How OPS (Optimizing Parallelizing System) May be Useful for Clang

SECR-2017

Lev R. Gervith, Sergey A. Guda, Denis V. Dubrov, Ruslan A. Ibragimov, Elena A. Metelitsa, Yury M. Mikhailuts, Artyom E. Paterikin, Victor V. Petrenko, Ilya R. Skapenko, Boris Ya. Steinberg, Oleg B. Steinberg, Vladislav A. Yakovlev, Mikhail V. Yurushkin

Southern Federal University, Rostov-on-Don, Russia

October 20, 2017

Optimizing parallelizing system (OPS)

OPS at a glance

- High-level internal representation: “Reprise”;
- Frontend (Clang based);
- Analysis:
  - Dependencies graph, Alias analysis, Computations graph, ...
- Transformations:
  - Recurrent loops parallelizing, Loop unrolling, Loop fusion, Loop nesting, Loop interchange ...

OPS at a glance (end)

- GUI for testing purposes;
- Data visualization:
  - Dependencies graph, ...
- Backends:
  - MPI, OpenMP, CUDA, VHDL, Clang, ...

Generating high-level code

- Automatic parallelization:
  - CUDA
  - OpenMP
  - MPI
  - VHDL
- Optimizing memory usage: tiling.

Clang + OPS integration

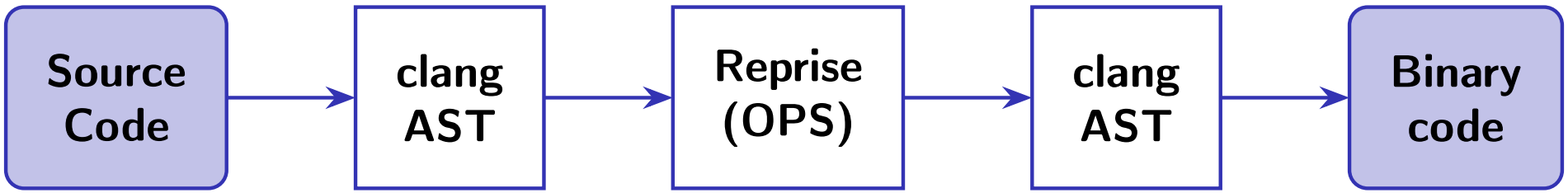


Figure 1: injecting OPS inside Clang

Low-level IR vs. High-level IR

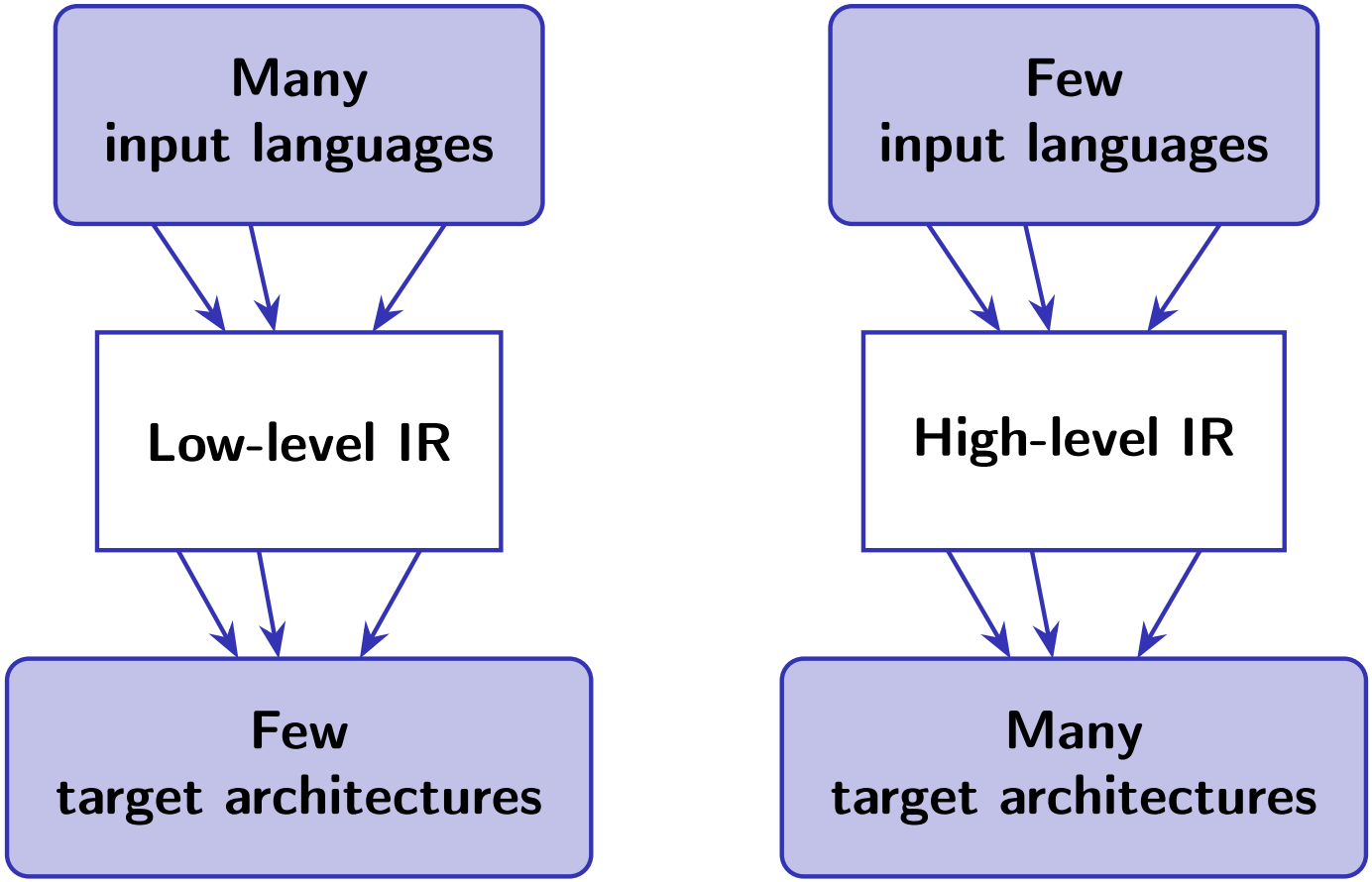


Figure 2: comparison of low-level and high-level internal representations

Advantages of using OPS

- Code generation for accelerators
  - GPU
  - FPGA
- Block data placement
  - shared memory
  - distributed memory
- Parallel programming visual aid: dependency visualization in terms of original source code.
- Dialog compilation

Block array placement

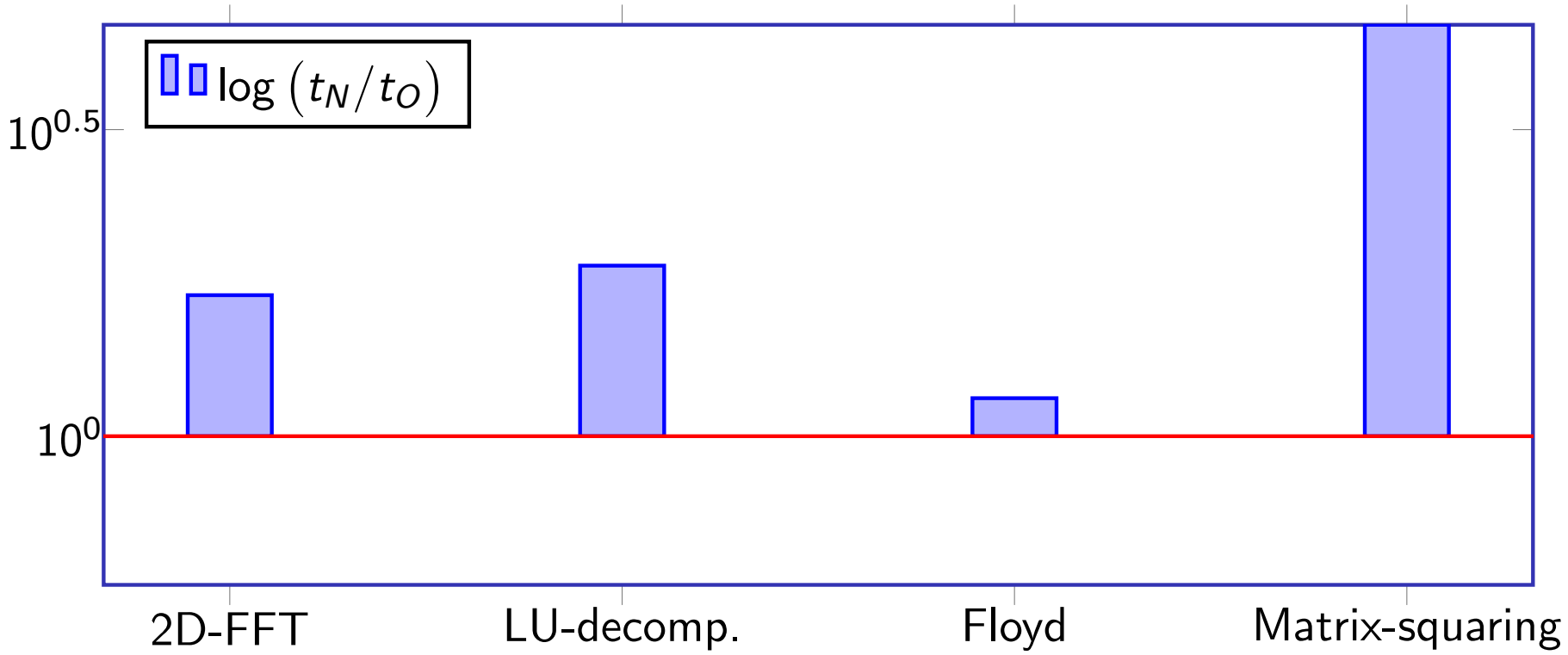


Figure 3: running times for algorithms with and without block placement in shared memory

Block affine array placement in distributed memory

Result

Automatically generated MPI code + block affine data placement.

Table 1: solving 3D Dirichlet problem for Poisson equation with iterative Jacobi algorithm

	Running time, s			
Size	1 node	2 nodes	4 nodes	8 nodes
$128 \times 128 \times 128$	38.25	19.74	10.9	5.97
$256 \times 256 \times 256$	310.19	165.64	87.11	48.64
$384 \times 384 \times 384$	1078	697.95	356.9	190.12
$512 \times 512 \times 512$	2786.47	1432.2	776.14	418.38

Graphics accelerator

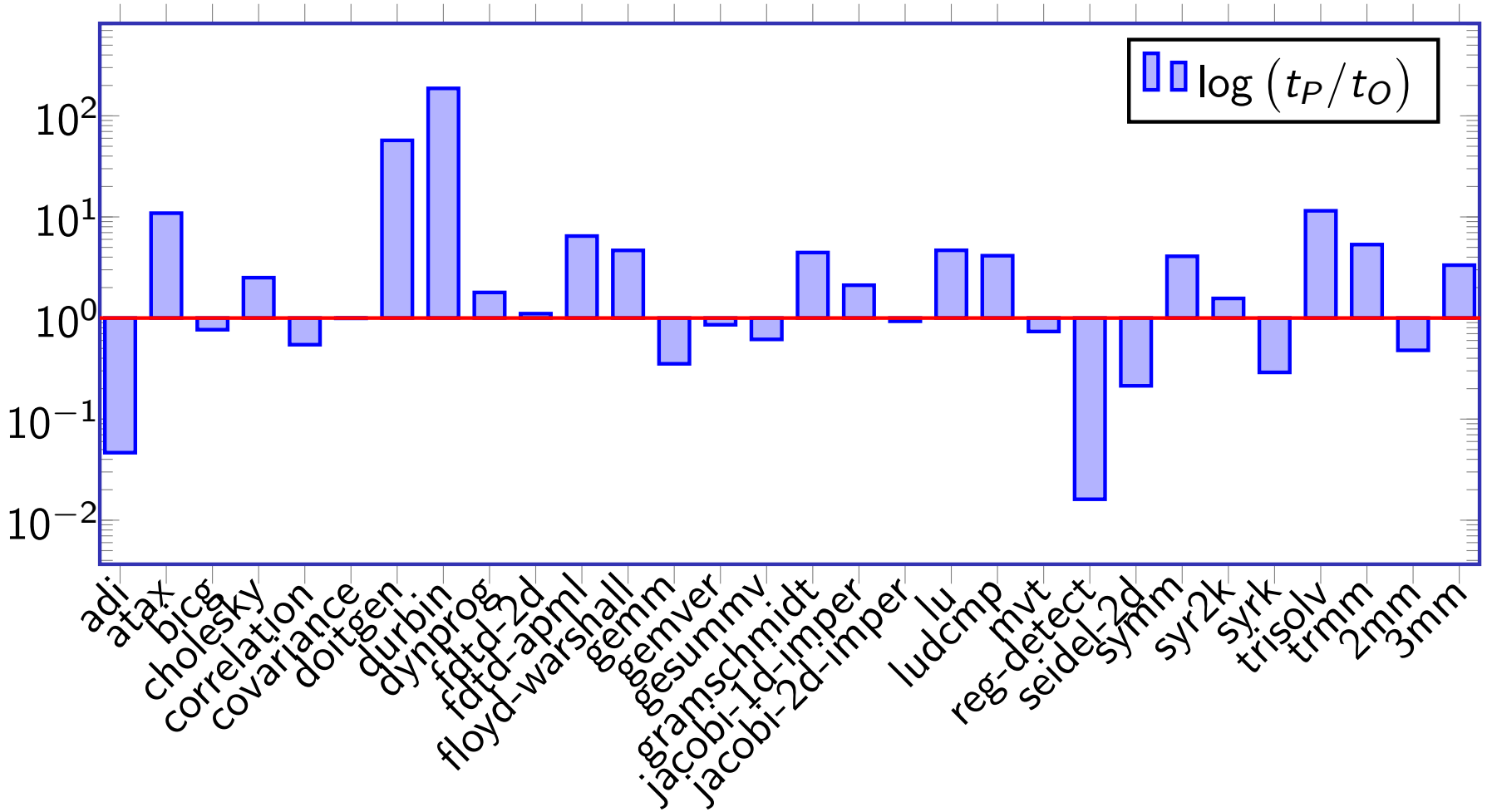
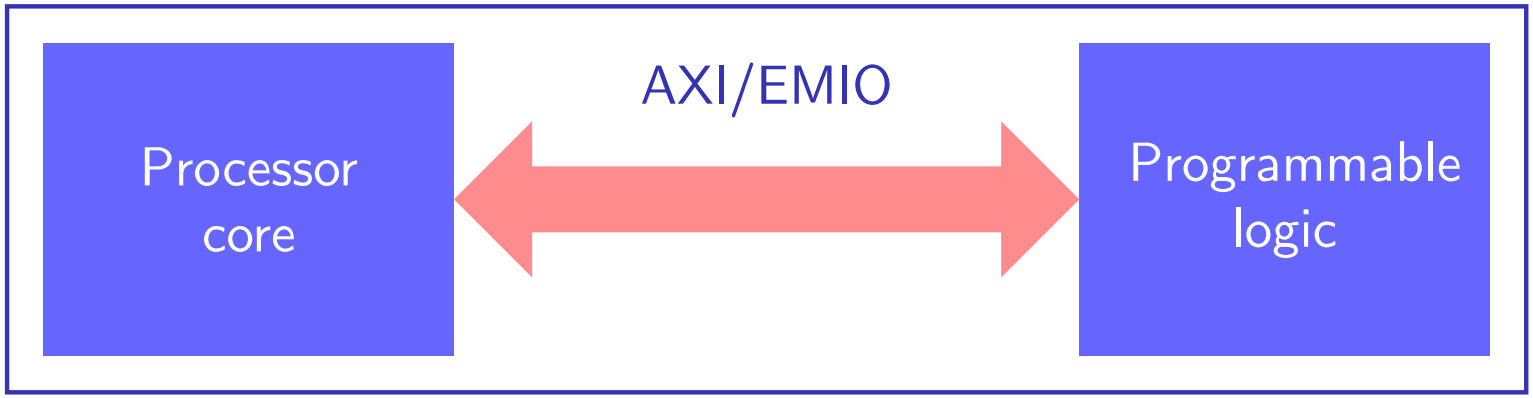
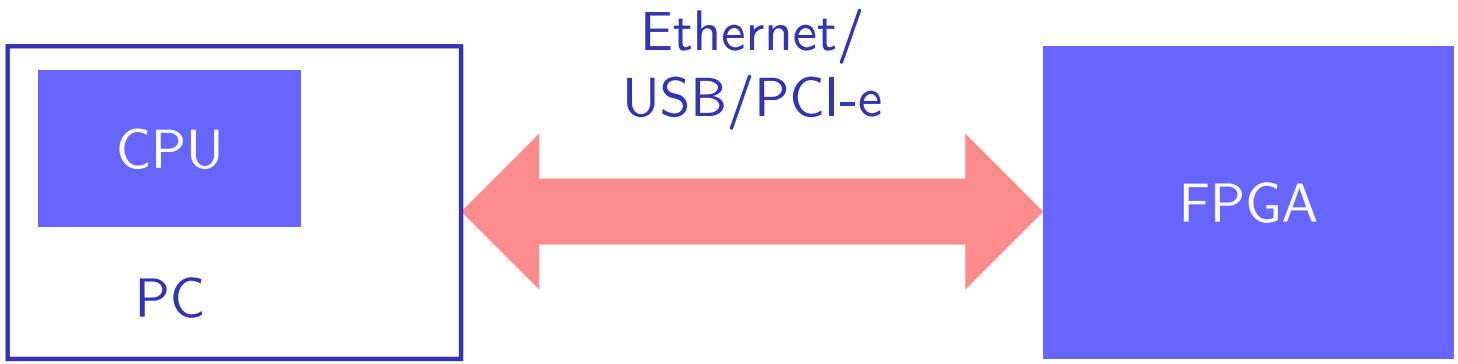


Figure 4: comparison of OPS-based solution with PPCG (`http://ppcg.gforge.inria.fr/`)

FPGA accelerator



a) on one core



b) on separate cores

Figure 5: the hybrid compute systems with FPGA/CPU

Dependency visualization

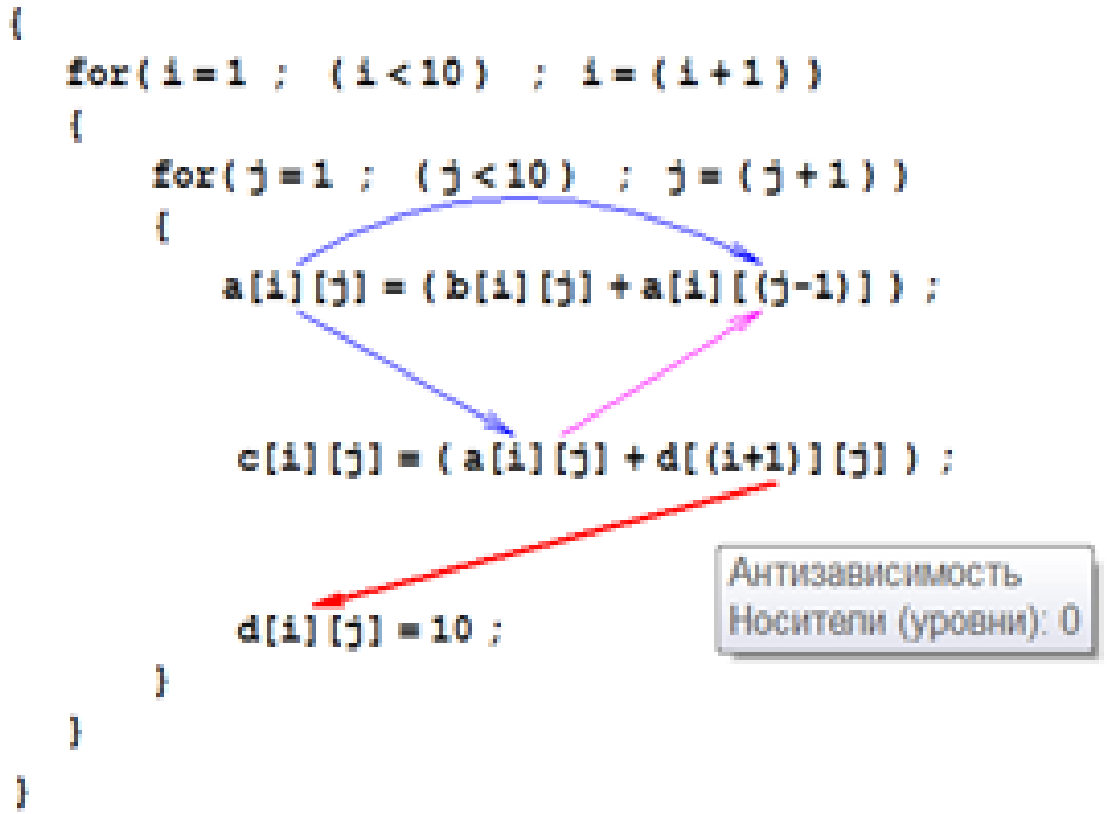
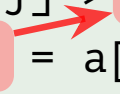


Figure 6: parallel programmer's training simulator output

Dependency refinement in dialog

Example (Floyd algorithm)

```
for (k = 0;  k < n; ++ k)
  for (i = 0; i < n; ++ i)
    for (j = 0; j < n; ++ j)
      if (a[i][j] > a[i][k] + a[k][j])
        a[i][j] = a[i][k] + a[k][j];
```



- Neither loop can be automatically parallelized due to data dependency.
- Actually, the dependency is not realized if `a[i][i] >= 0`.
- The dialog compiler may ask the question to the programmer.

OpenOPS

OpenOPS source code	`https://github.com/OpsGroup/open-ops`
OPS website	`http://www.ops.rsu.ru/`
Web auto-parallelizer	`http://ops.opsgroup.ru/en/`