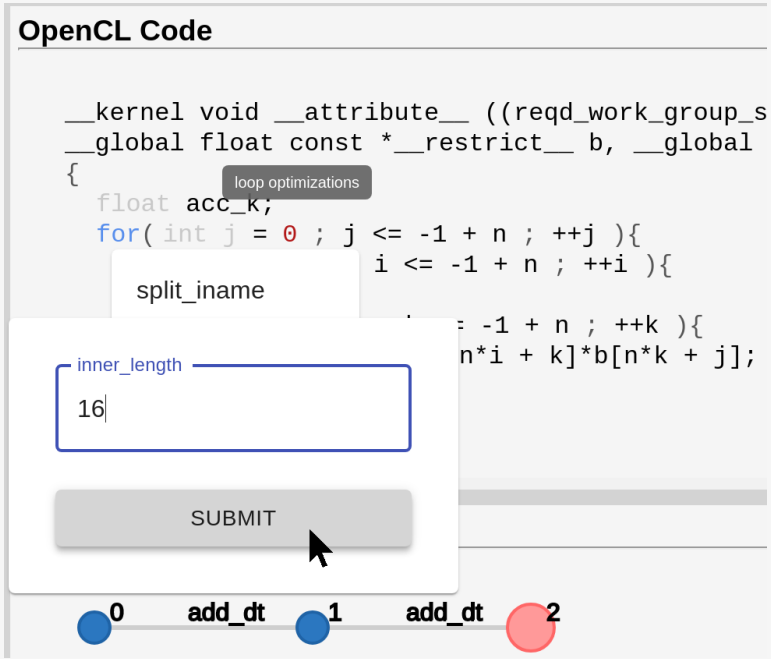
Summary of Key UI Capabilities



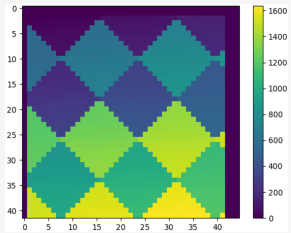
- Interactive variant exploration tree with stats and transformation info
  - View, compare, and evaluate multiple variant paths simultaneously
  - Reload tree to revisit search space, e.g., on new hardware

Summary of Key UI Capabilities

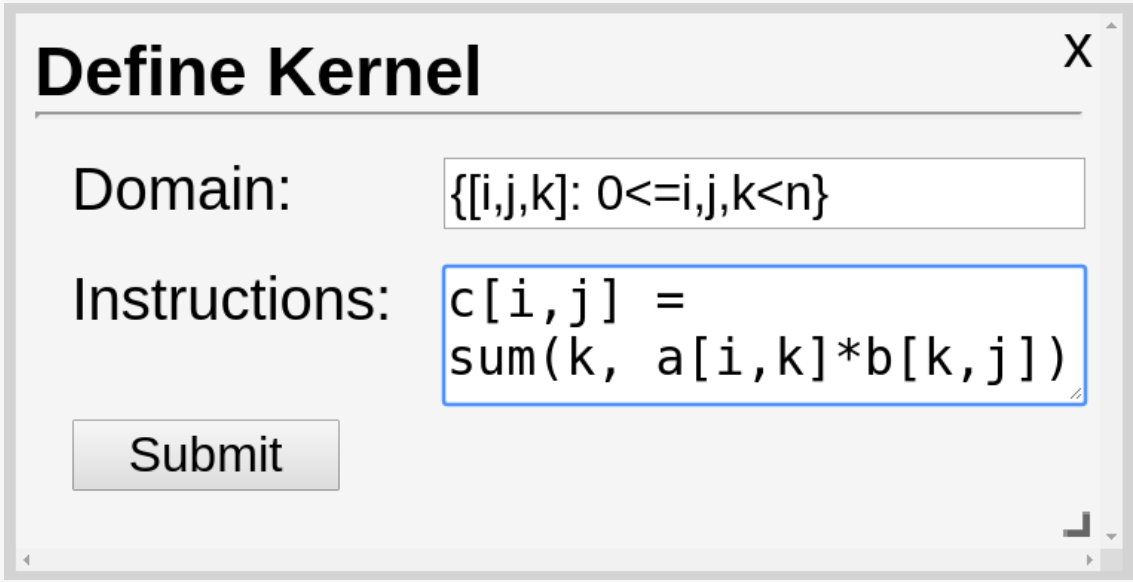




- Interactive source code
  - Access and execute relevant transformations
    - without writing/modifying code and
    - without extensive knowledge of transformation system
  - View performance-relevant program info, e.g., mouse hover for memory access pattern visualization



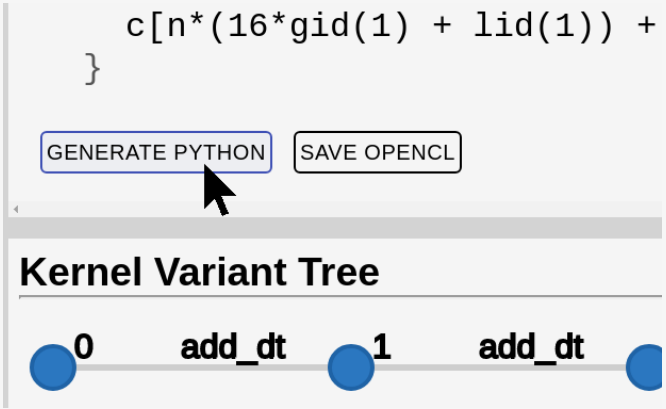
Summary of Key UI Capabilities



- Determine mathematical intent independently of implementation strategy
  - Address efficiency concerns without changing mathematical description, e.g., when moving to new hardware

Summary of Key UI Capabilities

- Export to production: save python transformation chain or source
  - Generate source on new hardware
  - Revisit search tree on new hardware



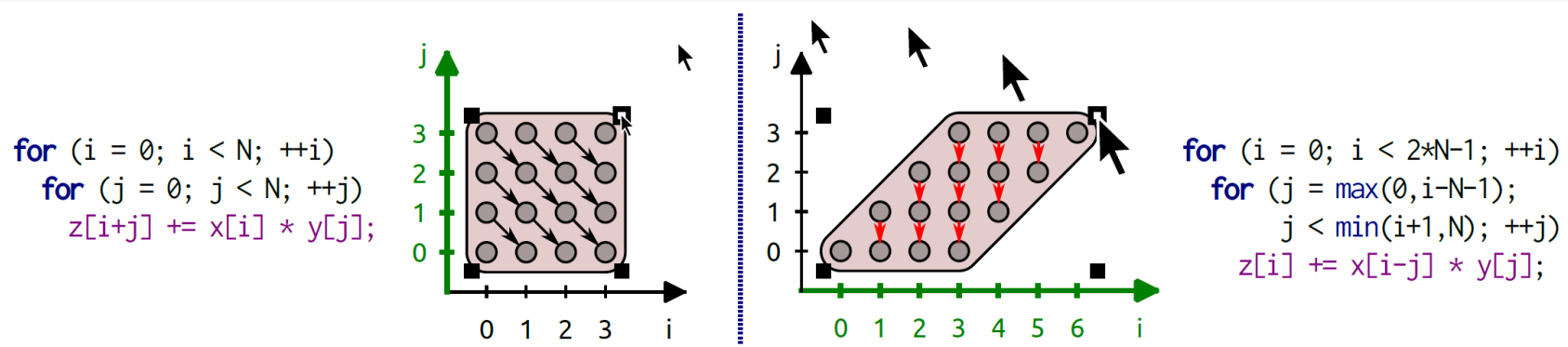
- Import program from existing LOOPY code
  - Use UI with kernel generated within larger program

```
# [...1000s of lines of prep code...]
knl = generate_kernel_with_complex_machinery(...)
launch_loopy_ui_with(knl, browser=default_browser)
```

Related Work

Related work differences:

- Method of transformation application
  - Interact with abstract representations/diagrams of program
  - Modify source directly (manually re-write)

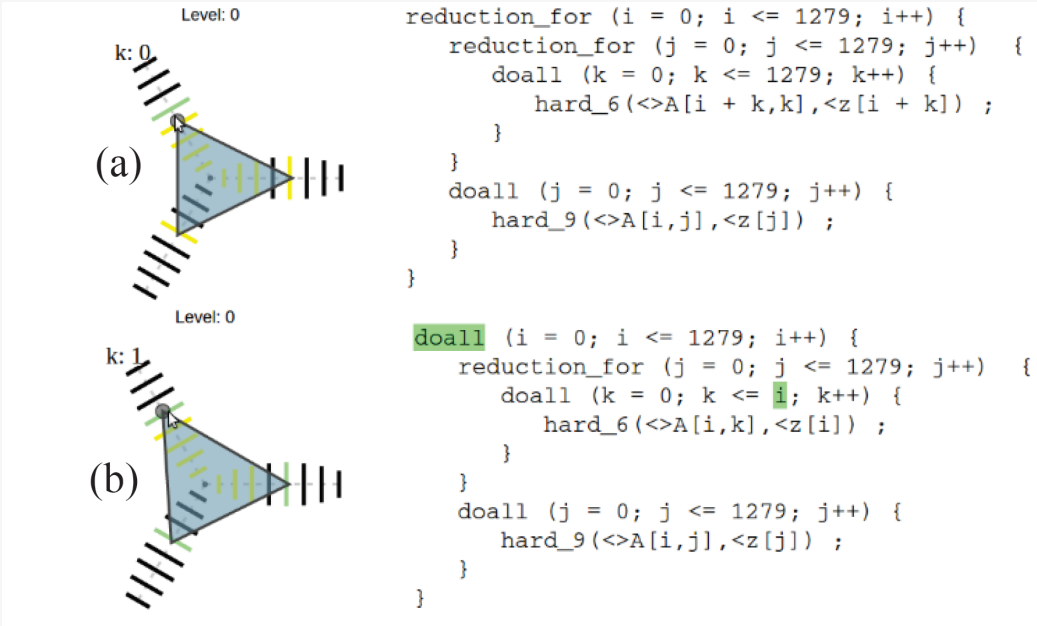


Zinenko et al.(2018)

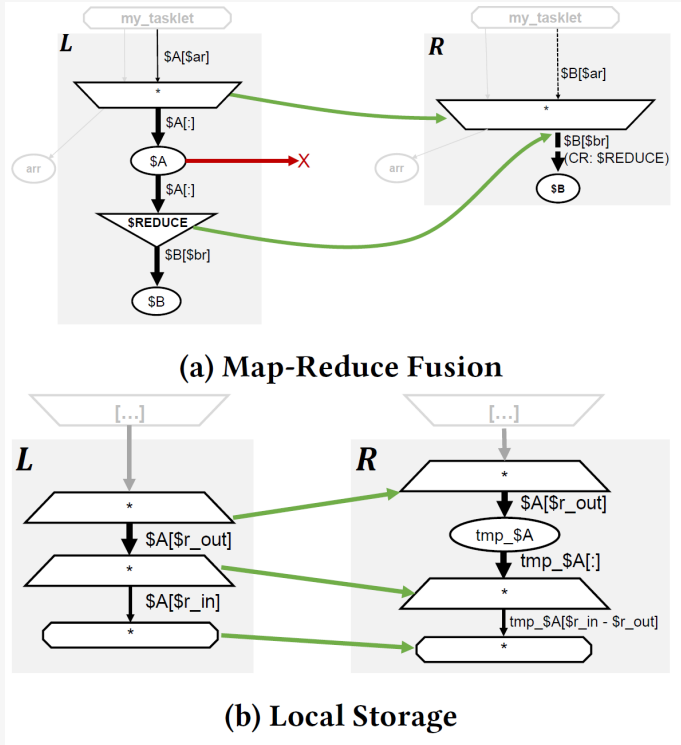
Related Work

Related work differences:

- Method of transformation application
  - Interact with abstract representations/diagrams of program
  - Modify source directly (manually re-write)



Papenhausen et al.(2016)



Ben-Nun et al.(2019)

Related Work

Related work differences:

- Input representation
  - Source, restricted python/matlab, numpy functions
- Search space accessibility/visibility
  - Most only display 1-d program history, if any
  - None(?) allow storage/reuse of full transformation tree w/branches
  - None(?) allow simultaneous viewing of stats for multiple variants
- Accessability of program internal representation
  - With LOOPY, user can directly interact with a program IR that is amenable to understanding (contrast with LLVM IR)
- Scope of transformations available

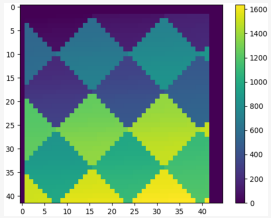
Conclusions and Future Work

Accomplished in current (alpha) version of UI:

- ☑ Kernel and transformation input via UI
- ☑ Interactive source code (limited transformation options)
- ☑ Statistics gathering
- ☑ Interactive, navigable variant tree displaying stats
- ☑ Saving source and LOOPY script to regenerate LOOPY kernel

Conclusions and Future Work

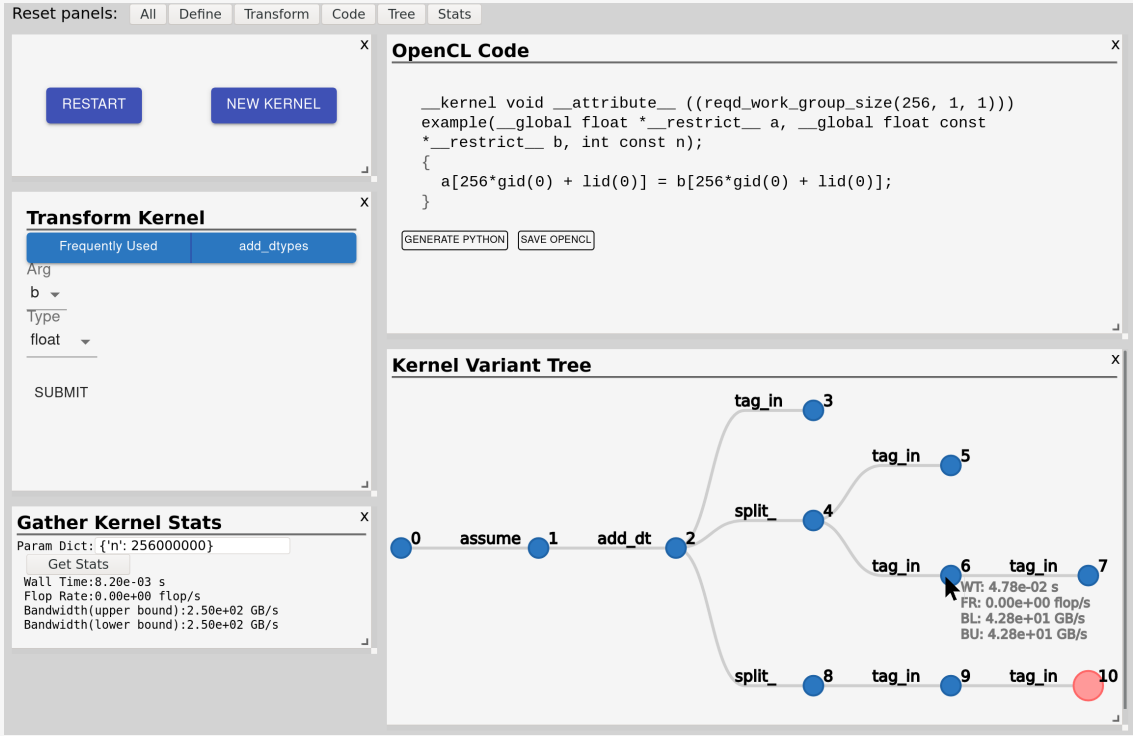
Coming soon:



- More transformations and info available via interactive code
  - E.g., hover over array access for memory access pattern visualization
- Store and **reload variant tree**
- Launch UI with existing Loopy kernel from Python script
- Tree improvements: clearer communication of transformations, **segment hiding**
- Fix bugs and otherwise improve user experience
- Get and incorporate **feedback from Loopy users**
- Convenient module installation and setup

Want to try it?

1. Install Loopy: `https://github.com/inducer/loopy`
2. Install Loopy-UI: `https://gitlab.tiker.net/jdsteve2/loopy-ui`
   - Get repo permission: jdsteve2@illinois.edu
   - Follow README for toolchain setup



Bibliography I

Tal Ben-Nun, Johannes de Fine Licht, Alexandros N. Ziogas, Timo Schneider, and Torsten Hoefler.

Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures.
In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.
ISBN 9781450362290.
doi: 10.1145/3295500.3356173.
URL https://doi.org/10.1145/3295500.3356173.

Andreas Klöckner.

Loo.Py: Transformation-based Code Generation for GPUs and CPUs.
In *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY'14, pages 82:82–82:87, New York, NY, USA, 2014. ACM.
ISBN 978-1-4503-2937-8.
DOI: 10.1145/2627373.2627387.

Andreas Klöckner.

Loo.Py: From Fortran to Performance via Transformation and Substitution Rules.
In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ARRAY 2015, pages 1–6, New York, NY, USA, 2015. ACM.
ISBN 978-1-4503-3584-3.
DOI: 10.1145/2774959.2774969.

E. Papenhausen, K. Mueller, M. H. Langston, B. Meister, and R. Lethin.

An interactive visual tool for code optimization and parallelization based on the polyhedral model.
In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 309–318, 2016.

Bibliography II

Oleksandr Zinenko, Stéphane Huot, and Cédric Bastoul.
Visual program manipulation in the polyhedral model.
ACM Trans. Archit. Code Optim., 15(1), March 2018.
ISSN 1544-3566.
doi: 10.1145/3177961.
URL https://doi.org/10.1145/3177961.

Tiled Matrix-Matrix Multiplication with Prefetching

OpenCL Code

```
__kernel void __attribute__ ((reqd_work_group_size(16, 16, 1))) example(__global float const *__restrict__ a,
__global float const *__restrict__ b, __global float *__restrict__ c, int const n);
{
  __local float a_fetch [16 * 16];
  float acc_k_outer_k_inner;
  __local float b_fetch [16 * 16];
  acc_k_outer_k_inner = 0.0f;
  for( int k_outer = 0 ; k_outer <= loopy_floor_div_pos_b_int32(-16 + n, 16) ; ++k_outer ){
    barrier(CLK_LOCAL_MEM_FENCE) /* for b_fetch (b_fetch_rule rev-depends on insn_k_outer_k_inner_update) */
    b_fetch[16*lid(1) + lid(0)] = b[n*(16*k_outer + lid(1)) + 16*gid(0) + lid(0)];
    a_fetch[16*lid(1) + lid(0)] = a[n*(16*gid(1) + lid(1)) + 16*k_outer + lid(0)];
    barrier(CLK_LOCAL_MEM_FENCE) /* for a_fetch (insn_k_outer_k_inner_update depends on a_fetch_rule) */
    for( int k_inner = 0 ; k_inner <= 15 ; ++k_inner ){
      acc_k_outer_k_inner = acc_k_outer_k_inner + a_fetch[16*lid(1) + k_inner]*b_fetch[16*k_inner + lid(0)];
    }
  }
  c[n*(16*gid(1) + lid(1)) + 16*gid(0) + lid(0)] = acc_k_outer_k_inner;
}
```

GENERATE PYTHON SAVE OPENCL

Kernel Variant Tree