Towards Self-Verification in Finite Difference Code Generation

Jan Hückelheim[1], Ziqing Luo[2], Fabio Luporini[1],Navjot Kukreja[1], Michael Lange[1], Gerard Gorman[1], Stephen Siegel[2], Matthew Dwyer[3], Paul Hovland[4]

1: Imperial College London, 2: University of Delaware,
3: University of Nebraska, 4: Argonne National Laboratory

November 12, 2017

Overview

Let's unpick the title:

Towards Self-Verification in Finite Difference Code Generation

1. Code generation
2. Self-verification
3. Test Case
4. *Towards*?

Code Generation

- Motivation: combine productivity and performance
- Define problem in high-level domain-specific language
- Generate high-performance code from that
- Goal: Separation of concerns
  - Domain experts should only care about modelling the problem, perhaps choosing a solution method.
  - Domain experts should not have to implement parallelism, care about cache use, data transfer, ...
  - HPC/compiler specialists should only care about efficiency
  - Porting to a different computer architecture should only affect the HPC people
  - Modifying the problem to be solved should only affect the domain experts

Devito

- Devito is a code generation framework, developed mainly for seismic imaging
- Free and open source
- http://www.opesci.org/devito-public

Devito

- Efficient stencil code through code generation
- Problems specified in python
- Example: Convection-Diffusion equation

$$\frac{\partial u}{\partial t} = d\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - a\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right).$$

- In Devito, this can be written as
  `eqn=Eq(u.dt,d*(u.dx2+u.dy2)-a*(u.dxl+u.dyl))`
- Devito will discretise the equation
- Devito will generate fast, parallel C code to solve the equation
- Parallelisation, SIMD vectorisation, cache blocking, and many other things happen under the hood, and the user does not need to know about any of this
- Solving a different equation? The code will still be fast.

Devito

Want to know more? See this talk:

- Friday at 11:10am, WOLFHPC 2017 Workshop
- "Optimised finite difference computation from symbolic equations"
- Fabio Luporini, Michael Lange, Navjot Kukreja, Mathias Louboutin, Charles Yount, Jan Hueckelheim and Gerard Gorman

Self-Verification

- Potential users of Devito and other code generation tools worry about correctness
- "If I don't write the code myself, and I don't see it, how can I trust it?"
- Testing has limitations everywhere, but especially here:
  - Generated code different on every platform
  - Generated code different for every problem
  - Generated code depends on user settings, environment variables, command line flags, ...

Self-Verification

- Idea: Integrate verification into code generation tool
- Whenever code is generated, it is formally verified for correctness (always, or e.g. if some debug command line flag is used)

CIVL: Concurrency Intermediate Verification Language

- a framework for verifying concurrent C programs
  - uses symbolic execution and model checking
  - Fortran support in progress
- supports MPI, CUDA, OpenMP, Pthreads
  - can be extended to support more concurrency dialects
- properties checked
  - generic: free of illegal pointer dereference, memory leak, ...
  - API specific: no race condition for OpenMP shared variable access, ...
  - application specific
    - user specified assertions, e.g., $assert(g \neq 0)$
    - equivalence of two programs
- free and open source
  - download at `http://vsl.cis.udel.edu/civl`
  - try online at `http://civl.cis.udel.edu/app`

Test Case

- Convection-diffusion equation as seen earlier

$$\frac{\partial u}{\partial t} = d\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - a\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right).$$

- Check if Devito generates a solver without violations of safety properties
- Use CIVL `compare` with simple baseline implementation
- CIVL will check if results match for any arbitrary input (up to some size)
- OpenMP loops are sequentialised if possible
- Bounds used: e.g. $n = 5$ (11s), $n = 7$ (200s)

CIVL driver program

```
$input int __TS = 2;
$input int  __BOUND = 7;
$input int __TIMES, __XS, __YS;
$assume(1 < __TIMES && __TIMES <= __BOUND);
$assume(1 < __XS && __XS <= __BOUND);
$assume(1 < __YS && __YS <= __BOUND);
$input float u_in[__TS][__XS][__YS];
$output float u_out[__TS][__XS][__YS];

int main(int argc, char** argv) {
  float u[__TS][__XS][__YS];
  $elaborate(__TS);
  $elaborate(__TIMES);
  $elaborate(__XS);
  $elaborate(__YS);
  memcpy(u,u_in,sizeof(double)*__TS*__XS*__YS);
   // This is the code we verify:
  Kernel(u,__TS,__XS,__YS,__TIMES);
  memcpy(u_out,u,sizeof(double)*__TS*__XS*__YS);
  return 0;
}
```

Simple solver

```
for(int time = 0; time < time_size - 1; time++) {
  for (int x = 2; x < x_size - 1; x += 1) {
    for (int y = 2; y < y_size - 1; y += 1) {
```

- This is just the loop head
- The body is important, too, but too long for the slides (see the paper)

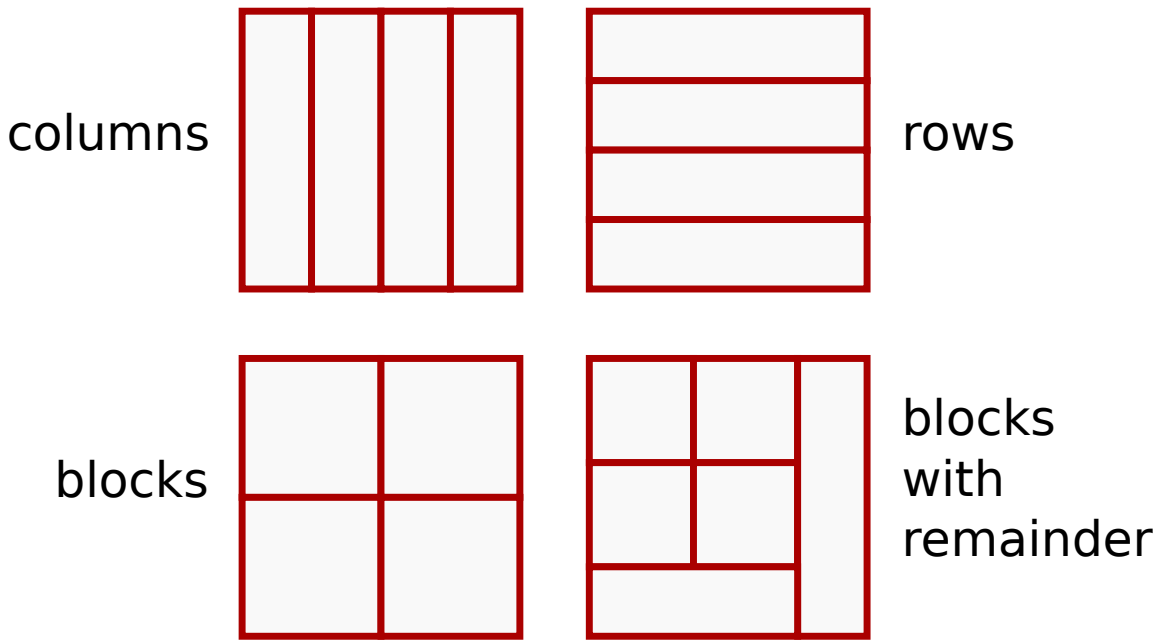
Advanced solver

```
for (int time = 0, t0 = (time)%(2), t1 = (time + 1)
    %(2); time < time_size - 1; time += 1, t0 = (time
    )%(2), t1 = (time + 1)%(2)) {
  #pragma omp parallel
  {
    #pragma omp for schedule(static)
    for (int x_block = 2; x_block < x_size - (x_size
        - 3)%(x_block_size) - 1; x_block +=
       x_block_size) {
      for (int y_block = 2; y_block < y_size - (
          y_size - 3)%(y_block_size) - 1; y_block +=
          y_block_size) {
```

- The equivalent, optimised loop head
- The code uses parallelism (multi-core and SIMD), blocking, ...

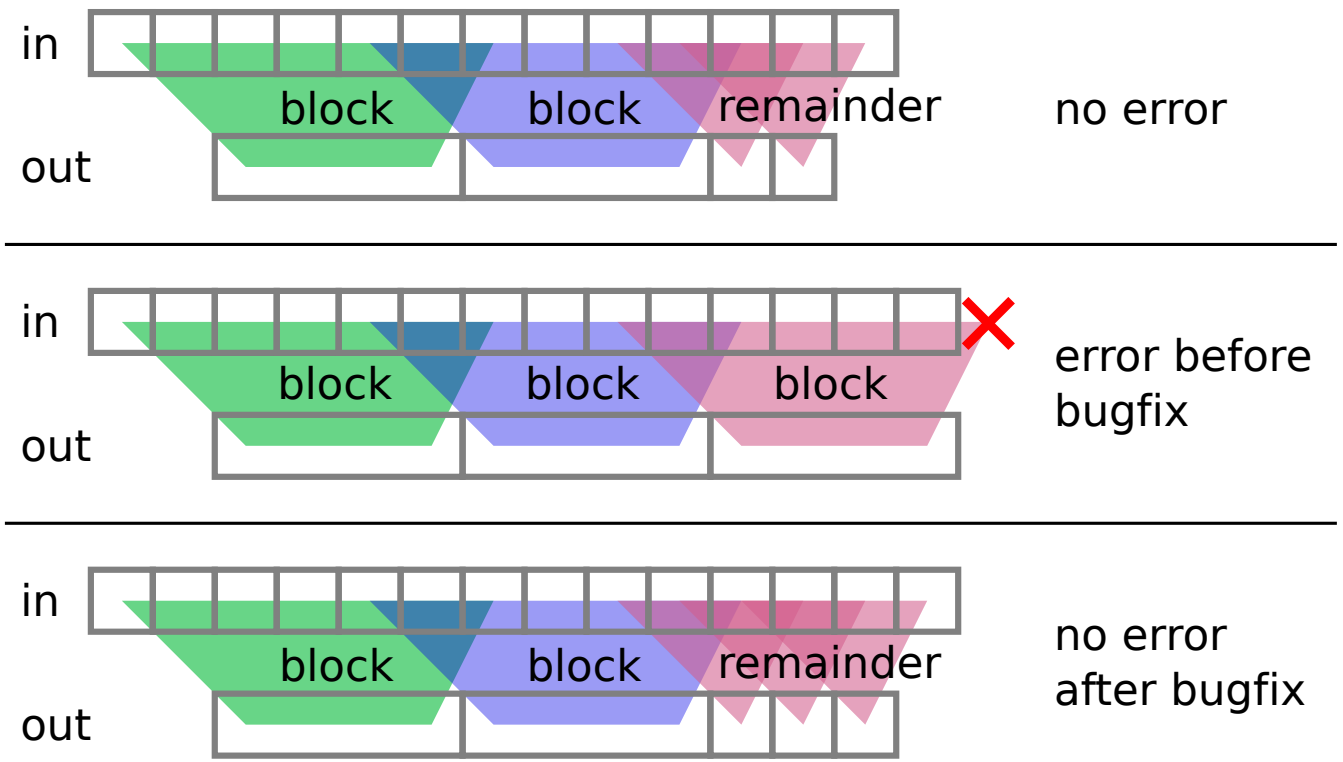
Short excursion: what is cache blocking?

- Devito uses blocking to reduce cache misses
- Optimal block size depends on problem, code, hardware
- Auto-tuning is used to find best block size



Bug found

- The blocked code had an out-of-bounds read access if domain_size−2 % block_size = 0
- Testing never found this, verification did: CIVL found a counterexample that causes this problem



Why "Towards"?

Verification is not fully automatic yet, and not complete:

- Some manual changes to the code to work around unsupported features in CIVL (some type casts, `restrict` keyword, etc.
- We assume that neither the C compiler nor other parts of the environment introduce new bugs
- We do not model or test for roundoff errors
- We only show correctness for problem sizes up to some limit.
- We assume that the baseline is correct, but: We could attempt to verify more properties, e.g. convergence order.
- We could extend this to other code generation frameworks.

Future work.

Thank you

Questions?