

A Machine Learning Based Parallelization Assistant

Aleksandr Maramzin, Christos Vasiladiotis, Roberto Castañeda Lozano,
Björn Franke, Murray Cole

The University of Edinburgh
United Kingdom



AI-SEPS 2019 workshop



SPLASH
ATHENS 2019

A Machine Learning Based Parallelization Assistant

Aleksandr Maramzin, Christos Vasiladiotis, Roberto Castañeda Lozano,
Björn Franke, Murray Cole

“It Looks Like You’re Writing a Parallel Loop”



AI-SEPS 2019 workshop



SPLASH
ATHENS 2019

Presentation Structure

- Problem statement
- How to use the assistant tool
- Assistant internals
- Parallelization of SNU NPB benchmarks

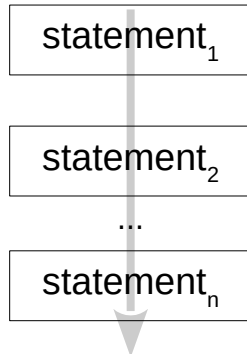


Problem Statement

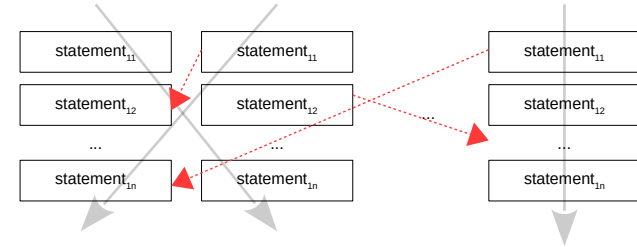
**Parallel Hardware
is Ubiquitous**



**Software
is Sequential**



**Automatic Parallelization
is Limited**



**Manual Parallelization
is Hard**



Problem Statement

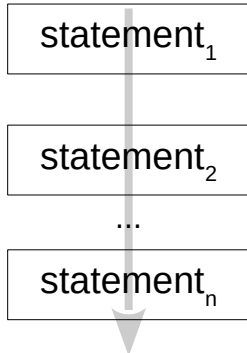
Parallel Hardware
is Ubiquitous

Automatic Parallelization
is Limited

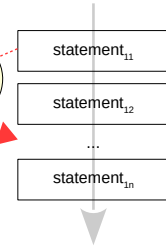
The **assistant solution** we propose alleviates
the process of manual parallelization



Software
is Sequential



Manual Parallelization
is Hard



Problem Statement

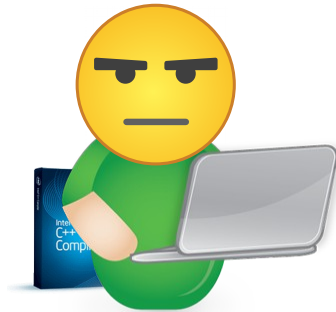
NAS Parallel
Benchmarks
(NPB)



**Parallelization Speedup
(Gmean)**

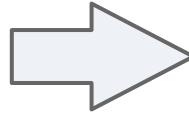
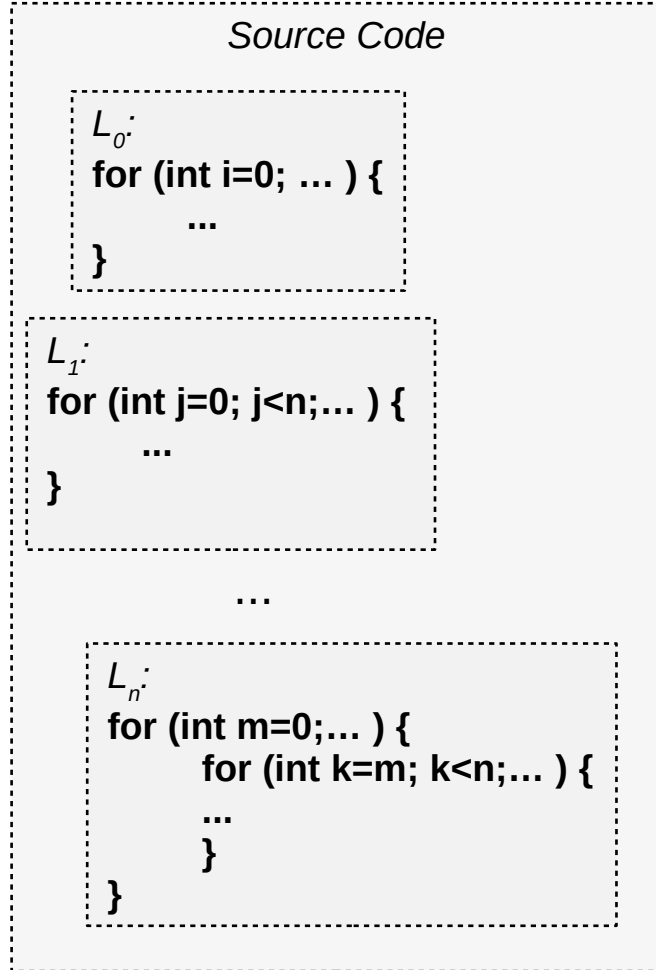
Manual: **1.73x**

Automatic: **0.79x - 1.10x**

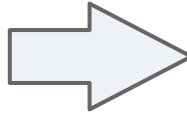
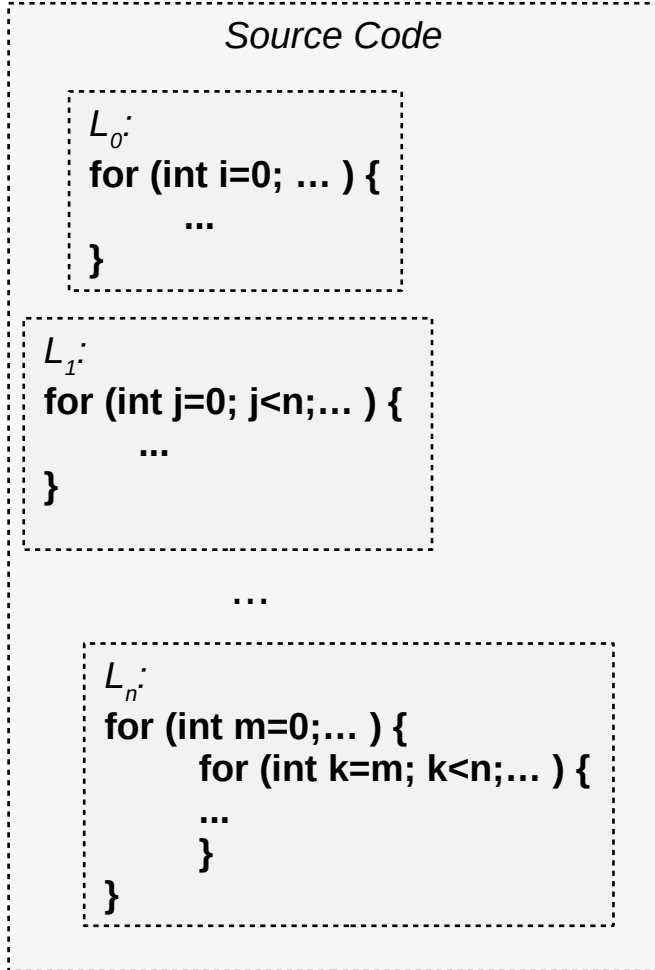


**Programmer still plays the major role
in the hard task of software
parallelization**

Manual Parallelization: Where Should I Start?

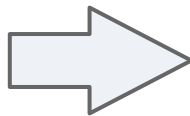
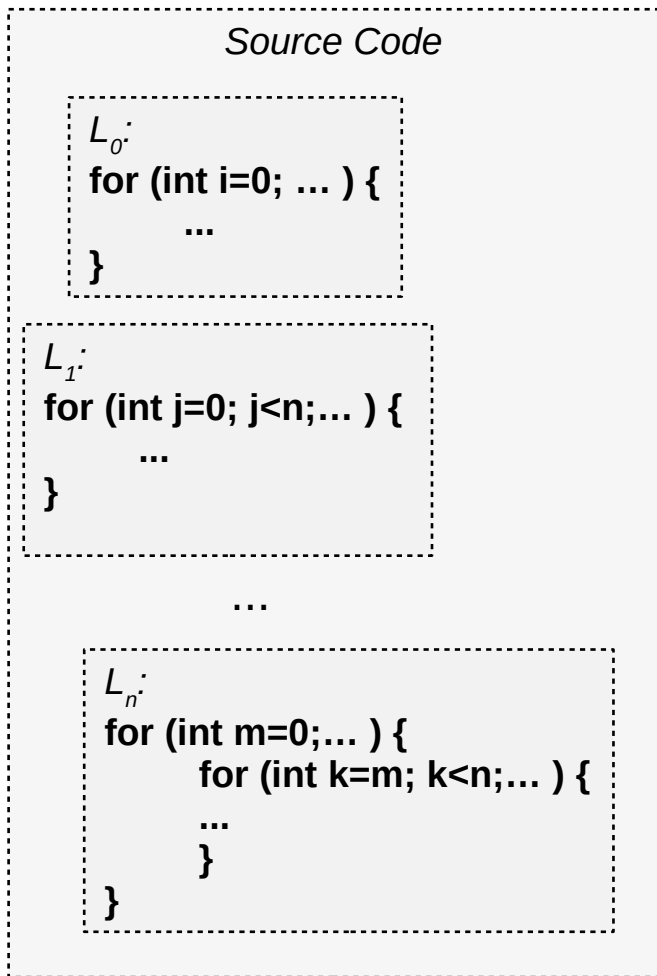


Manual Parallelization: Where Should I Start?



Where Should I Start?

Manual Parallelization: Where Should I Start?

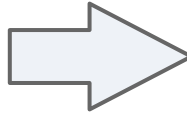
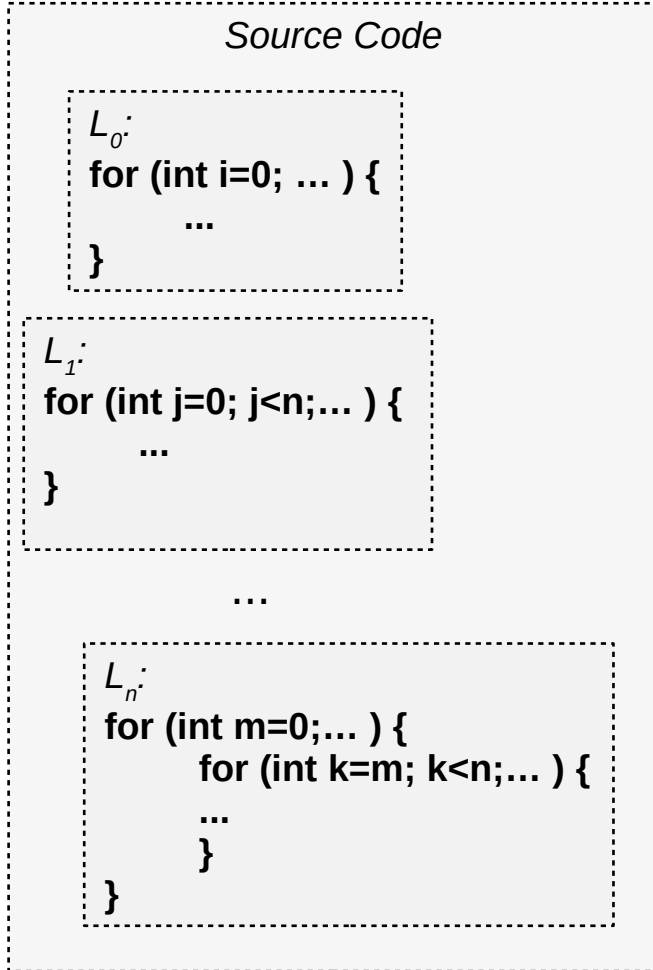


Program Loop Rankings

Source Order

L_0, L_1, \dots, L_n

Manual Parallelization: Where Should I Start?

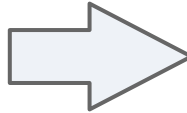
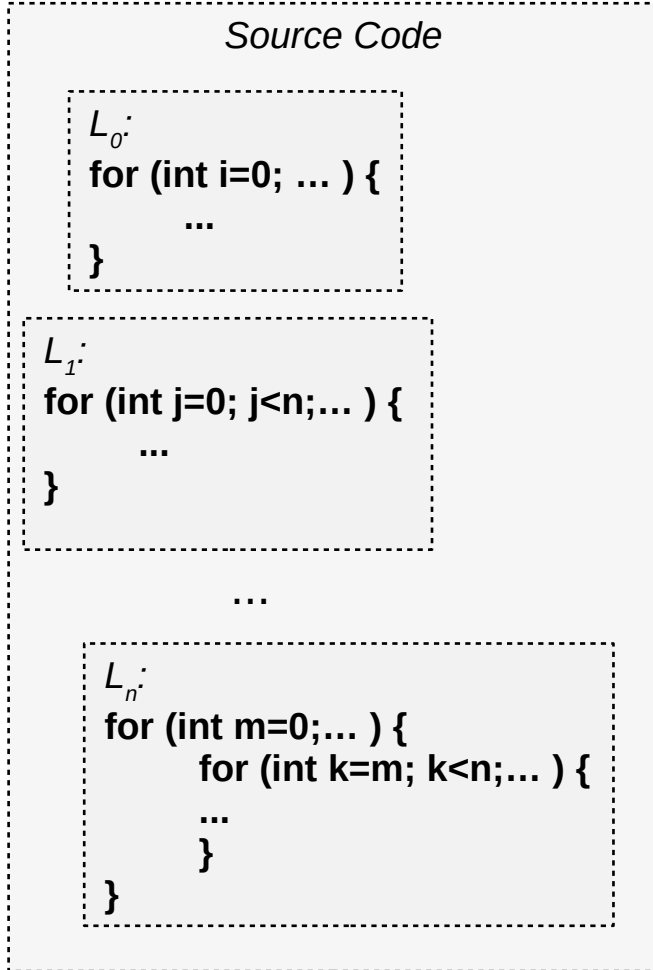


Program Loop Rankings



Source Order
 L_0, L_1, \dots, L_n

Manual Parallelization: Where Should I Start?



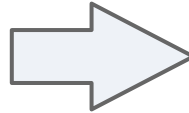
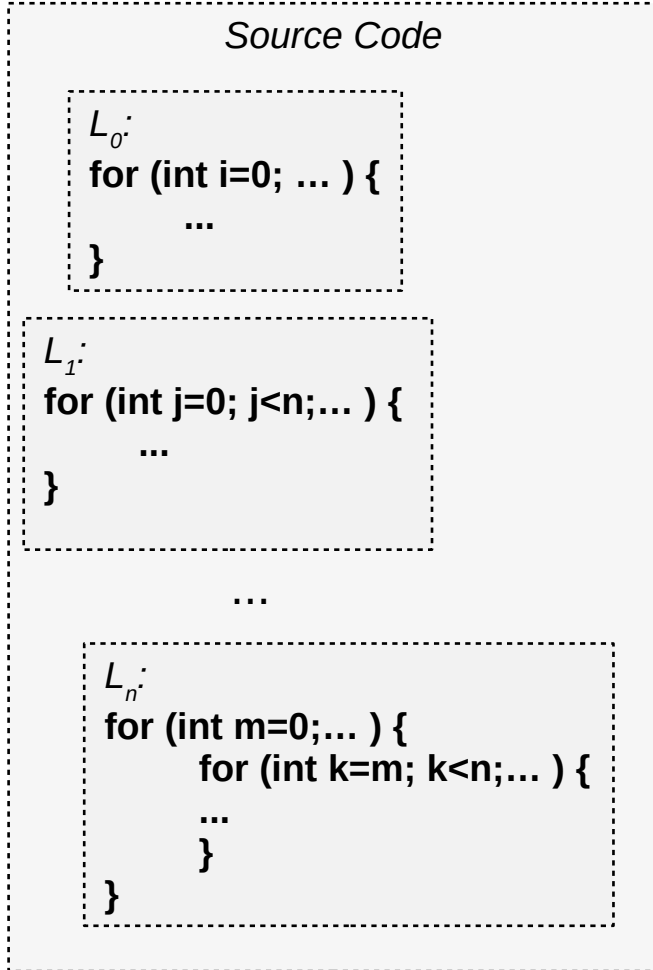
Program Loop Rankings



Source Order
 L_0, L_1, \dots, L_n



Manual Parallelization: Where Should I Start?



Program Loop Rankings



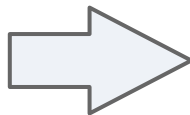
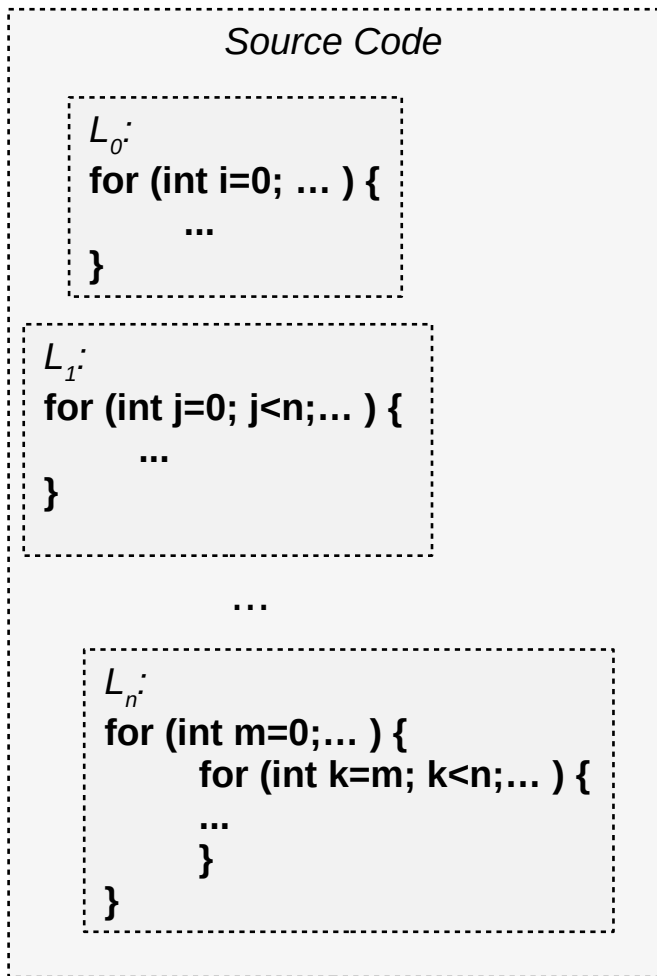
Source Order
 L_0, L_1, \dots, L_n



Traditional Empirical and
Experimental Method

Profile Order
 $L_{17}, L_{12}, \dots, L_1$

Manual Parallelization: Where Should I Start?



Program Loop Rankings



Source Order

L_0, L_1, \dots, L_n



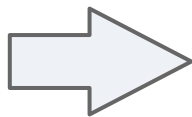
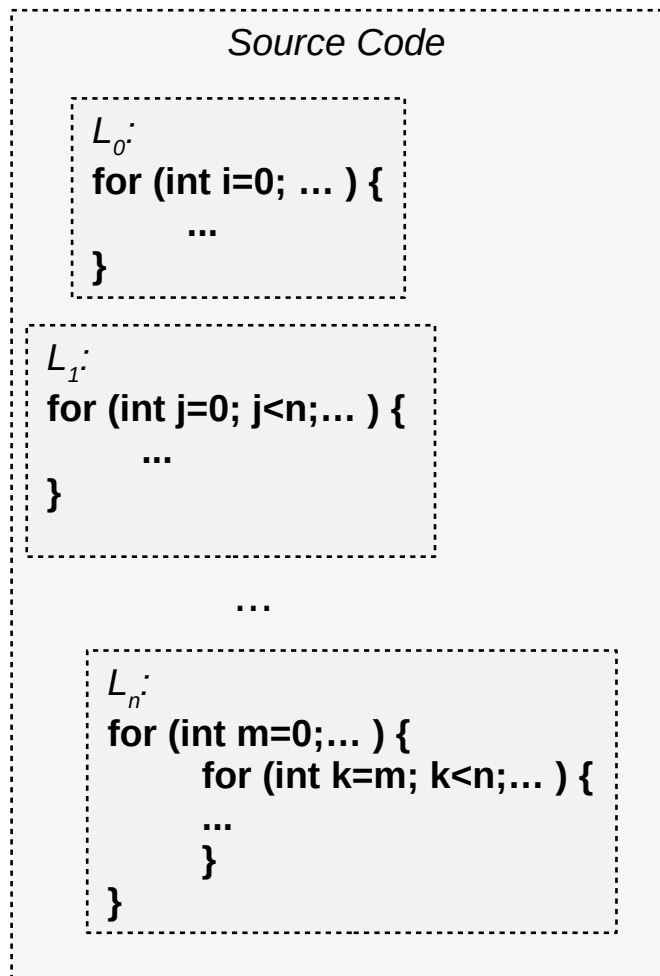
Traditional Empirical and
Experimental Method

Profile Order

$L_{17}, L_{12}, \dots, L_1$



Manual Parallelization: Where Should I Start?



Traditional Empirical and Experimental Method

Program Loop Rankings



Source Order
 L_0, L_1, \dots, L_n



Profile Order
 $L_{17}, L_{12}, \dots, L_1$

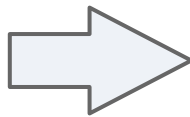
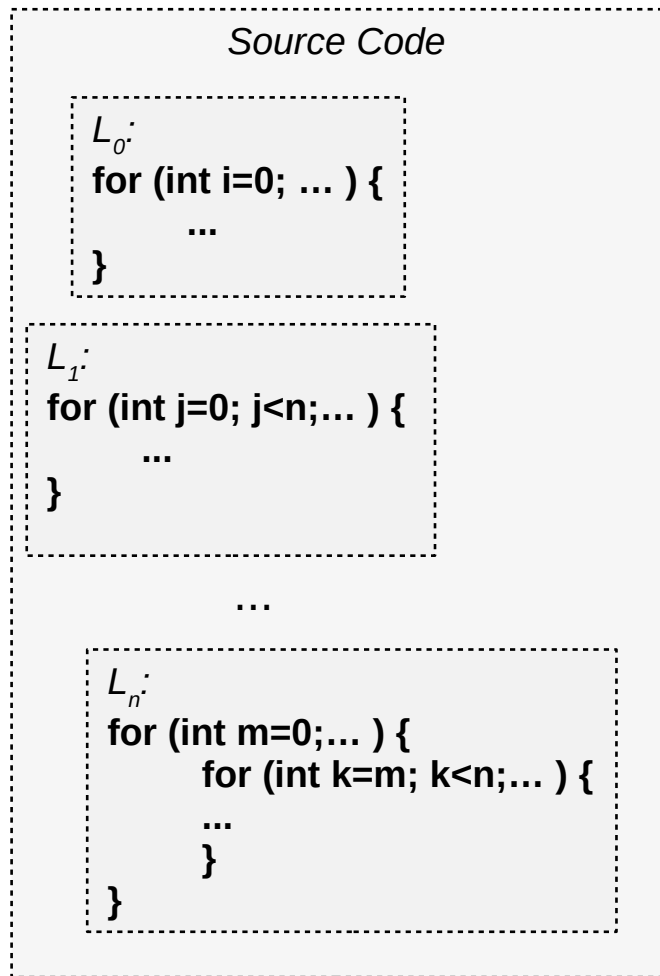


AI-inspired Method

Profitability* Order
 L_{12}, L_7, \dots, L_3

$$* \text{Profitability} = F\left(\frac{\text{Performance Gain}}{\text{Programmer Effort}}\right)$$

Manual Parallelization: Where Should I Start?



Traditional Empirical and
Experimental Method

Program Loop Rankings



Source Order
 L_0, L_1, \dots, L_n



Profile Order
 $L_{17}, L_{12}, \dots, L_1$



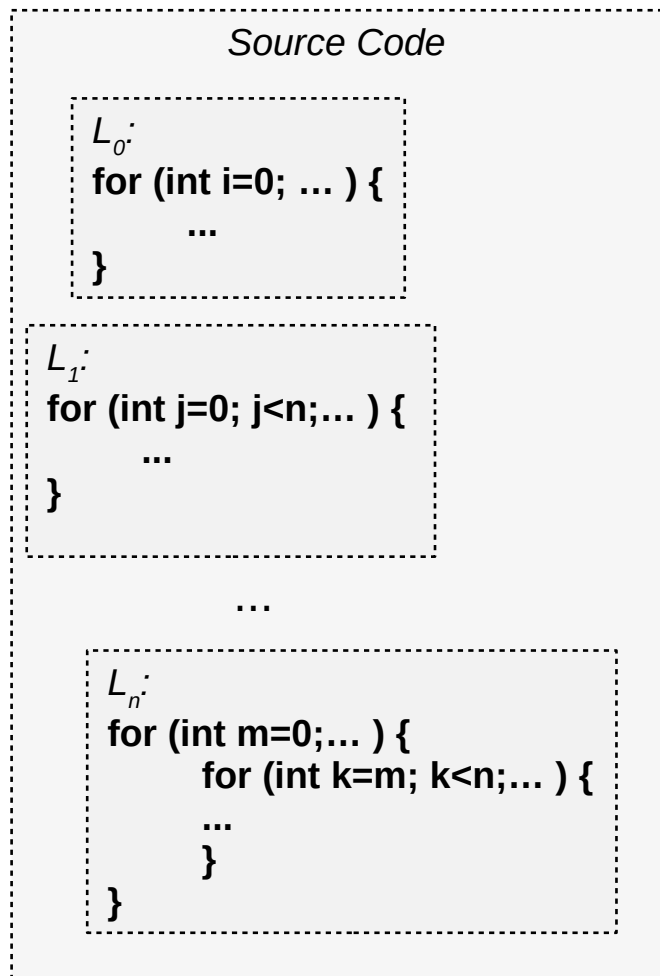
Profitability* Order
 L_{12}, L_7, \dots, L_3



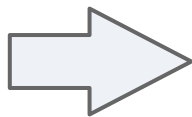
AI-inspired Method

$$* \text{Profitability} = F\left(\frac{\text{Performance Gain}}{\text{Programmer Effort}}\right)$$

Manual Parallelization: Where Should I Start?



Final Programmer Decision



Traditional Empirical and Experimental Method



Program Loop Rankings



Source Order
 L_0, L_1, \dots, L_n



Profile Order
 $L_{17}, L_{12}, \dots, L_1$



Profitability* Order
 L_{12}, L_7, \dots, L_3



AI-inspired Method

$$* \text{Profitability} = F\left(\frac{\text{Performance Gain}}{\text{Programmer Effort}}\right)$$

Proposed Solution

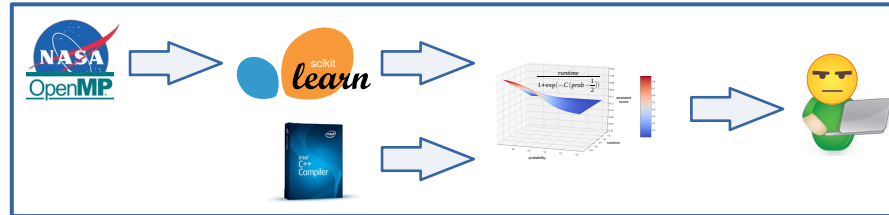
(ML based software parallelization assistant)

Tool



&

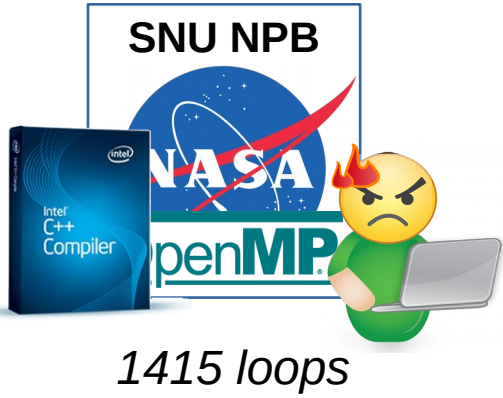
Methodology



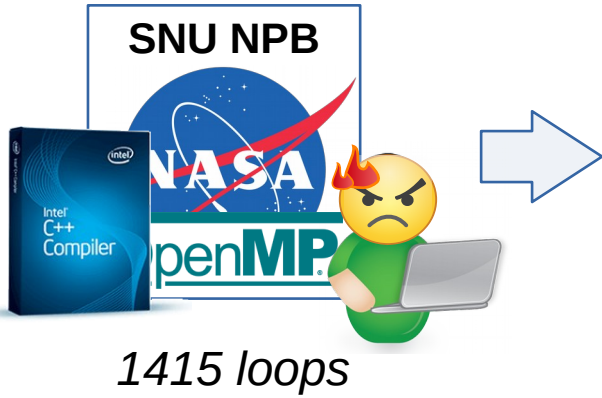
AI-SEPS 2019 workshop

Assistant Training

Assistant Training



Assistant Training



Training Data Set

Parallelizable loops

```
for (int i=...) { ... } [ f1, f2, ..., fn ]  
...  
for (int j=...) { ... } [ f1, f2, ..., fn ]
```

995 loops *

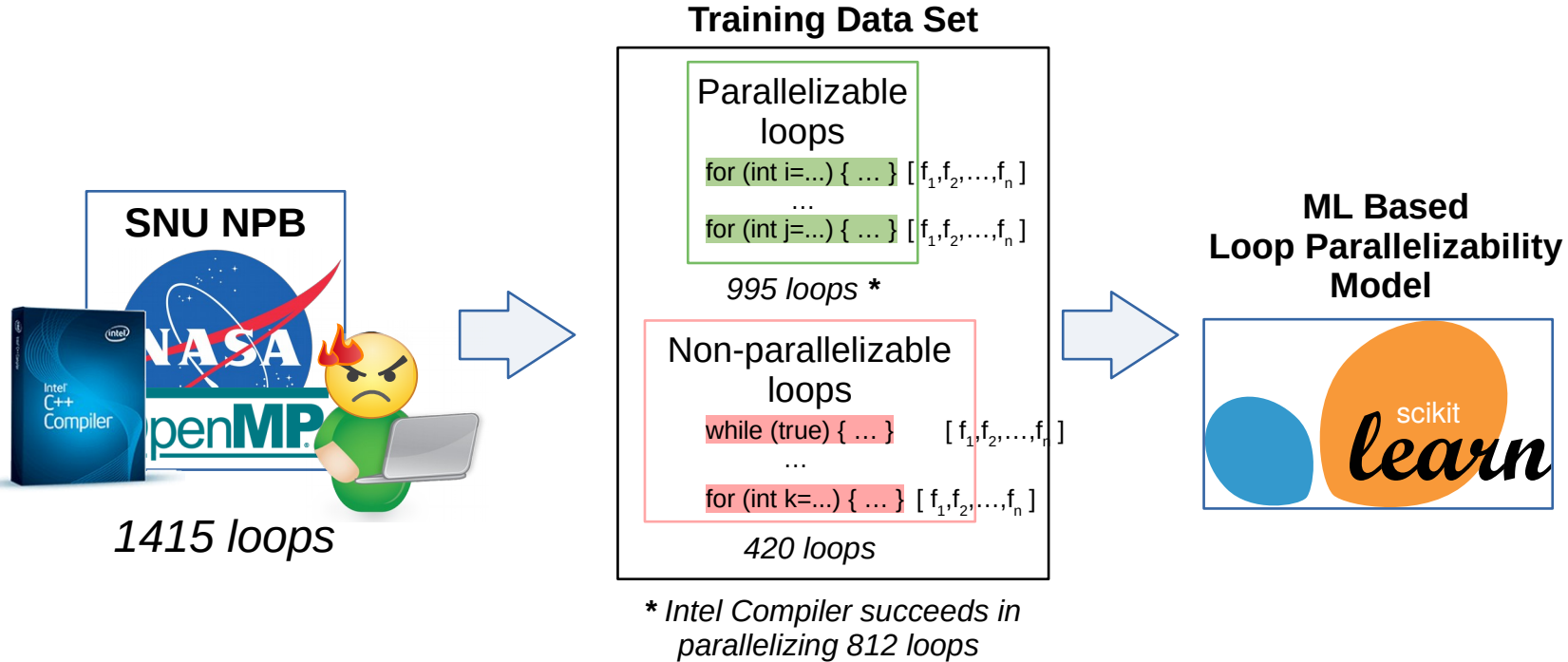
Non-parallelizable loops

```
while (true) { ... } [ f1, f2, ..., fn ]  
...  
for (int k=...) { ... } [ f1, f2, ..., fn ]
```

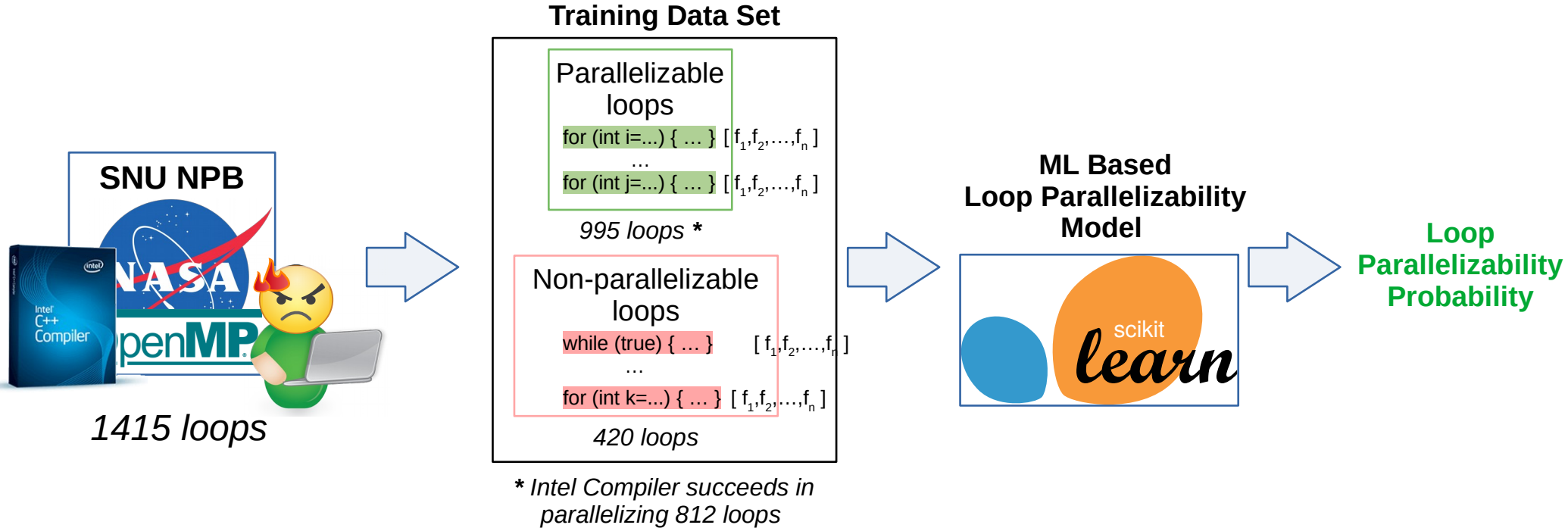
420 loops

* Intel Compiler succeeds in parallelizing 812 loops

Assistant Training

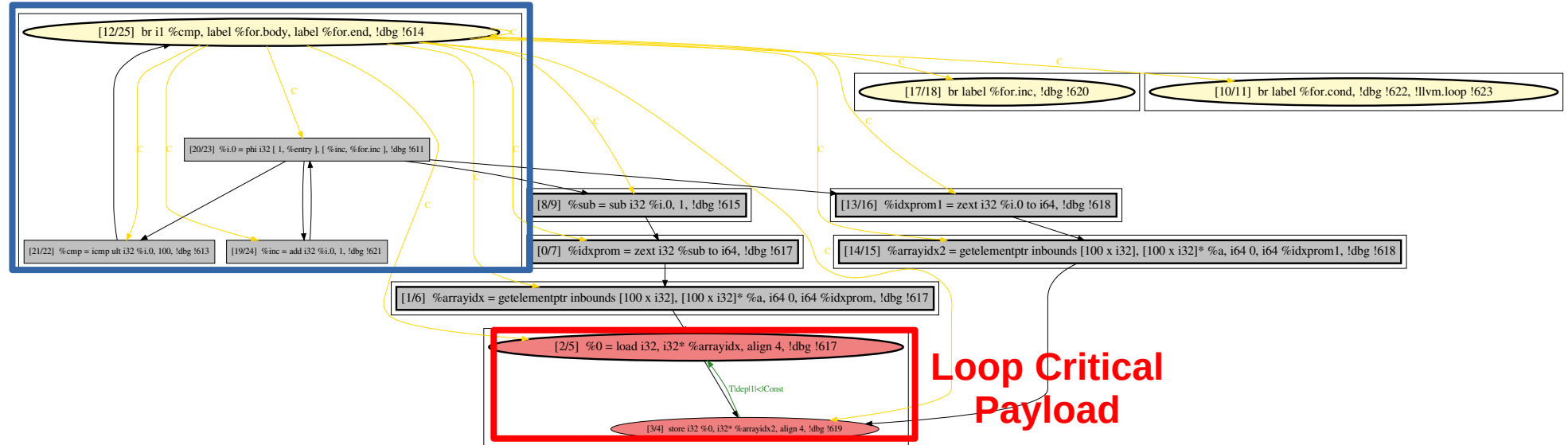


Assistant Training



Loop Features

Loop Iterator



Loop features are based on static structural properties of loop program dependence graphs (PDGs):

- Absolute size
- Loop Iterator/Payload cohesion
- Number of dependence edges
- Instruction types (calls, loads/stores, etc.)
- etc.

Loop Ranking Score

ML Based
Loop Parallelizability
Model



Loop Ranking Score

ML Based
Loop Parallelizability
Model



Not Aware of Loop
Running Time

Loop Ranking Score

Profiler



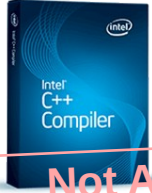
**ML Based
Loop Parallelizability
Model**



**Not Aware of Loop
Running Time**

Loop Ranking Score

Profiler



Not Aware of Loop
Parallelizability

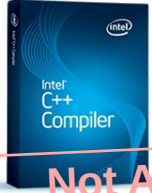
ML Based Loop Parallelizability Model



Not Aware of Loop
Running Time

Loop Ranking Score

Profiler



Not Aware of Loop
Parallelizability

**ML Based
Loop Parallelizability
Model**



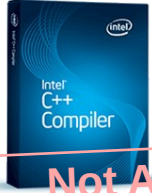
Not Aware of Loop
Running Time

Programmer Effort



Loop Ranking Score

Profiler



Not Aware of Loop
Parallelizability

Performance Gain



ML Based
Loop Parallelizability
Model



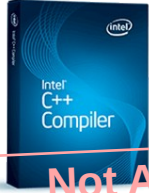
Not Aware of Loop
Running Time

Programmer Effort



Loop Ranking Score

Profiler

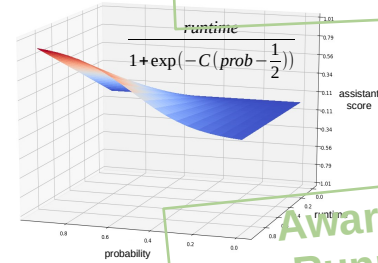


Not Aware of Loop Parallelizability

Performance Gain



Aware of Loop Parallelizability *



Aware of Loop Running Time

ML Based
Loop Parallelizability
Model



Not Aware of Loop Running Time

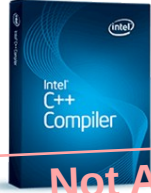
Programmer Effort



$$* \text{Profitability} = F\left(\frac{\text{Performance Gain}}{\text{Programmer Effort}}\right)$$

Loop Ranking Score

Profiler



Not Aware of Loop
Parallelizability

Performance Gain



ML Based
Loop Parallelizability
Model



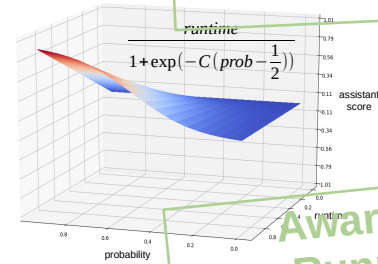
Not Aware of Loop
Running Time

Programmer Effort



Aware of Loop
Parallelizability

*



Aware of Loop
Running Time

Improved Loop
Ranking

$$* \text{Profitability} = F\left(\frac{\text{Performance Gain}}{\text{Programmer Effort}}\right)$$

Results



Deployment of our assistant on
SNU NPB benchmarks



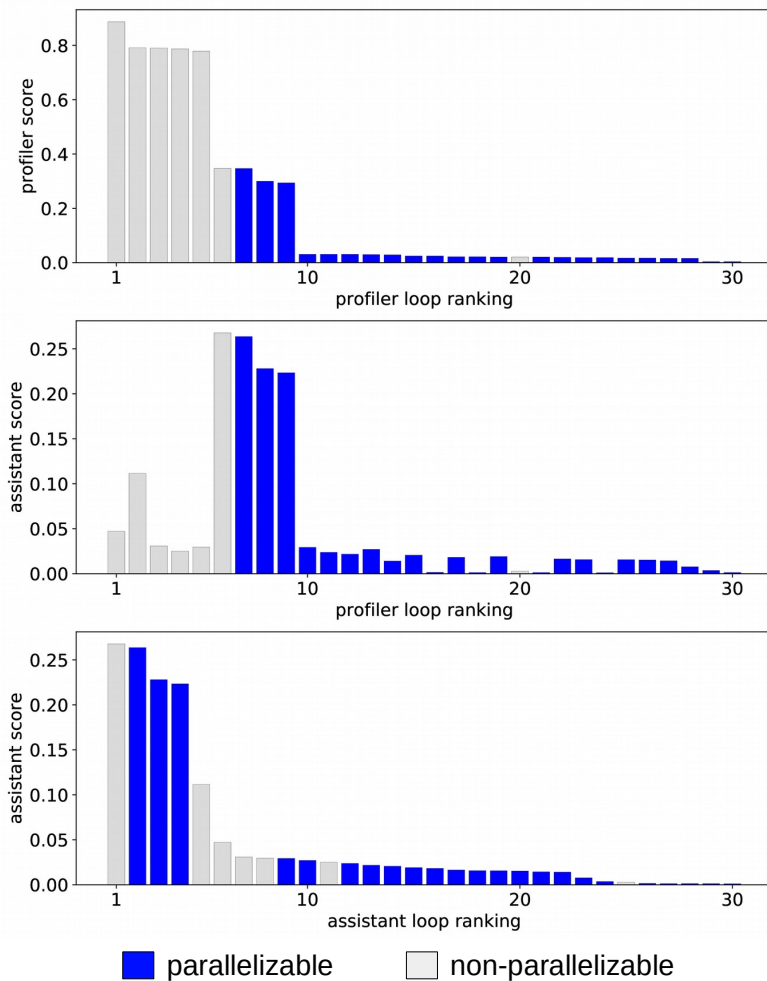
Predictive performance of our ML
based loop parallelizability model



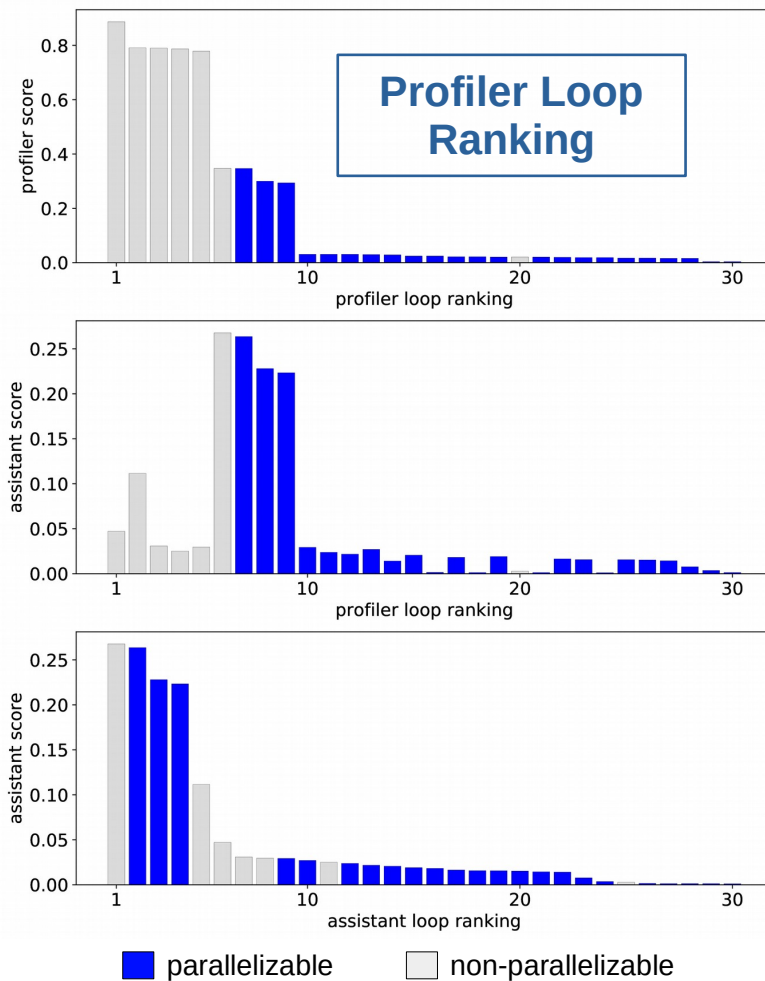
AI-SEPS 2019 workshop



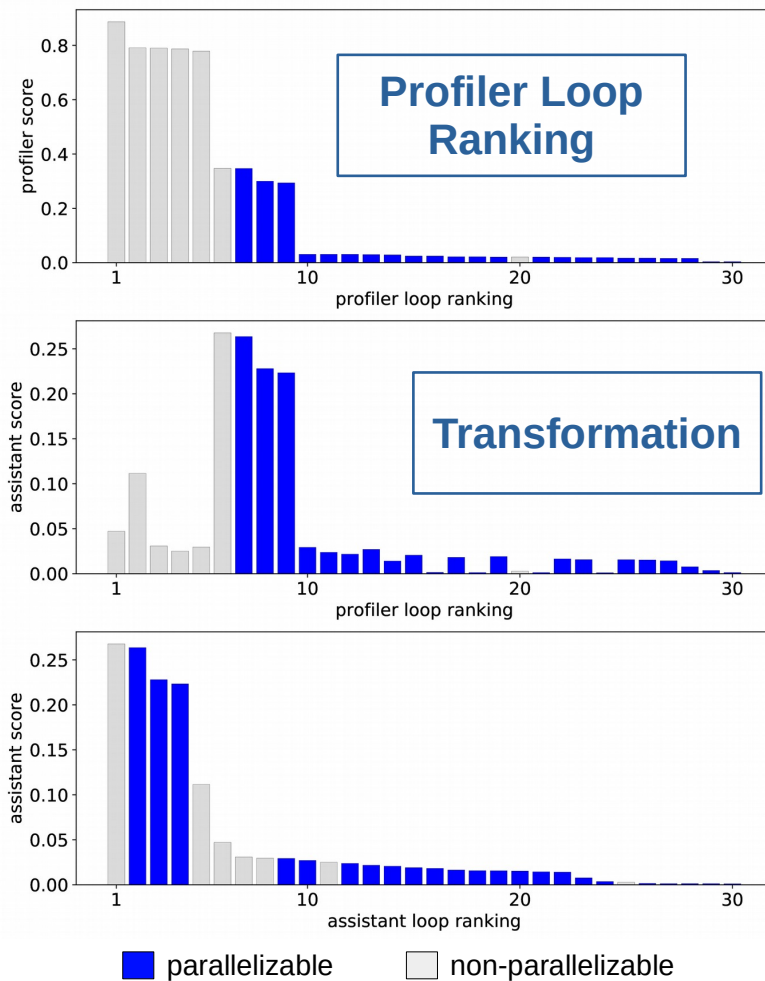
Loop Rankings: Profiler vs. Assistant



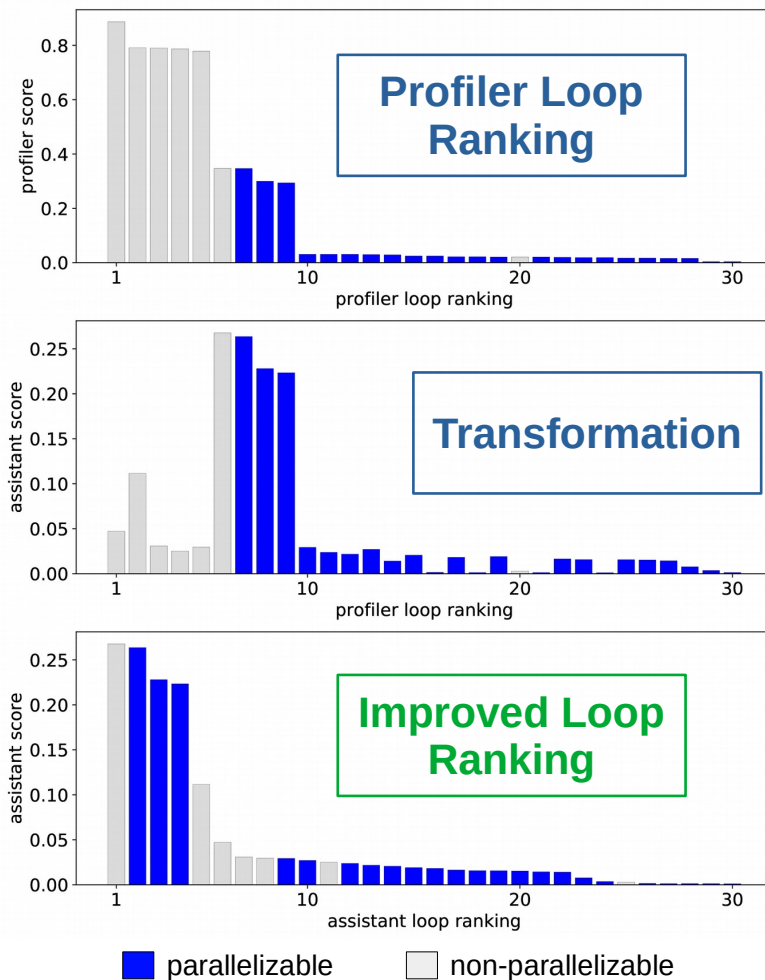
Loop Rankings: Profiler vs. Assistant



Loop Rankings: Profiler vs. Assistant

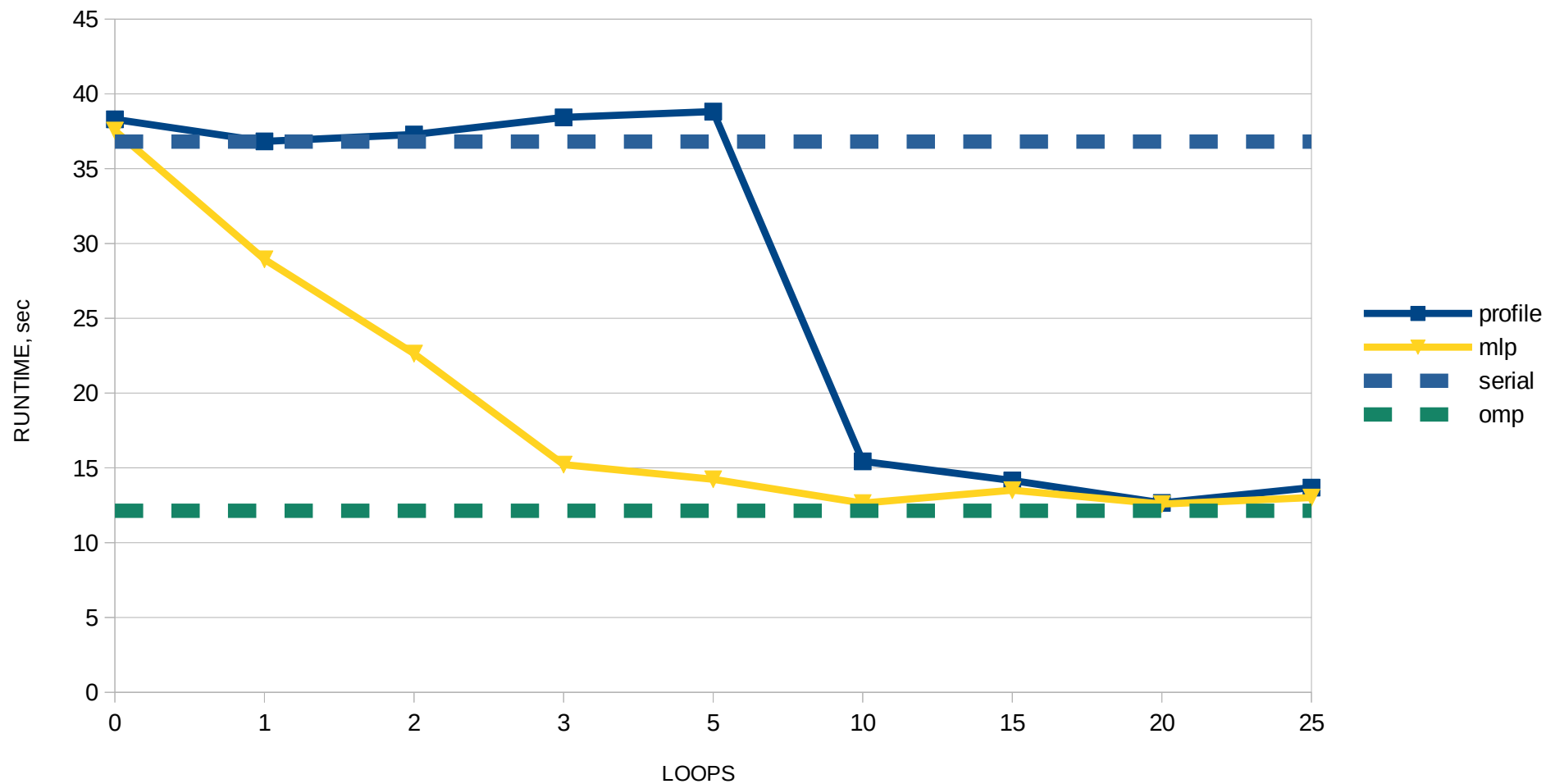


Loop Rankings: Profiler vs. Assistant

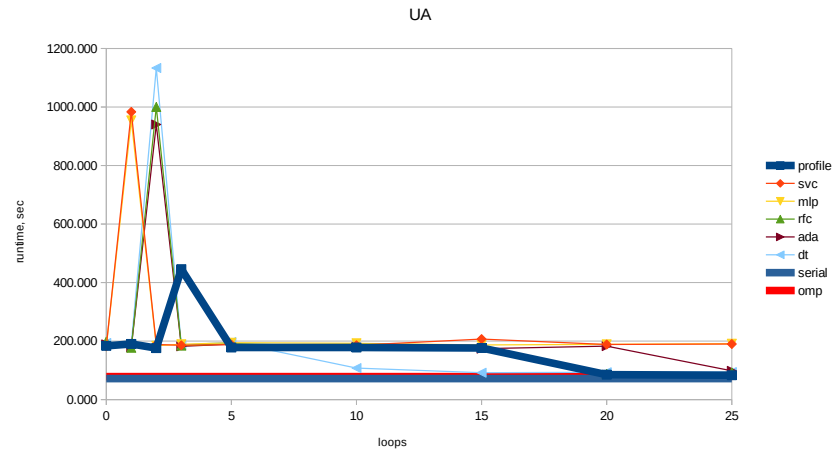
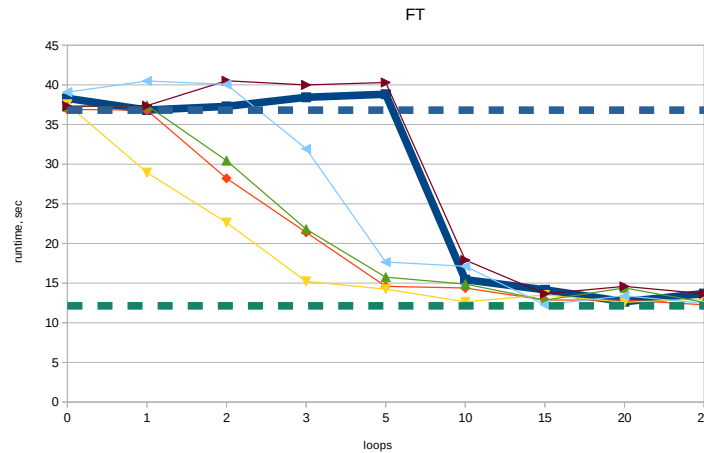
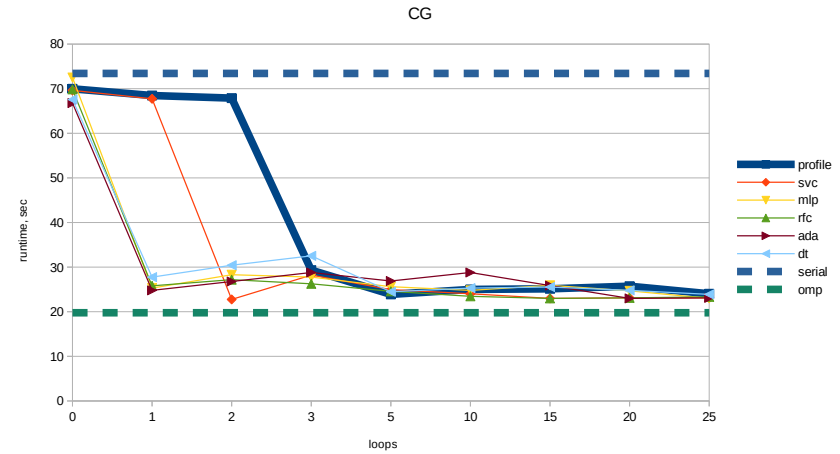
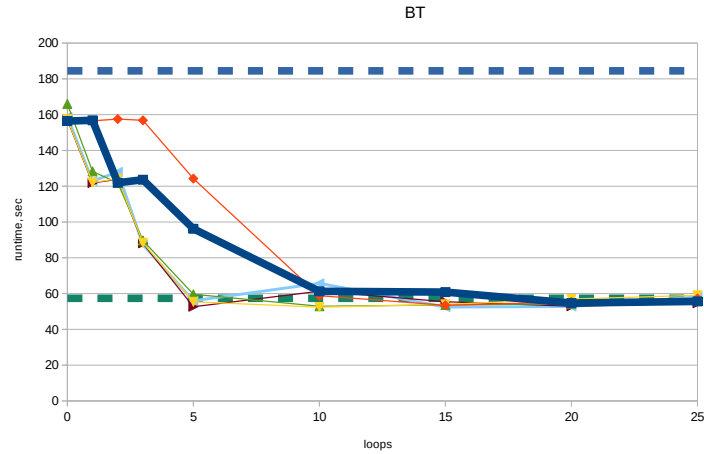


How well do we do?

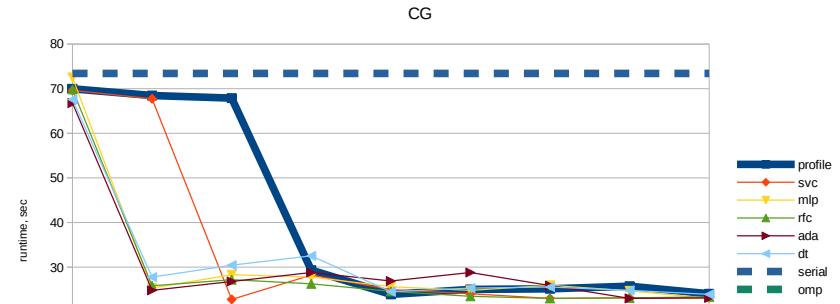
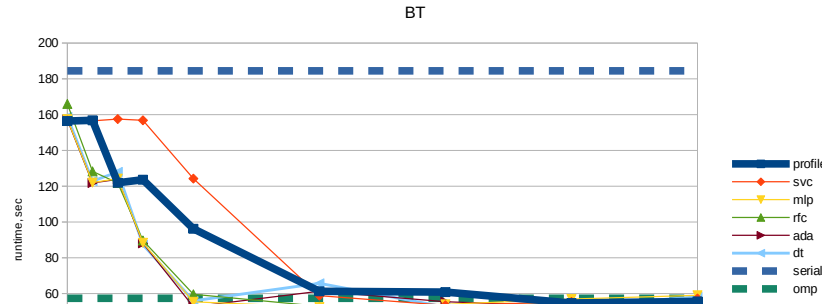
FT



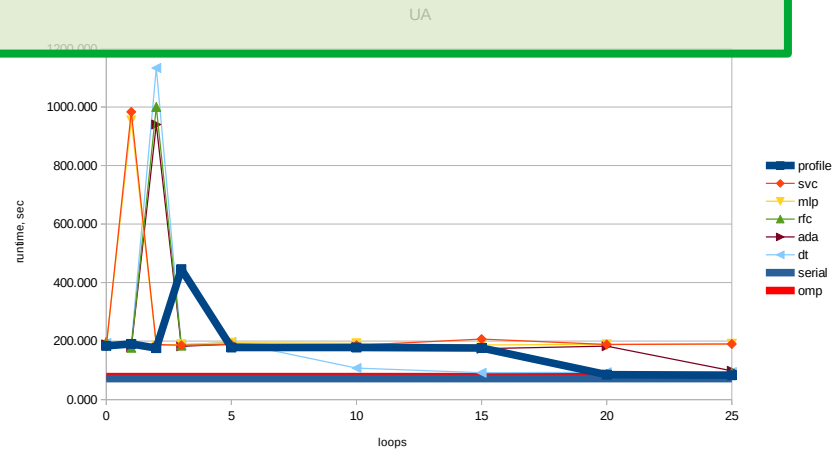
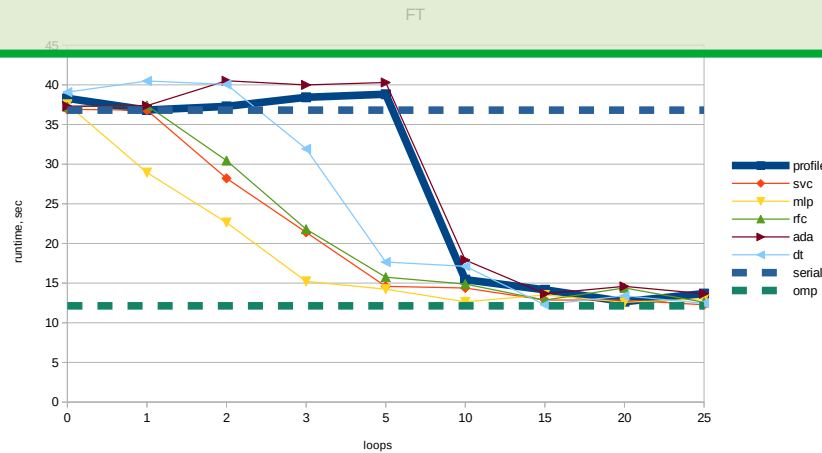
How well do we do?



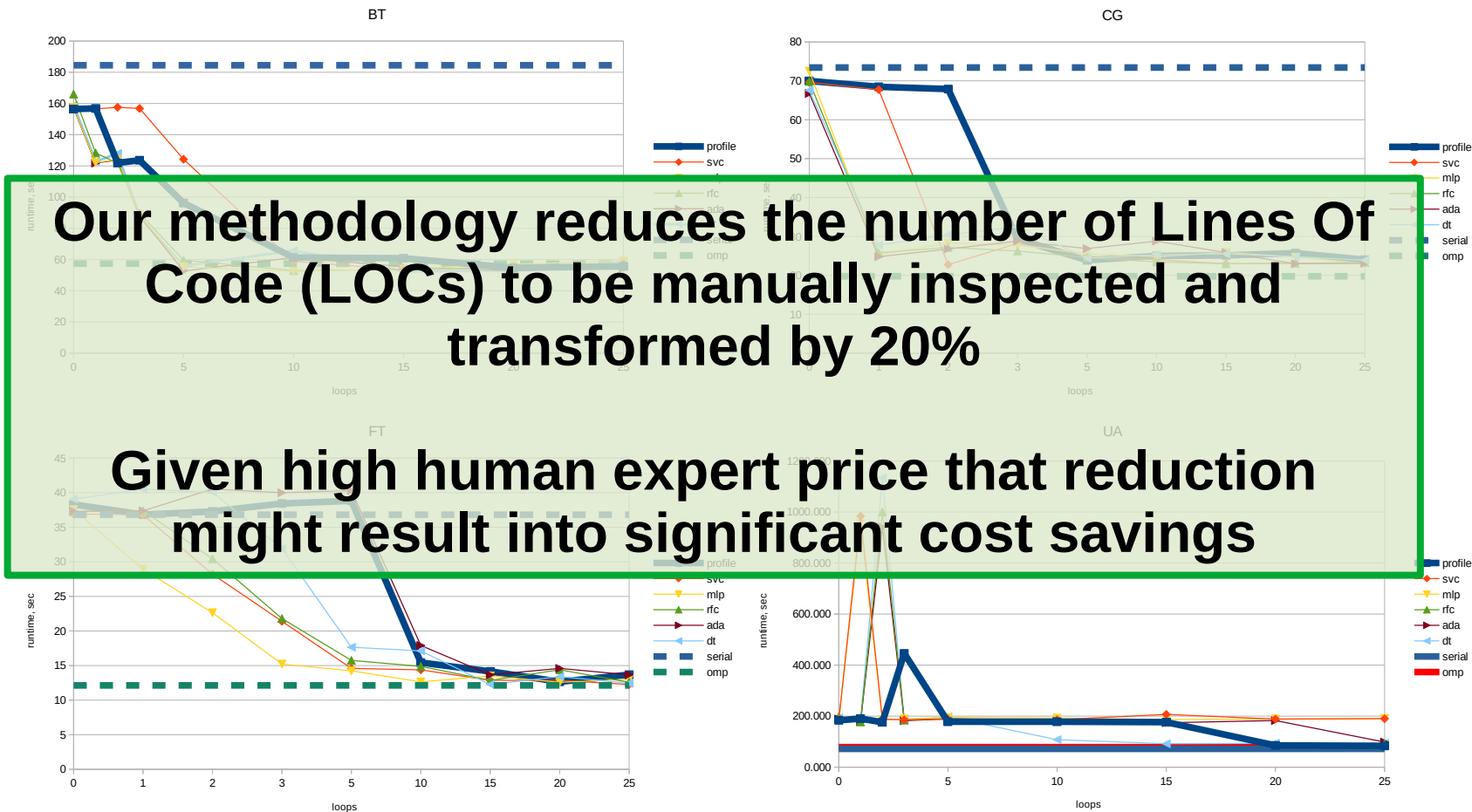
How well do we do?



8 benchmarks in total: 4 wins / 4 draws



How well do we do?



ML Model Predictive Performance

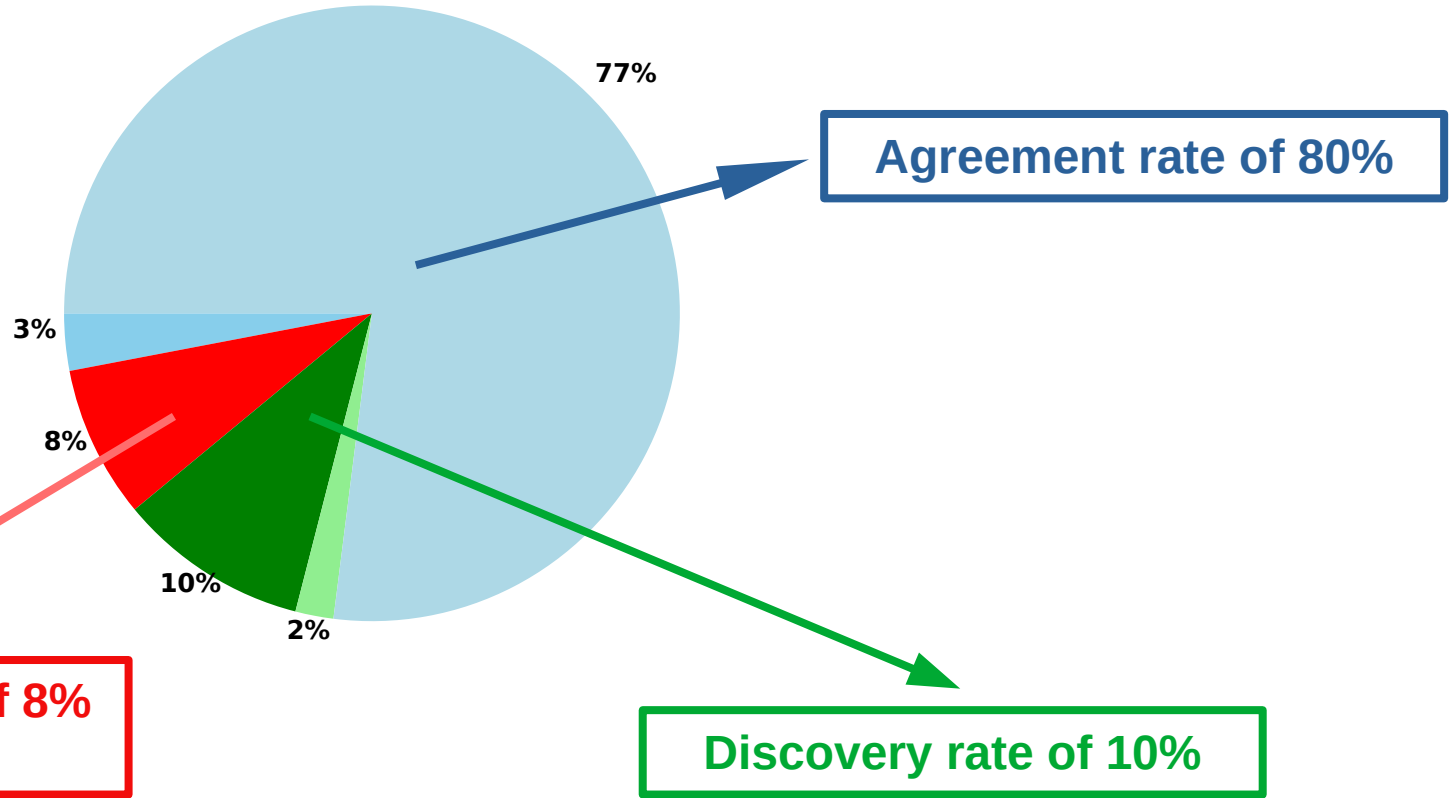
Reference

ML model	accuracy	recall	precision
constant	70.32	100	70.32
uniform	46.27	41.50	69.79
SVC	90.04	95.24	91.06
AdaBoost	86.96	92.92	89.06
DT	84.36	89.57	87.90
RFC	86.65	93.22	88.47
MLP	89.40	93.77	91.39

Around 90% predictive accuracy

ML Model Predictive Performance

ML Based
Loop Parallelizability
Model

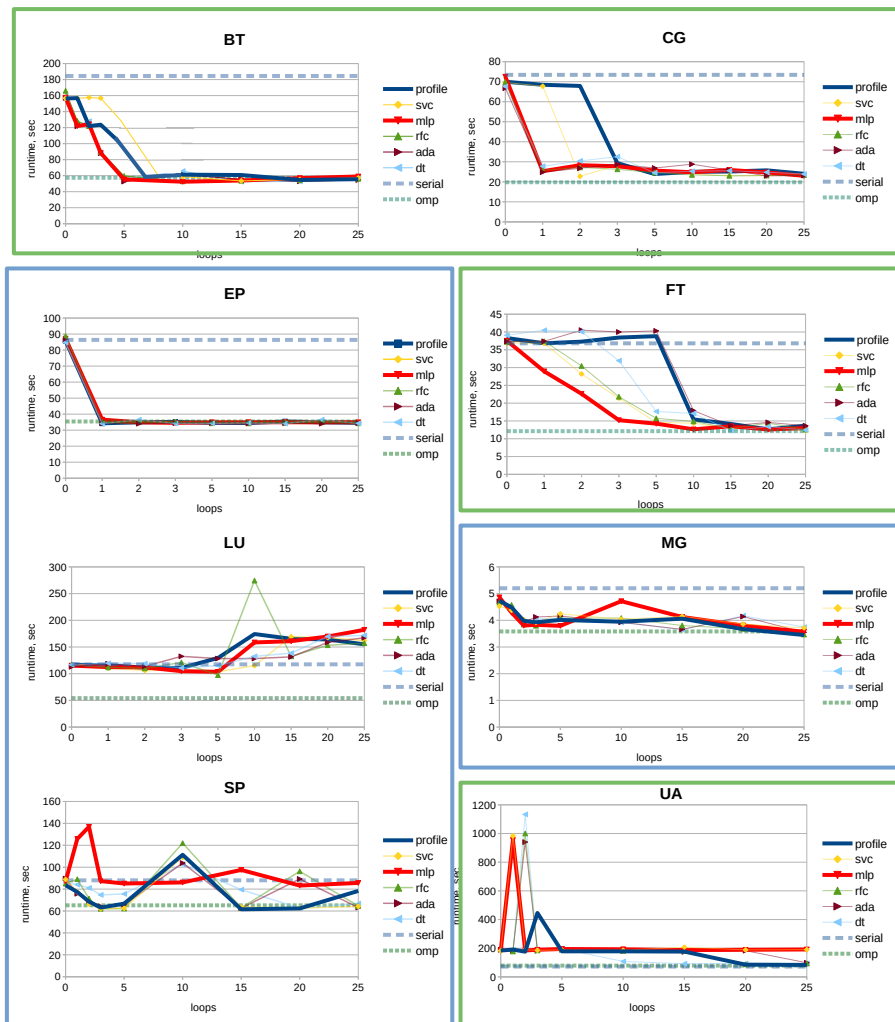


Summary & Conclusions

- Human experts still play the major role in the task of software parallelization
- Loop parallelizability is learnable property (we introduce a ML model and train it to work with the accuracy of above 90%)
- ML model of loop parallelizability has been integrated into parallelization assistant
- Deployed against SNU NPB benchmarks our assistant showed a faster parallelization process: 20% Lines Of Code (LOC) reduction



How well do we do?



Motivating example



SNU NPB Conjugate Gradient (CG) benchmark

```
for (j = 0; j < lastrow-firstrow+1; j++) {  
    sum1 = 0.0;  
    for (k = rowstr[j]; k < rowstr[j+1]; k++)  
        sum1 = sum1 + a[k]*p[colidx[k]];  
    q[j] = sum1;  
}
```

cg.c:509

```
for (it = 1; it <= NITER; it++) {  
    ...  
    if (timeron) timer_start(T_conj_grad);  
    conj_grad(colidx,rowstr,x,z,a,p,q,r,&rn timeron) timer_stop(T_conj_grad);  
    ...  
    printf("      %5d      %20.14E%20.13f\n", it,  
        rn timeron, zeta);  
    ...  
}
```

cg.c:326

Profiler's Loop Ranking

- 1 **cg.c:326**
- 2 **cg.c:484**
- 3 **cg.c:509**

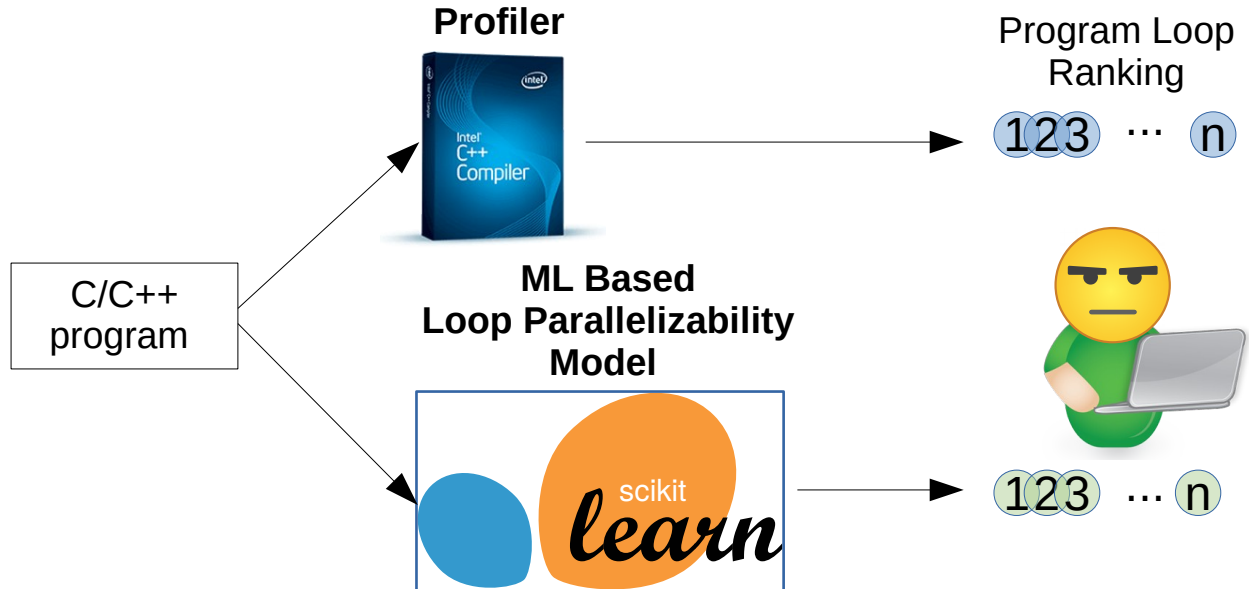
VS.

Assistant's Loop Ranking

- 1 **cg.c:509**
- 2 **cg.c:326**
- 3 **cg.c:484**

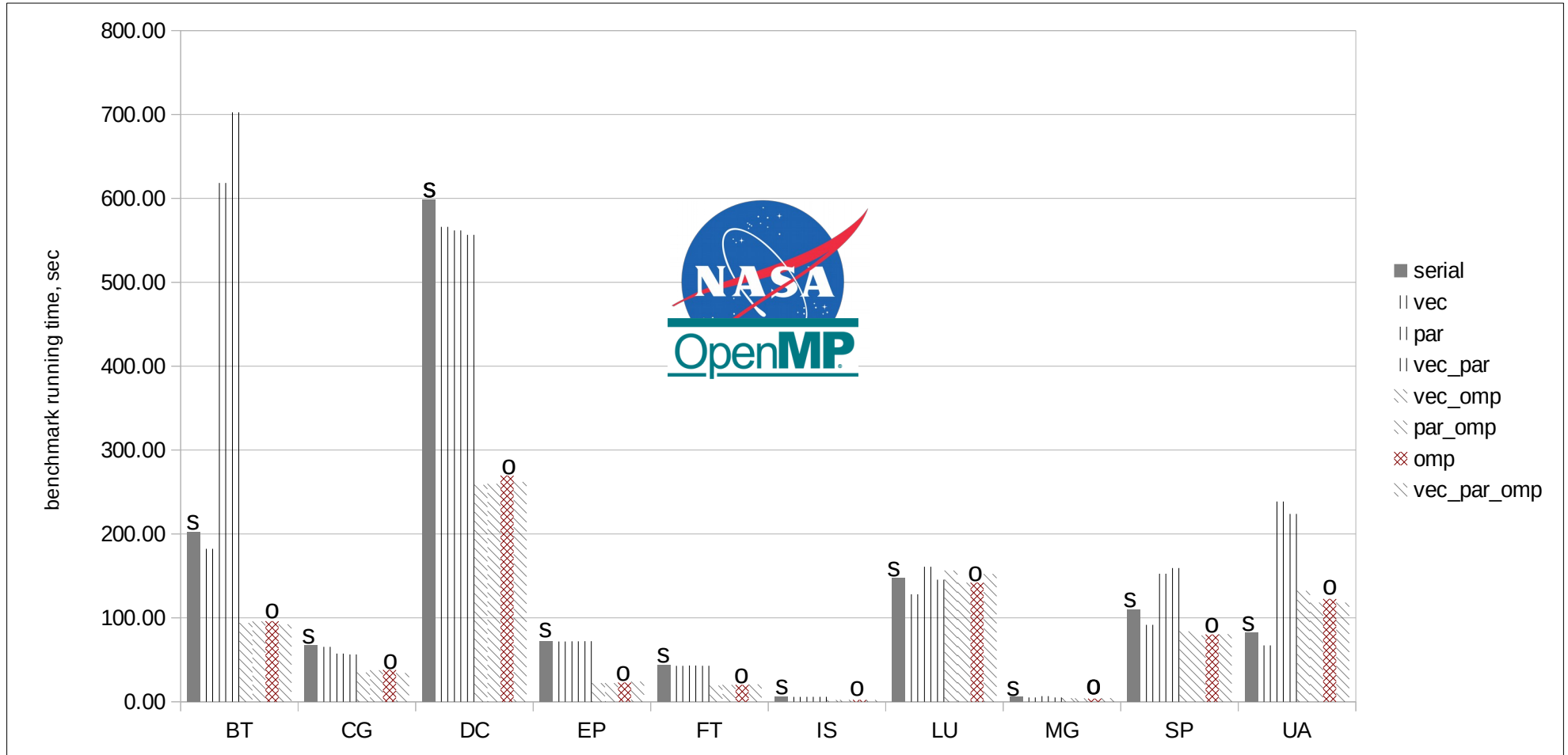
Assistant's Parallel Loop Probability

- [85%]
[29%]
[8%]

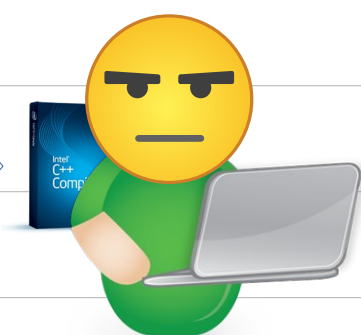
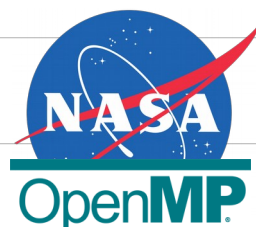


Problem Statement 2/2

NAS Parallel Benchmarks (NPB)



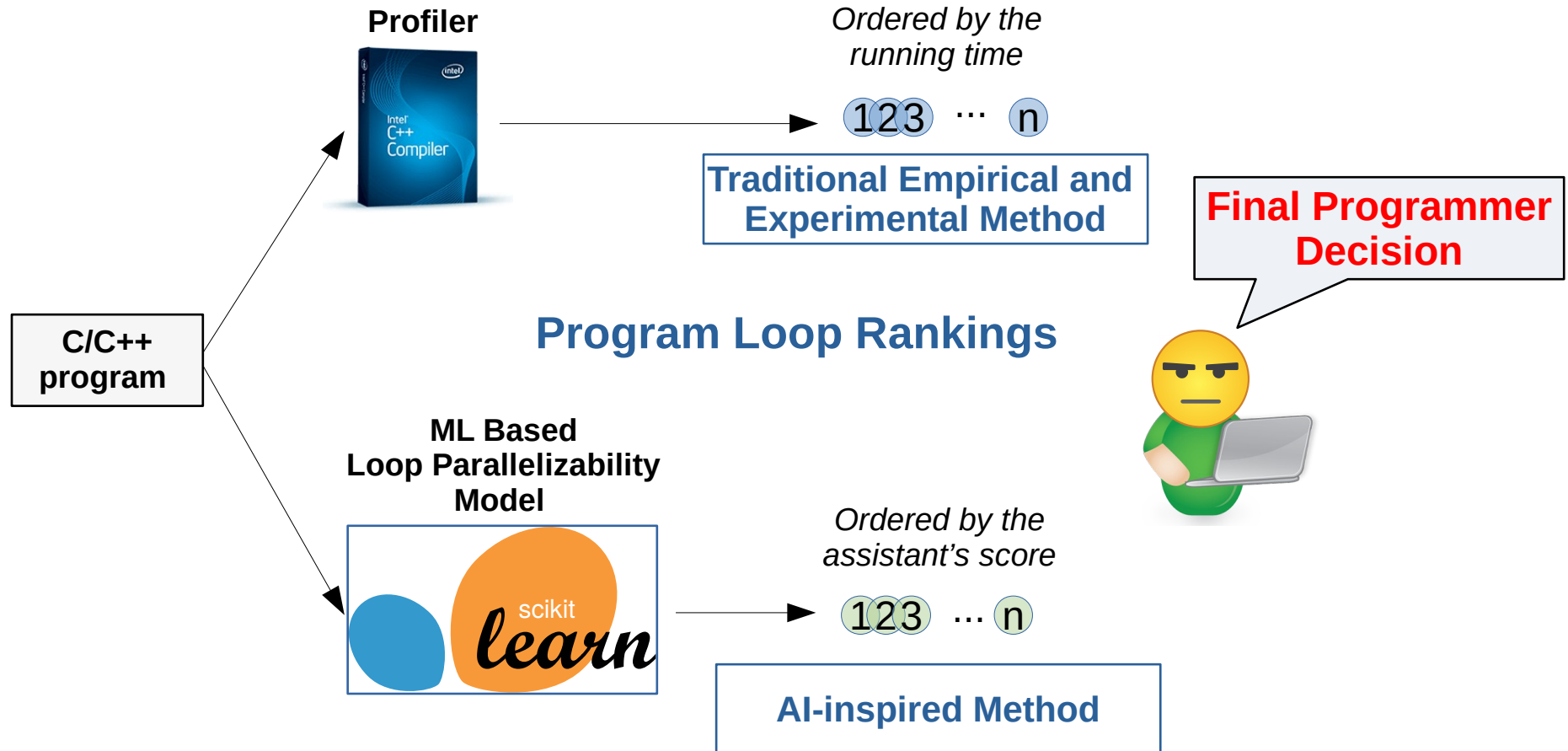
NAS Parallel Benchmarks (NPB)



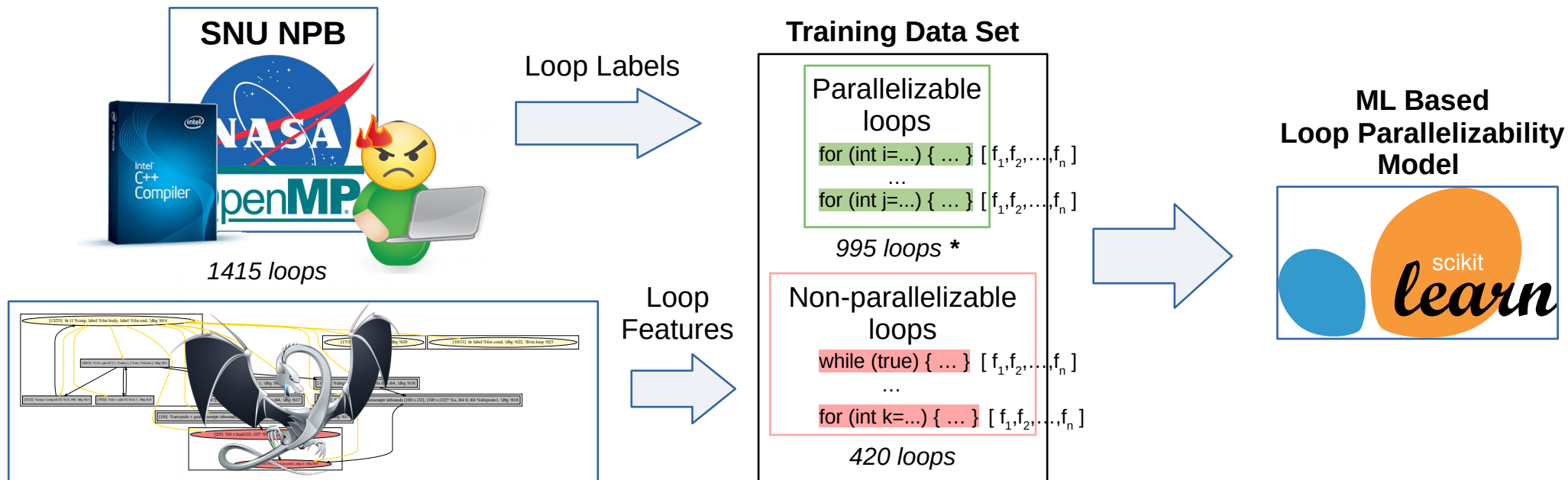
- serial
- || vec
- || par
- || vec_par
- \\ vec_omp
- \\ par_omp
- ⊠ omp
- \\ vec par omp

How to use the assistant

(from the programmers perspective)



Solution Scheme [1/2]



* Intel Compiler succeeds in parallelizing 812 loops

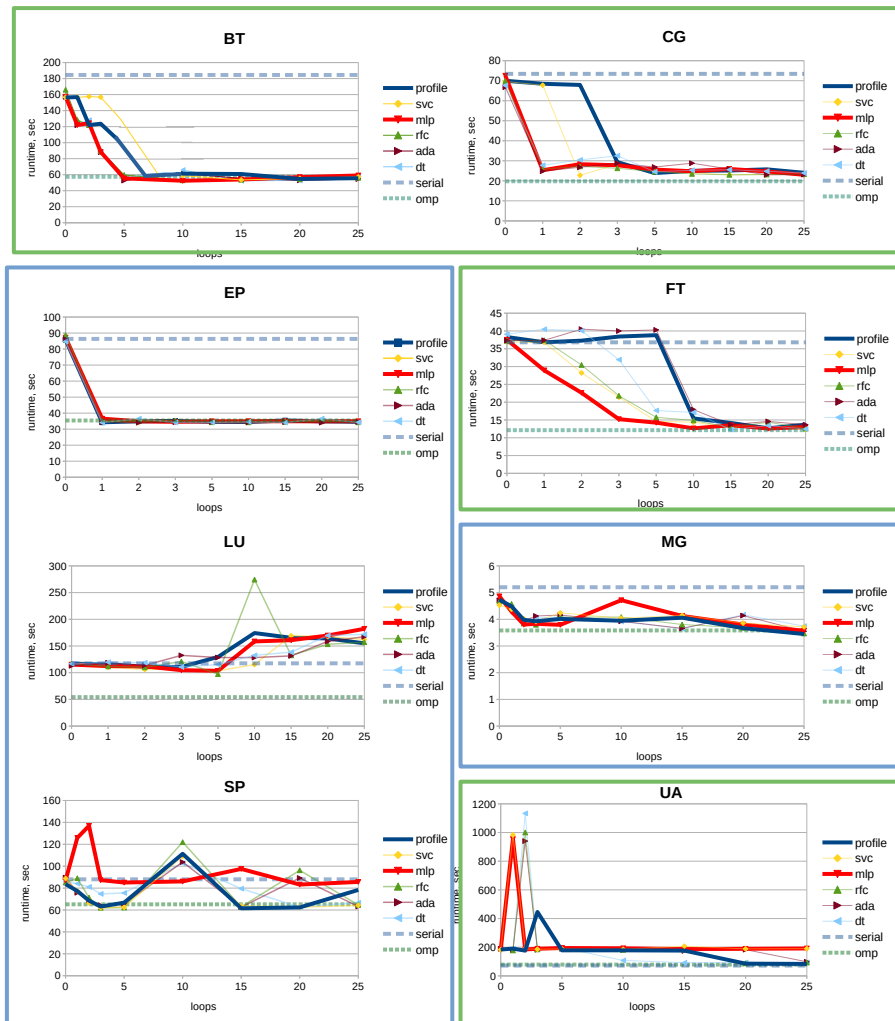
Loop features are based on static structural properties of loop program dependence graphs (PDGs):

- Absolute size
- Loop Iterator/Payload cohesion
- Number of dependence edges
- Instruction types (calls, loads/stores, etc.)

Loop labels are derived out of OpenMP pragmas present in parallelized SNU NPB versions as well as from the Intel Compiler's parallelization/vectorization reports.

How well do we do?

Improvement in
4 benchmarks



No change in
4 other
benchmarks

Problem Statement

NAS Parallel Benchmarks (NPB)

