A Hybrid Approach to Parallel Pattern Discovery in C++

C. Brown, V. Janjic,
A. Barwell, J. Thomson
University of St Andrews, UK

R. Castañeda Lozano,
M. Cole, B. Franke
University of Edinburgh, UK

J.D. Garcia-Sanchez,
D. Del Rio Astorga
Universidad Carlos III de Madrid, Spain

K. MacKenzie
IOHK

This research has been funded by the Engineering and Physical Sciences Research Council, UK (EP/P022642/1)

Goal

To help programmers parallelize C++ programs where fully automated methods fail:

- **where** should parallelism be introduced?
- **what** type of parallelism should be introduced?

Parallel Patterns

- Model and implement parallel constructs
- Benefits: simplicity, portability, performance
- Here: *map* and *reduce* patterns

```
float dot_product(Vector<float> &a, Vector<float> &b) {
    auto dotprod = MapReduce<2>(
        [] (float a, float b) {return a * b;},
        [] (float ab1, float ab2) {return ab1 + ab2;});
    return dotprod(a, b);
}
```

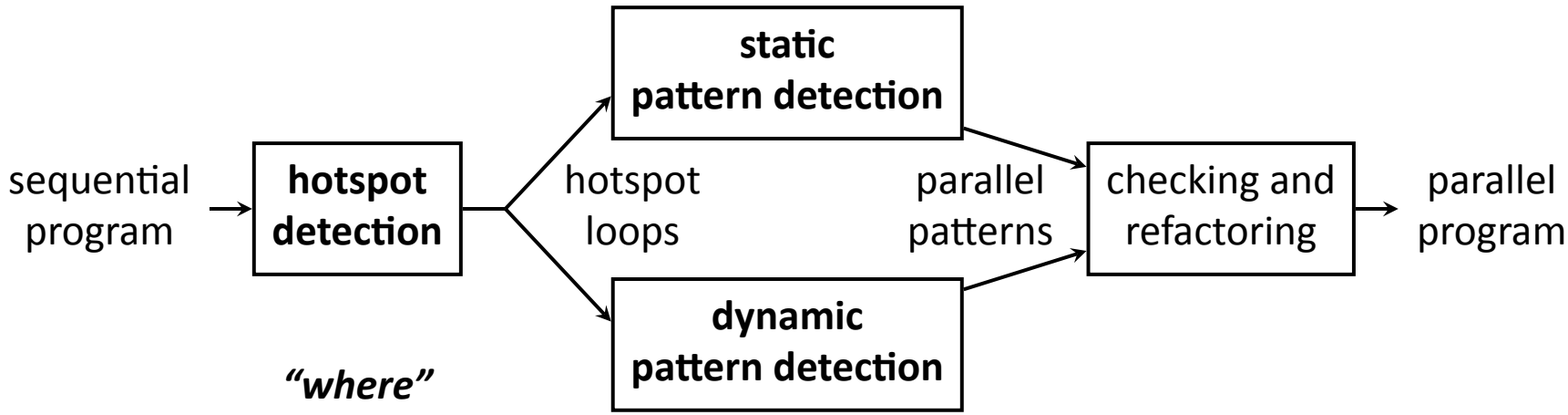
(from `https://www.ida.liu.se/labs/pelab/skepu/#mapreduce`)

Contributions

- Approach to find parallelism in C++ programs
- Detects most of the *hot* patterns
  - *hot*: computationally intensive
- Results in successful parallelization (often)

Approach

Parallel Pattern Analyzer Tool
[Del Rio Astorga et. al, IJHPCA 2018]



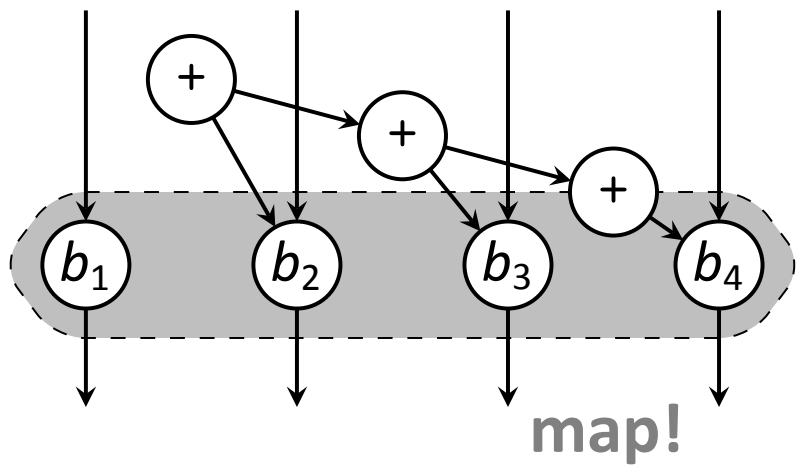
(new trace-and-match method)

Dynamic Pattern Detection

1. Tracing: Dynamic Data-Flow Graph (DDG)
   - nodes: executions of instructions/loop bodies
   - arcs: flow of data
2. Matching: graph pattern matching on the DDG

```
for (i=1; i<=N; i++) {
  ...
  // iteration b_i
  ...
}
```

(a) C++ loop



(b) DDG of the loop for `N = 4`

Experimental Evaluation

- Five sequential C++ programs

  Convolution, Mandelbrot, Ant Colony,
  Black-Scholes, Transfil

- Typical parallel programming benchmarks
- Expert pattern detection as ground truth
- Parallelization: just the hottest pattern detected

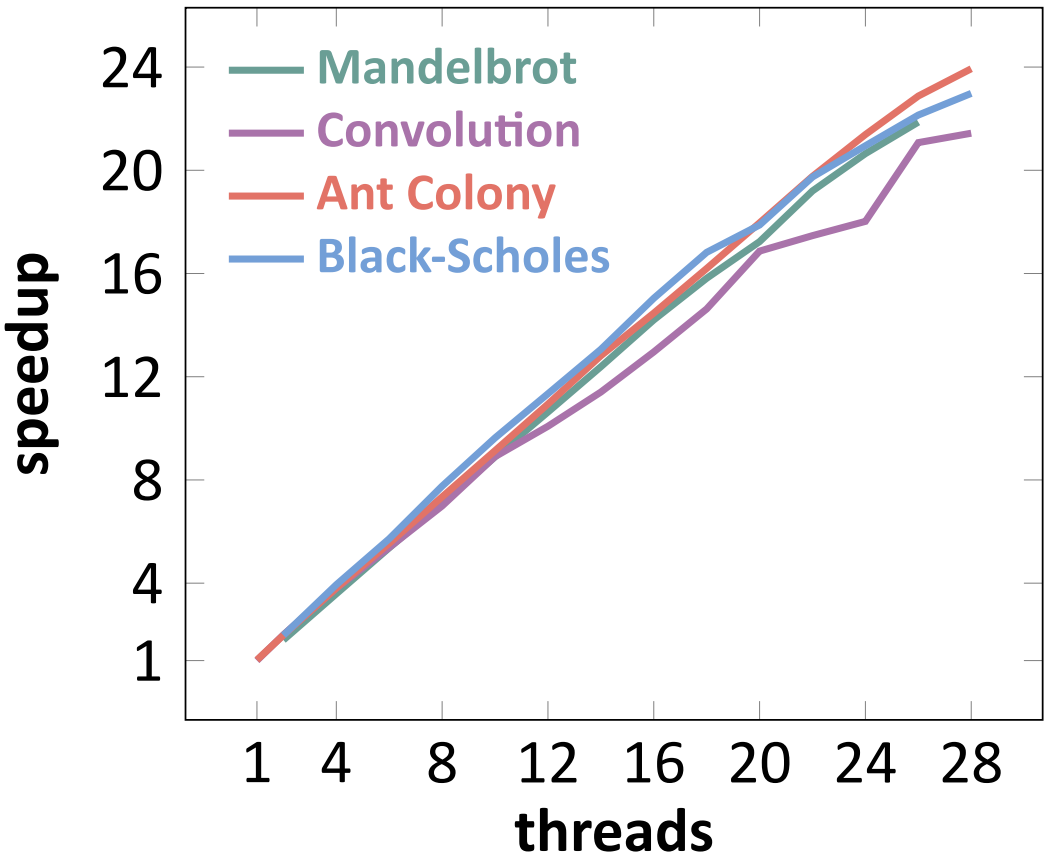
Performance of Pattern Detection

- Detection performance measured in:
  - **accuracy:** loops classified correctly
  - **precision:** real patterns among detected patterns
  - **recall:** detected patterns among real patterns

analysis	accuracy	precision	recall
static	**79 %**	**100 %**	54%
dynamic	**79 %**	82%	**69 %**

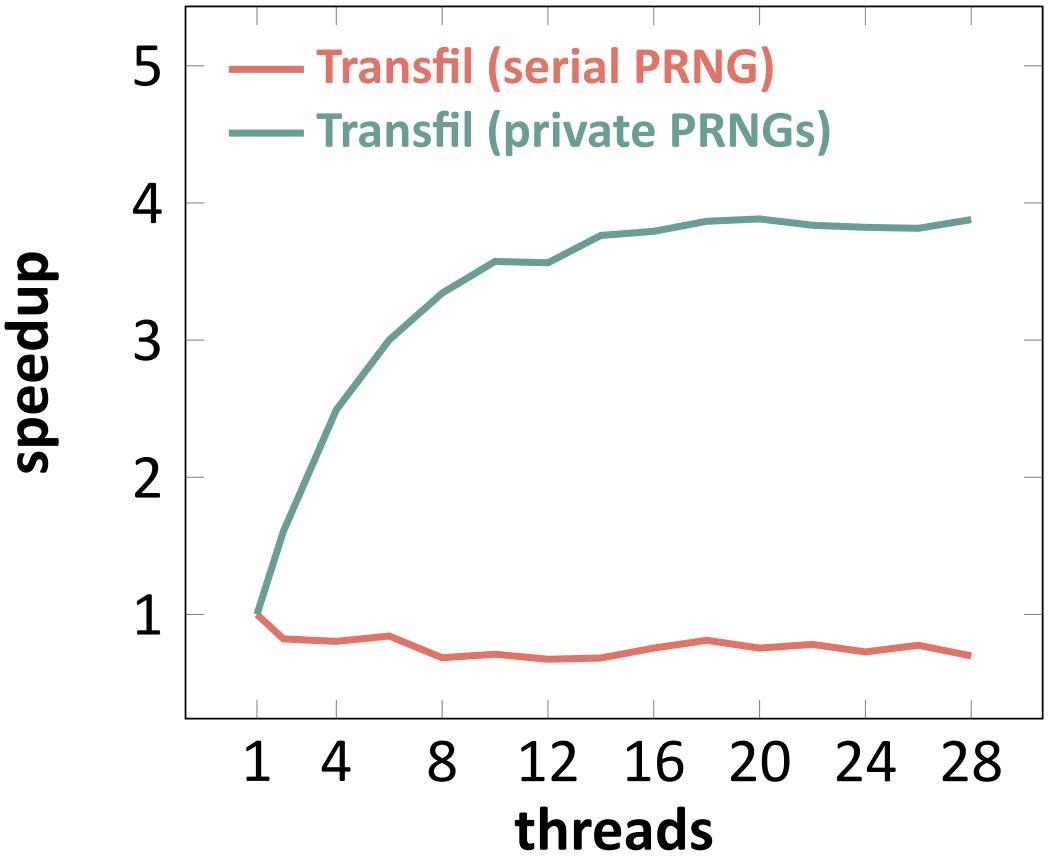
- Similar accuracy, complementary strengths
- Recommendation: apply serially
  - reduces human checking effort by 25%

Performance of the Parallelized Code



- The approach detects the hottest patterns

Performance of the Parallelized Code



- No speedup in the initial parallelization
  - bottleneck: serial random generator (PRNG)
- Still hot pattern: speedup after privatization
- Bottleneck analysis outside of our scope

Conclusion

- Approach to find parallelism in C++ programs
- Hybrid static/dynamic analysis
- Detects hot *map* and *reduce* patterns
- Results in successful parallelization (often)
- Lots of future work
  - more patterns
  - better detection performance
  - identification of other bottlenecks
  - …