#CnC'21 Workshop

PLUSS: Parallel Locality Analysis using Static Sampling

Fangzhou Liu, Dong Chen, Wesley Smith, Chen Ding

University of Rochester

Motivation

- Two effects of caching:
  - Collaborative: One thread brings the data that will be later reused by other threads.
  - Interfering: One thread brings the data but never reused by other threads.

Motivation

- We classify existing models based on their analysis approach and assumption.

	w/o DATA SHARING	w/ DATA SHARING
Trace-based Analysis	RD [Chandra et al. HPCA'05, Xu et al. ISPASS'10] fp [Brock et al. ISMM'18]	CRD [Jiang et al. PACT'10] PCT-RD [Li et al. LCPC'17] sfp [Luo et al. PPoPP'17] PPT-SASMM[Barai et al. MEMSYS'20]
Static Analysis	[Tolubaeva et al. IPDPSW'14]* PLUSS [This work]	

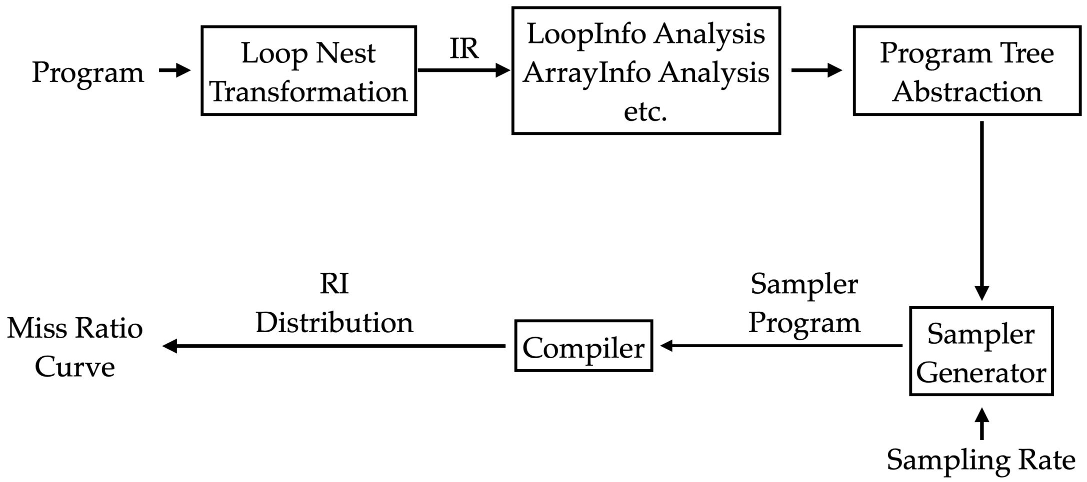
Key Features of PLUSS

- Computing RD has higher costs than RI.
  - $O(n\log n)$ vs. $O(n)$. [Hu et al, TOC’18, Yuan et al. TACO’19]
- Trace collection has high overhead.
  - PIN/Cachegrind collects the memory access trace using a global lock.
  - PPT-SASMM stores the generated trace, which consumes 967MB – 4.2GB space.

- Modeling the cache performance using RI.
- Shallow Execution
  - Address tracing only
- Lock-free.
  - Pre-defined Interleaving [Arafa et al. ICS’20, Barai et al. MEMSYS’20]
- No trace storing.

Background

- SPS [Chen et al. PLDI'18] analyzed the program structure through the intermediate representation (IR) and generates a special piece of code, named sampler.
- The sampler collects RIs for each reference using static sampling.
    - For each reference, the sampler randomly choose an iteration from the iteration space, then it follows the program flow until it find a reuse.



Background

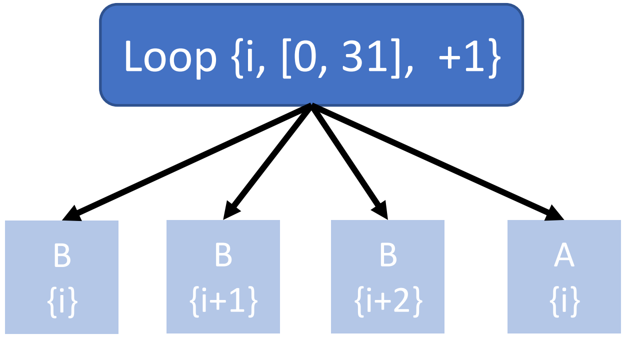
1. Sample i = 5 for `B[i+1]`
2. Sampler begin to traverse i in range [5, 31]

Shallow Execution

```
// i = 5
B[5], B[6], B[7], A[5]
// i = 6
B[6], B[7], B[8], A[6]
// i = 7
B[7], B[8], B[9], A[7]
......
```

3. `B[i]` at iteration i = 6 forms a reuse with `B[i+1]`, with RI = 3; back to step 1.

```
void kernel_jacobi_1d(double *A, double *B) {
  int i;
  for (i = 0; i < 32; i++) {
    A[i] = (B[i] + B[i+1] + B[i+2]) / 3.0;
  }
}
```



When parallelized by OpenMP directives

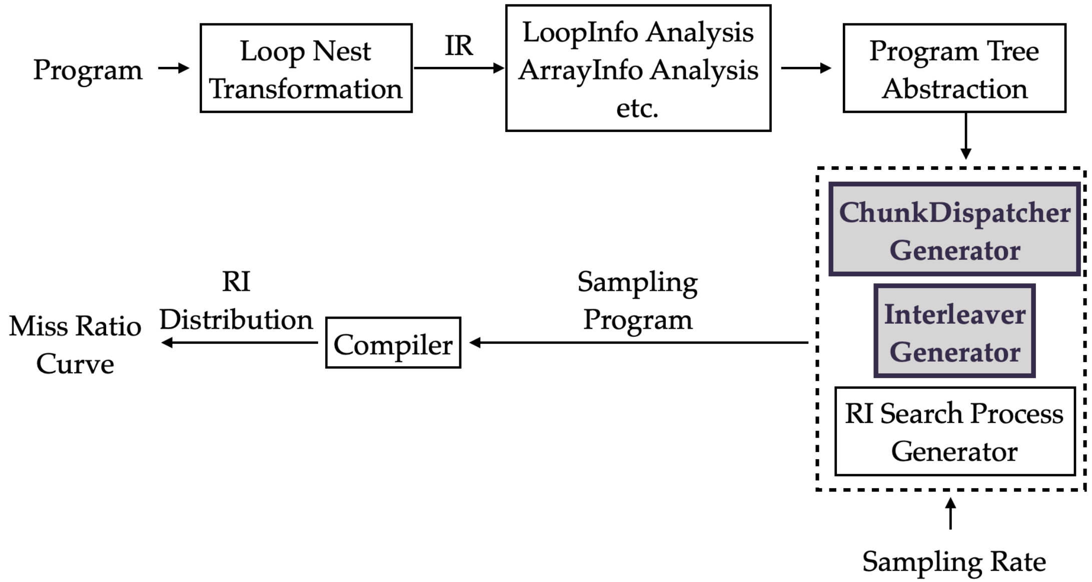
- OpenMP directives indicates the parallel loop will be separated into chunks, each has 4 iterations; These chunks will be distributed to 4 threads using the OpenMP static scheduling algorithm.

```
T0: [0, 3], [16, 19]
T1: [4, 7], [20, 23]
T2: [8,11], [24, 27]
T3: [12, 15], [28, 31]
```

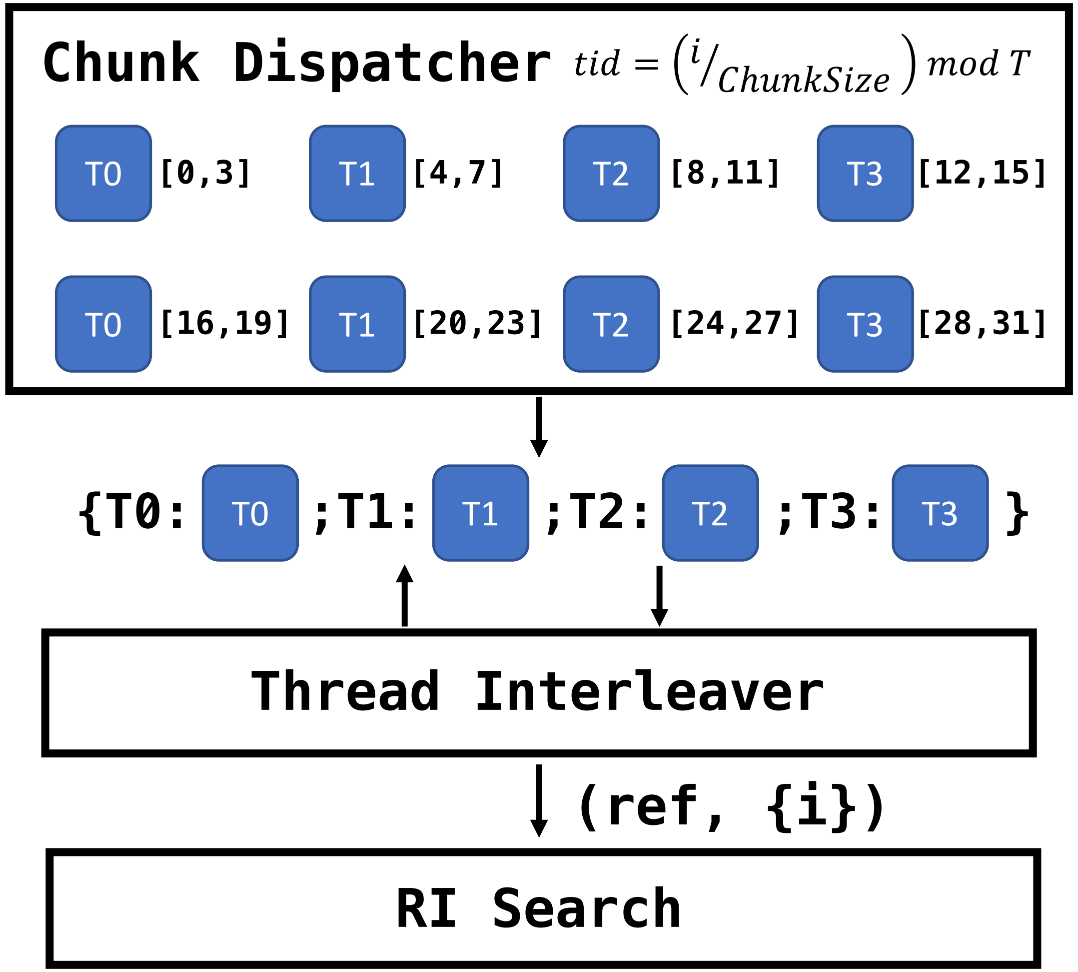
```
void kernel_jacobi_1d(double *A,
double *B) {
  int i;
  #pragma omp parallel num_threads(4)
  {
  #pragma omp for schedule(static, 4)
  for (i = 0; i < 32; i++) {
    A[i] = (B[i] + B[i+1] + B[i+2])
/ 3.0;
  } // end of for loop
  } // end of #pragma omp parallel
} // end of kernel_jacobi_1d
```

PLUSS Working pipeline

- PLUSS adds two components in the Sampler CodeGen module to handle the loop parallelization.
  - *Chunk Dispatcher*: Generates chunks & Does chunk-to-thread mapping
  - *Interleaver*: Simulates the thread interleaving.



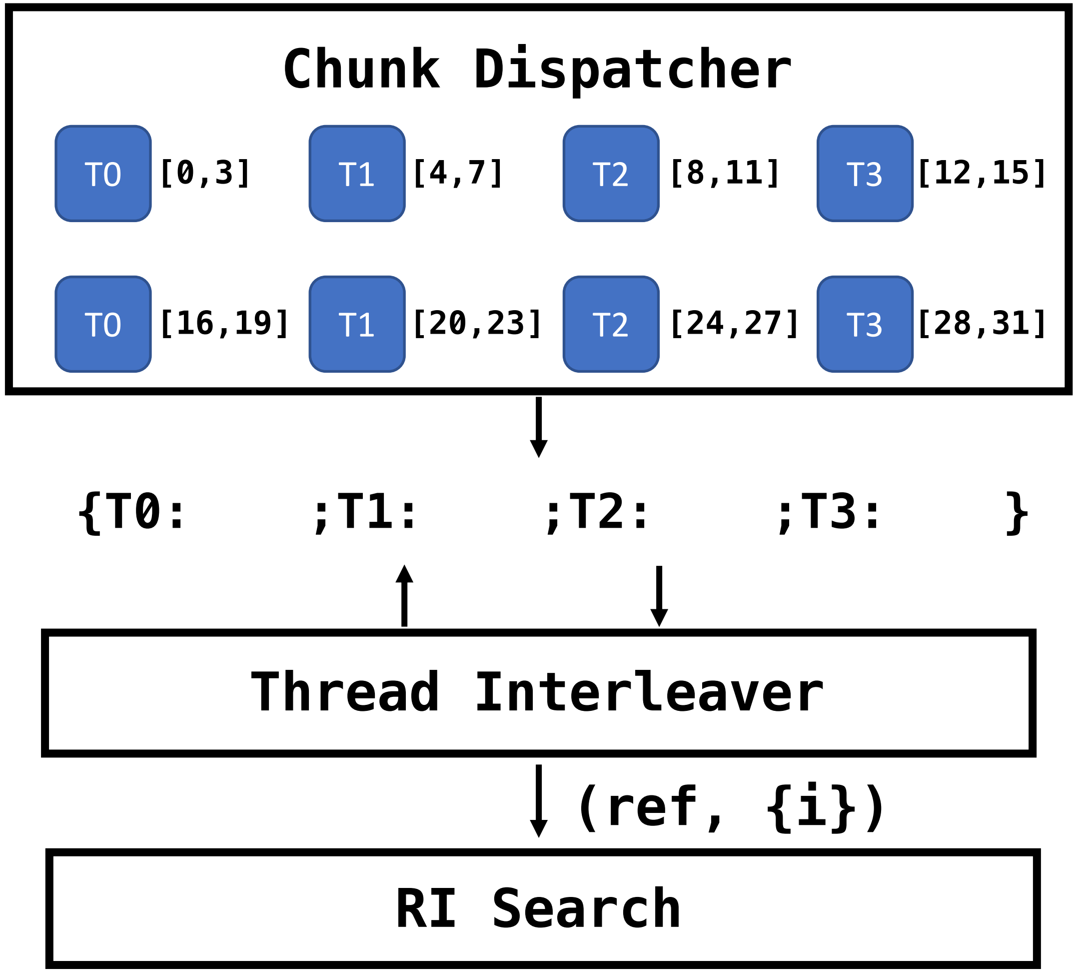
Putting together




```
void kernel_jacobi_1d(double *A,
double *B) {
  int i;
  #pragma omp parallel num_threads(4)
  {
  #pragma omp for schedule(static, 4)
  for (i = 0; i < 32; i++) {
    A[i] = (B[i] + B[i+1] + B[i+2])
/ 3.0;
  } // end of for loop
  } // end of #pragma omp parallel
} // end of kernel_jacobi_1d
```

Putting it all together

1. Sample i = 5 for B[i+1], 5 is the second iteration of the first chunk of T1.
2. The `Interleaver` start traversing the second iteration of the first chunk in each thread.
   B[1], **B[6]**, B[10], B[14], // B[i+1]
   B[3], B[7], B[11], B[15], // B[i+2]
   A[1], A[5], A[9], A[13], // A[i]
   B[2], **B[6]**, B[9], B[12], // B[i]
   ……
3. `B[i+1]` forms a reuse with `B[i]`, with RI = 12, back to 1.

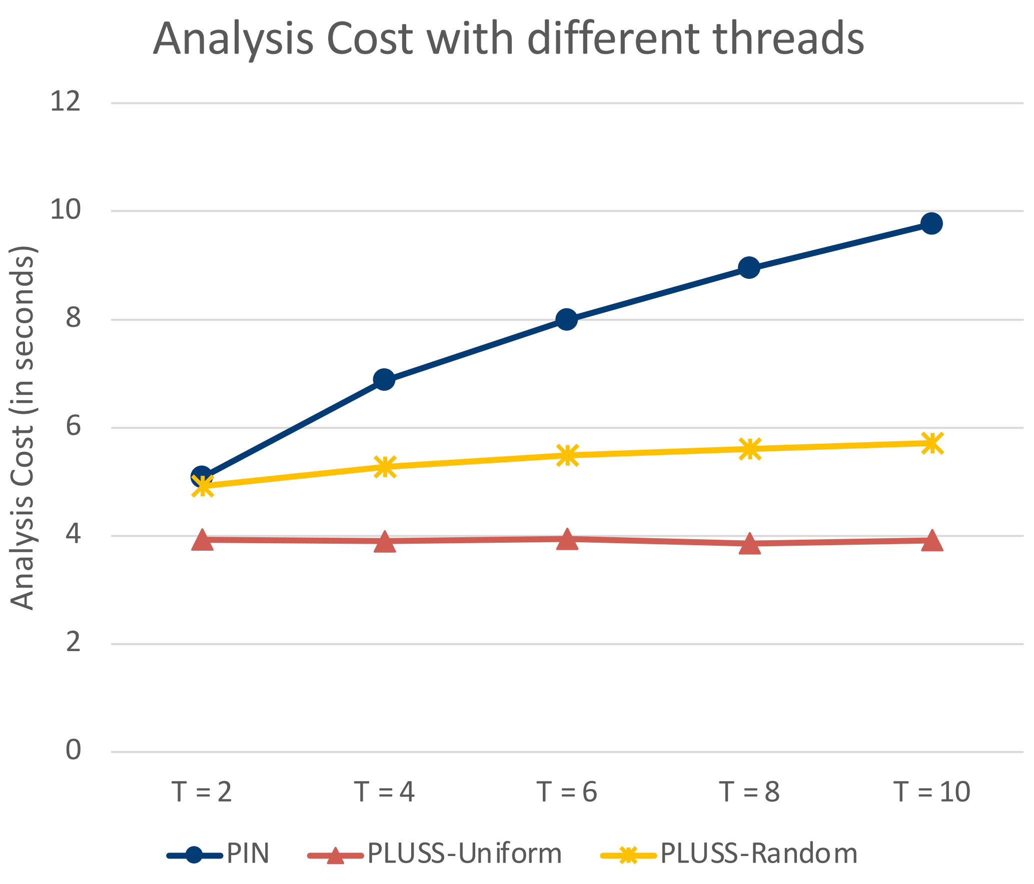


Evaluation

- We implement PLUSS on LLVM 11, and measures both the accuracy and the efficiency on 21/30 benchmarks from PolyBench.
- We use the binary instrumentation tool, PIN, to collect the baseline RI histogram.
- In terms of thread interleaving, we test two models: Uniform Interleaving and Random Interleaving.

Evaluation - Speed

- When T = 4, PLUSS achieves 1.3x, 1.7x speedup.
- With the thread counts increases, the costs of PLUSS scales the least.

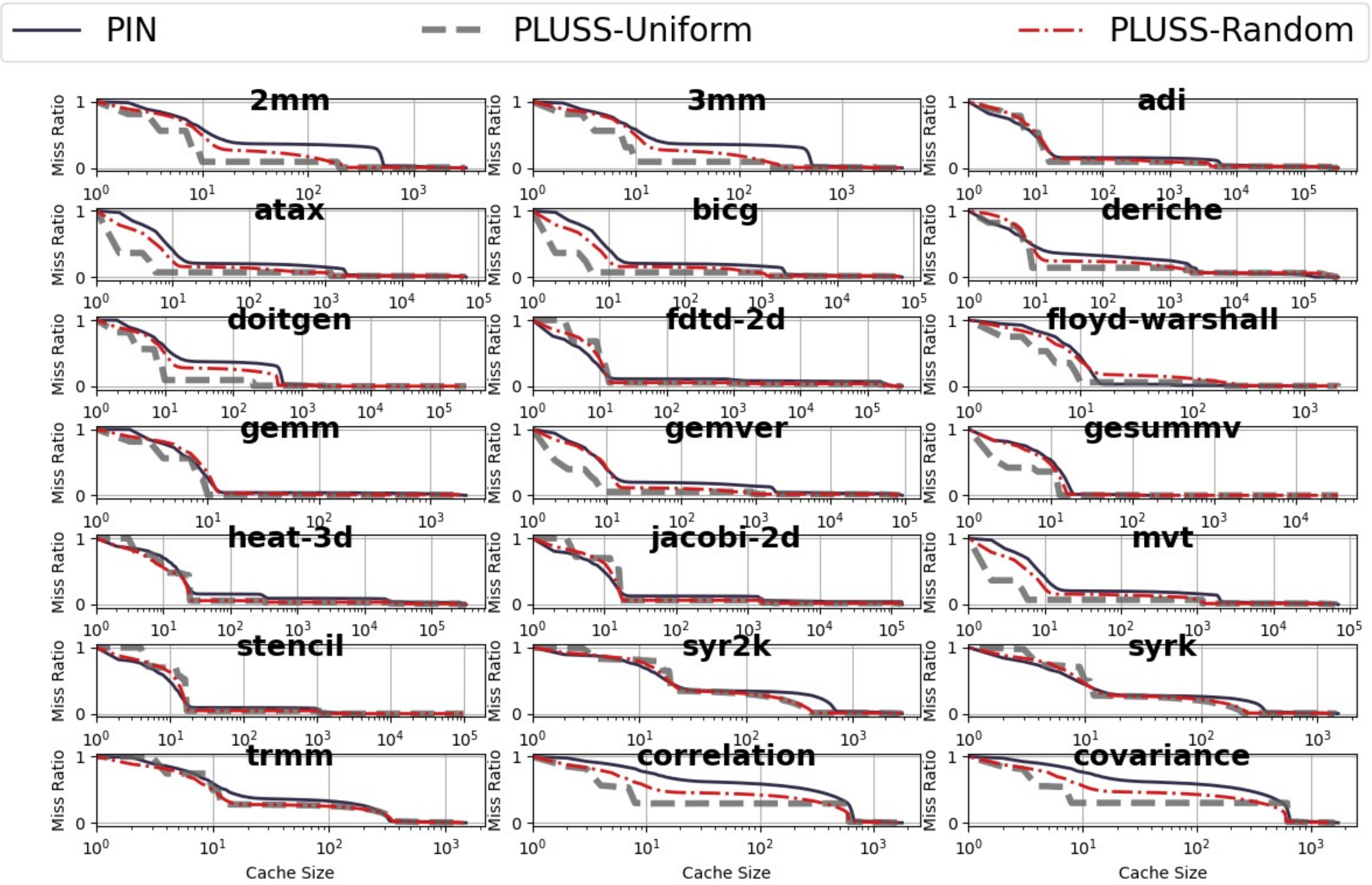


Evaluation - Accuracy

We separated the miss ratio curves into 3 regions and computes the L1-norm between two curves (PLUSS vs. PIN) as the accuracy.

PLUSS Techniques	**Similarity with PIN (Geomean)**			
	C < 640B	640B < C < 64KB	C > 64KB	Overall
Uniform	80.61%	91.23%	98.69%	96.18%
Random	92.93%	92.90%	98.70%	96.72%

Evaluation - Accuracy



arity with PIN (Geomean)		
640B < C < 64KB	C > 64KB	Overall
91.23%	98.69%	96.18%
92.90%	98.70%	96.72%

Limitations

- Support SCoP [Tobias et al. IMPACT’11] loop regions only.
  - Loop bounds and array subscripts are affined.
- Limited OMP directives support
  - static / dynamic scheduling clause, with an optional chunk size.
- Does not consider branch conditions.
  - All branches will be considered taken during the RI search phase.

Potential Application to CnC

- Block size tuning.
  - Specify the input size of each step
- Thread affinity tuning.
  - Specify the Thread-to-core mapping -> step_tuner::affinity
- Priority tuning.
  - Specify the “Locality-dependence “-> step_tuner::priority

Any Questions?