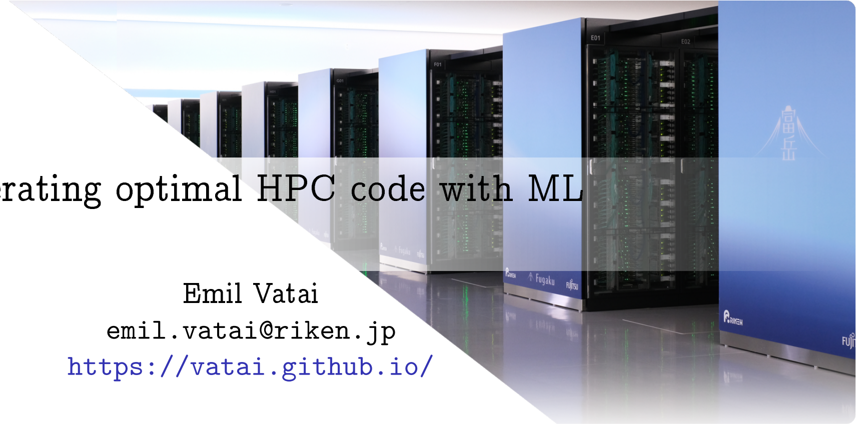


Generating optimal HPC code with ML

Emil Vatai
emil.vatai@riken.jp
https://vatai.github.io/

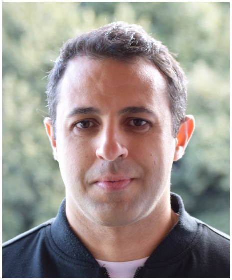
PASC23

Outline

- Intro
- What is code generation? Not ChatGPT!
- Representation? + HPC = Polyhedral
- Democratise Polyhedral: a polyhedral mini-tutorial
- Current status: Tadashi

RIKEN Center for Computational Science

High Performance Artificial Intelligence Systems Research Team

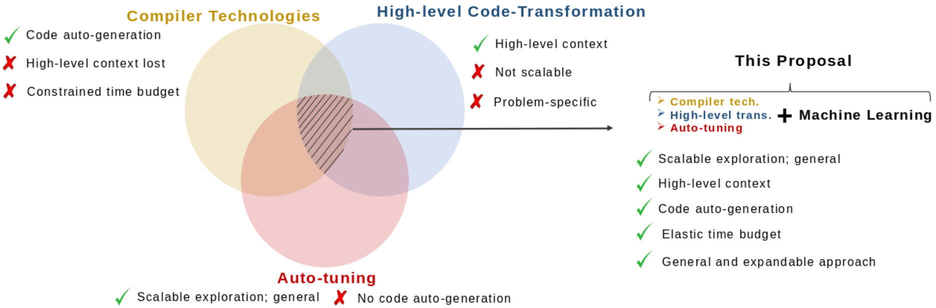


Mohamed Wahib
HPC+compilers

Aleksandr Drozd
HPC+AI

Emil Vatai
HPC+math

Motivation/Overview



What is code generations?

The ultimate is goal: “Hey AI, optimise this code!”

- Source to source transformations
- Targeting high-level optimisations.
  - Here High-level optimisations are code transformations which exploit deeper insight, an overview of the overall structure/context of the application
  - This is in contrast to low-level, local transformations performed by compilers.
- Something with practical use/impact.

Fundamental requirement: the transformations need correct/legal.

What is not code generation? (at least in this context)

Code generations (by ML) is very popular:

- ChatGPT, Co-pilot, generative models
- Deepmind's "new" sort algorithm[1] and "new" matrix multiplication[2]
- NLP: code from human languages/commit messages

Code generation in general:

- Compilers: compiler pass

[1] Mankowitz et al, Faster sorting algorithms discovered using deep reinforcement learning

[2] Fawzi et al, Discovering faster matrix multiplication algorithms with reinforcement learning

The (potential) problem with LLMs/generative models

Deepmind[1] found algorithms using unittests, however tests don't guarantee correctness/legality.

- Primary purpose of testing is to check *human* code.
- Writing tests is hard, especially ones that ensure full coverage.
- Not universal: each program needs new unittests.
  - Writing an AI to write unittests is just moving the goalpost (how do we know tests writen by AI ensure correctness/legality).

Example of bad unittests

Original

```
double gold(double input[N]) {
  double result = 0;
  for (int i = 0; i < N; i++)
    result += input[i];
  return result;
}
```

Transformed

```
double cgpt(double input[N]) {
  double result = 0;
  for (int i = N - 1; i >= 0; i--)
    result += input[i];
  return result;
}
```

Unittest

```
void unittest(int kpass) {
  srand((unsigned int)time(NULL));
  double input[N];
  for (int k = 0; k < kpass; k++) {
    for (int i = 0; i < N; i++)
      input[i] = (double)rand();
    compare(input);
  }
}
```

Main

```
int main(int argc, char *argv[]) {
  unittest(10);
  double tricky_input[] = {400000000, 9e-8, 9e-8};
  printf("And now for some tricky input:\n");
  compare(tricky_input);
  return 0;
}
```

```
Equal? yes; delta: 0.00000000000000000000; gold: 2243918836.000000; cgpt: 2243918836.000000;
Equal? yes; delta: 0.00000000000000000000; gold: 4117298770.000000; cgpt: 4117298770.000000;
...
Equal? yes; delta: 0.00000000000000000000; gold: 3835775724.000000; cgpt: 3835775724.000000;
And now for some tricky input:
Equal? no ; delta: 0.00000005960464477539; gold: 400000000.000000; cgpt: 400000000.000000;
```

Representation

One of the first questions we have was: When training the ML model, which representation(s) do we use?

Representations at different compiler passes:

1. Source code
2. Abstract Syntax Tree (AST)
3. Intermediate Represation(s) (IR), e.g. LLVM IR
4. Assembly code
5. Binary code

Other representations:

1. Graphical representations[3] (call flow data flow graph)
2. Polyhedral model

[3] Cummins at al, ProGraML: Graph-based Deep Learning for Program Optimization and Analysis

HPC codes, just the right ratio of difficult

The next question: How to constrain the problemspace, to make it more feasible while still keeping it relevant/impactful?

We target HPC/scientific codes (e.g. stencils, simulations) because:

- The plethora of research papers describing optimisations of HPC codes is evidence that this is not a solved problem.
- HPC codes usually contain deep and complex nested loops, but each loop separately is regular (regular memory accesses and boundaries).
- We have experience with optimising such codes.

Polyhedral model

Why polyhedral? "Best bang for the buck."

- Reasonable restrictions.
- Mathematically provable correctness/legality.
- Compact way to express optimisation opportunities (e.g. parallelism)
- Compact way to express big transformations (e.g. schedule of the tile)

Reasonable restrictions

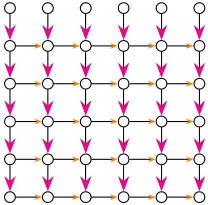
SCoP/SANA[4]: Most is true for HPC codes

- Static control: control does not depend on input data
- Affine: all relevant expressions are (quasi-)affine
- No Aliasing: essentially no pointer manipulations

These restrictions can be relaxed if care is taken.

[4] Verdoolaege, Polyhedral compilation without polyhedra

Working example



Dpendecy in the outer loop, inner loop can be parallel:

```
for(int i = 1; i < N; i++)
  for(int j = 0; j < M; j++)
S1: a[i][j] += a[i-1][j];
```

Components of polyhedral compilation

- SCoP extraction
- Dependency analysis
- Find a schedule θ
- Legality check
- Generate the new source code

Polyhedral basics

Everything can be represented as a matrix

- Statements: S_1 (S_1 is a label). `S1: a[i][j] += a[i-1][j];`
- Statement instances $S_1(i,j)$ (i, j are symbols for integer variables)
- Domain of S_1: $\{S_1(i,j) : 1 \leq i \leq N-1, \quad 0 \leq j \leq M-1\}$ (N is a symbolic constant, unknown but not changing)
- Dependency graph: $e_1 : S_1(i_s, j_s) \rightarrow S_1(i_t, j_t)$ (between statement instances)
  - Notation: s = source (before), t = target (after)
  - Dependency polyhedron: $P_e = \{S_1(i_s, j_s, i_t, j_t) : i_s = i_t - 1, \quad j_s = j_t\}$

Dependency check

Original: $\theta_0 : S_1(i,j) \rightarrow (i,j)$

- Dependencies are maps between event instances: $S_1(i-1,j) \rightarrow S_1(i,j)$
- Schedules are maps from statement instances to (multidimensional) time

Apply the schedule to the range and domain

- Dependency: $S_1(i-1,j) \rightarrow S_1(i,j)$
- Map to time: $(i-1,j) \prec (i,j)$ ($\prec$ is the lexicographic order) or
- $(i,j) - (i-1,j) = (1,0) \succ 0$ OK!
- $\theta(\vec{s}) \prec \theta(\vec{t})$ for the dependency $\vec{s} \rightarrow \vec{t}$
- $\delta(i,j) \succ 0$ where $\delta(i,j) = \theta(i,j) - \theta(i-1,j)$

Expressing Transformations

Swap loops $\theta_1 : S_1(i,j) \to (j,i)$

- Check: $(j,i) - (j,i-1) = (0,1) \succ 0$ OK!
- You can start get θ_1 from scratch, but you can also modify θ_0: in this case $\theta_1 = T \circ \theta_0$ where $T = (i,j) \mapsto (j,i)$. $\theta_1 : S_1(i,j) \xrightarrow{\theta_0} (i,j) \xrightarrow{T} (j,i)$
- The zero in $\delta = (0,1)$ we can parallelise the j loop

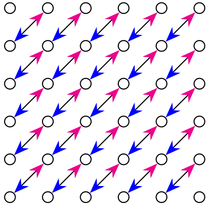
Reverse j $\theta_2 : S_1(i,j) \to (i,-j)$

- Check: $(i,-j) - (i-1,-j) = (1,0) \succ 0$ OK!

Reverse i $\theta_3 : S_1(i,j) \to (-i,j)$

- Check: $(-i,j) - (-(i-1),j) = (-1,0) \not\succ 0$ ILLEGAL!

More transformations



Diagonal from $(0,0)$ $\theta_4 : S_1(i,j) \rightarrow (i+j, j)$:

- Check: $(i+j,j) - (i-1+j,j) = (1,0)$: OK!

Alternative diagonal from $(0,0)$ $\theta_5 : S_1(i,j) \rightarrow (i+j, i)$

- Check: $(i+j,i) - (i-1+j,i-1) = (1,1)$: OK!

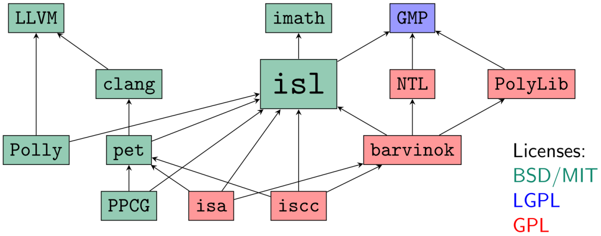
Tiling: $\theta(i,j) = (\lfloor i/T \rfloor, \lfloor j/T \rfloor, i \bmod T, j \bmod T)$

- $(\lfloor i/T \rfloor, \lfloor j/T \rfloor, i \bmod T, j \bmod T) - (\lfloor (i-1)/T \rfloor, \lfloor j/T \rfloor, (i-1) \bmod T, j \bmod T)$
- The delta: $(q_i, 0, r_i, 0)$ where $q_i = \lfloor i/T \rfloor - \lfloor (i-1)/T \rfloor$,
  - here $q_i = 1$ if $i \mid T$ and $q_i = 0$ when when $i \nmid T$
  - $r_i = 1 - q_i T$ which is $1 - T < 0$ if $i \mid T$
  - when $i \mid T : (1, 0, 1-T, 0) \succ 0$; when $i \nmid T : (0,0,1,0) \succ 0$: OK!

Tools

A slide from Verdoolaege, "Polyhedral compilation without polyhedra".

isl and Related Libraries and Tools



isl: manipulates parametric affine sets and relations
barvinok: counts elements in parametric affine sets and relations
pet: extracts polyhedral model from clang AST
PPCG: Polyhedral Parallel Code Generator
iscc: interactive calculator
isa: prototype tool set including derivation of process networks and equivalence checker

Tadashi

Ultimate goal: legality check

- Ask `<random LLM/generative model>` to optimise your code, and have a tool to check the legality of the output the model produced!
- Very difficult: which original statement corresponds to which transformed statement?

正:Tadashi

- Uses polyhedral.
- Checks the legal of any schedule.
  - Quite easy to do with the ISL library.
- Generates[5] the transformed code (if the transformation is legal).

[5] work in progress.

Restrictions and relaxations

The restrictions

1. Polyhedral is oblivious to the statements
2. Polyhedral is oblivious to the hardware
3. Bending the SANA/SCoP rules

And how to bend them

1. More involved data flow analysis
2. The δ encodes info about parallelism and data locality
   - Transformations in and after polyhedral
3. Approximations and/or pw_qpolynomial

A framework to automate the process