Understanding PolyBench/C 3.2 Kernels

Tomofumi Yuki
INRIA

PolyBench

- Collection of small, polyhedral, kernels
- Aimed to uniformize experimental validation
  - How to performing timing
  - Same variant of “matrix multiply”
- C/Fortran/GPU implementations
- Being used by many people

INVENTEURS DU MONDE NUMÉRIQUE

PolyBench

- Collection of small, polyhedral, kernels
- Aimed to uniformize experimental validation
  - How to performing timing
  - Same variant of “matrix multiply”
- C/Fortran/GPU implementations
- Being used by many people
- But,
  - **description of the kernels are lacking**

`lu` and `ludcmp`

- Description (from PolyBench web)
  - `lu`: LU Decomposition
  - `ludcmp`: LU Decomposition
  - no additional description in source

`lu` and `ludcmp`

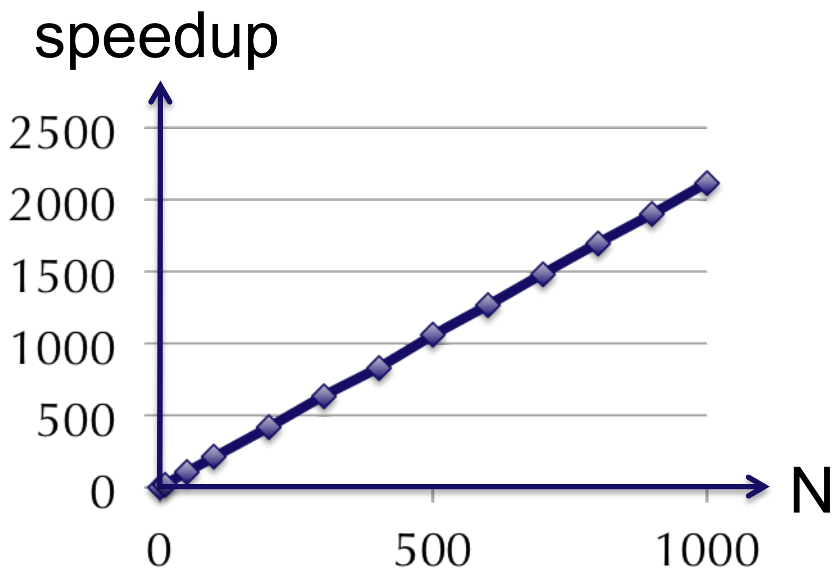
- Description (from PolyBench web)
  - `lu`: LU Decomposition
  - `ludcmp`: LU Decomposition
  - no additional description in source
- Only one-line description for many kernels
- Many complications are not obvious
  - memory allocation
  - legal input data set
  - bugs and questionable properties

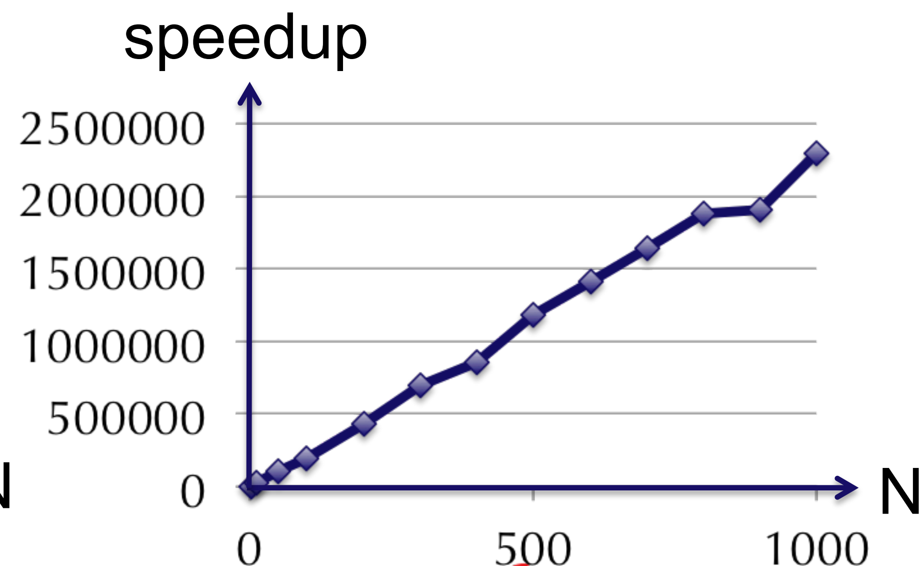
PolyBench as Specification

- Equational/Mathematical specification of the computation should be *the* PolyBench
  - expected input/output
  - context—typical use case
- Reference implementations should:
  - implement the same computation
  - clearly explain implementation decisions
  - algorithms may be different

Extreme Example

- 2 kernels exhibit *parametric* speedup
  - excessive (single assignment) memory
  - redundant work





Redundant Work

- Can be legitimate target of optimization
  - e.g., UNAfold, MSS
- These two kernels have *artificial* outer loop

```
for (n=0; n<N; n++) {
  //init
  …

  //compute
  …
}
```

What has been done so far

- Preliminary specification
  - `polyweb.irisa.fr/polybench-report.pdf`
- List of bugs and questionable behaviors
- PolyBench/Alpha
  - Executable specification

Using different starting points

- We have 3 implementations of PolyBench
  - C1, C2, and Alpha
  - all versions implement the same specification
- Performance of `gemm` (on the same machine)
  - Tool A performs best with PolyBench/C1
  - Tool B performs best with PolyBench/C2
  - Tool C performs best with PolyBench/Alpha
- How should we evaluate the tools?

Impact of Implementation

- Implementation decisions significantly influence performance of tools
- Ex1: in-place memory allocation
  - ☺ memory expansion + parallelization
  - ☹ memory contraction
- Ex2: single assignment code
  - ☺ easier for compiler to analyze
  - ☹ terrible performance without contraction
  - ☹ when does compiler see SA code?

Discussion

- Not restricted to PolyBench!