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

EPSRC
Engineering and Physical Sciences Research Council

THE UNIVERSITY of EDINBURGH informatics

icsa | Institute for Computing Systems Architecture

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

Problem Statement

Parallel Hardware is Ubiqutous	Software is Sequential
Auto Parallelization is Limited	Manual Parallelization is Hard

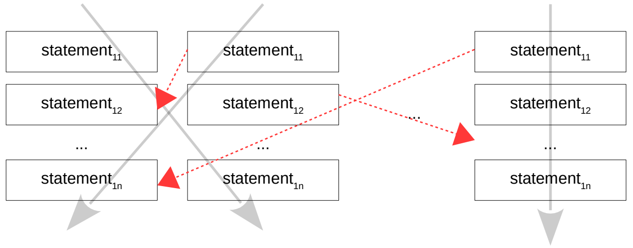
Problem Statement [1]

Hardware is parallel

(across the whole range of computing systems)

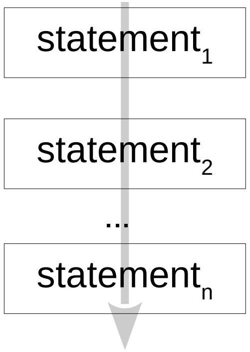
Automatic parallelization is limited

(automatic parallelization does not achieve performance levels of manually parallelized code)



Software is sequential

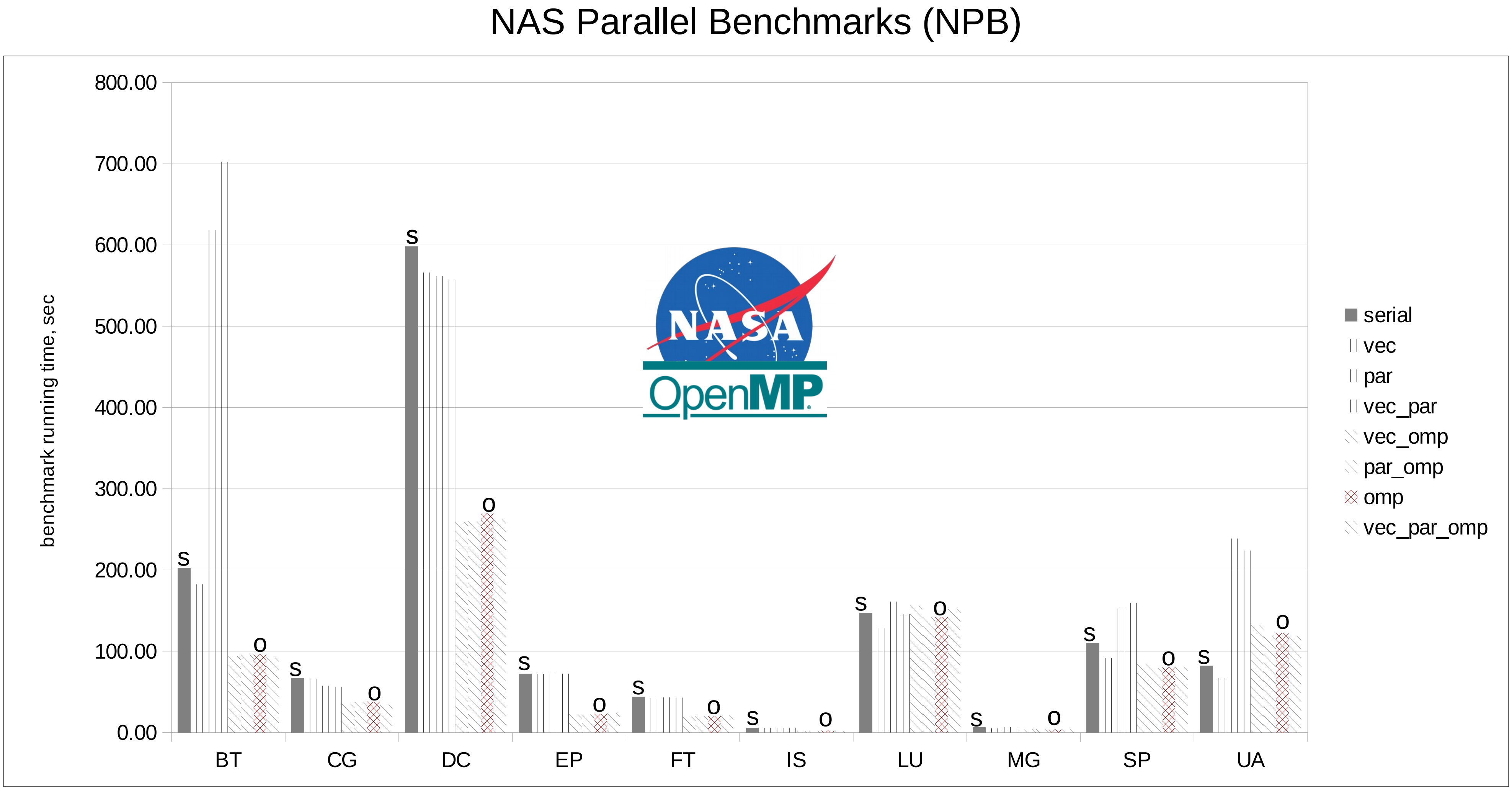
(a great deal of legacy software is written in a sequential fashion)



Parallel programming is hard

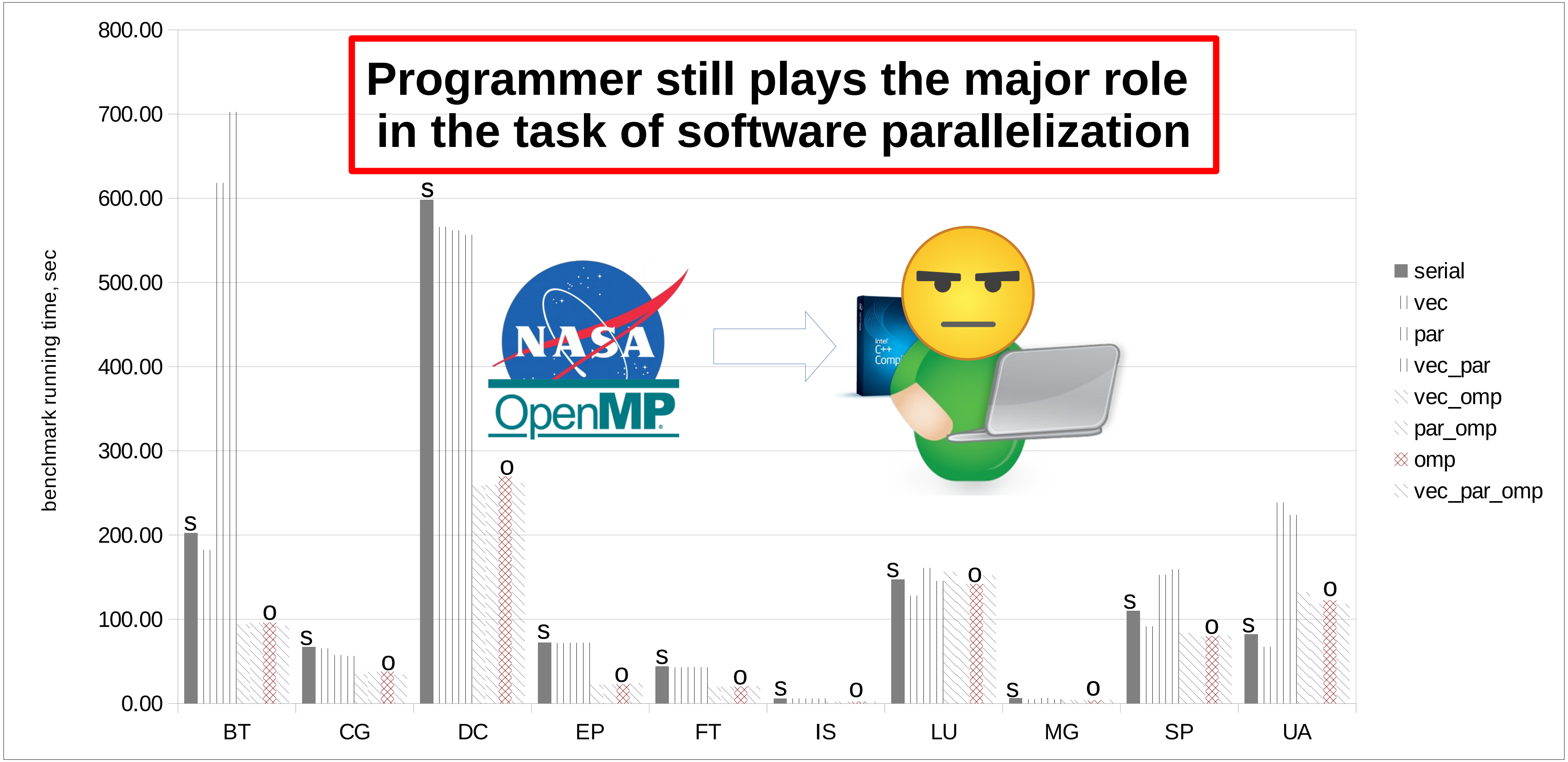
(requires application domain expertise, familiarity with parallel programming in general and exact frameworks (OpenMP, MPI) in particular, etc.)

Problem Statement [2]



Problem Statement [2]

NAS Parallel Benchmarks (NPB)



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

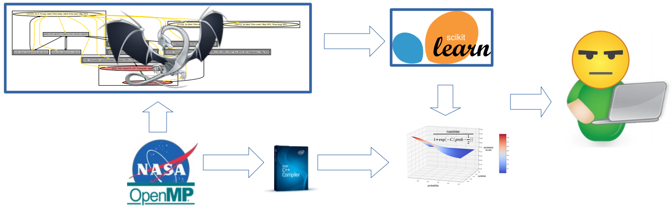
Proposed Solution

(*we propose a novel assistant tool and the methodology to alleviate the process of manual software parallelization*)

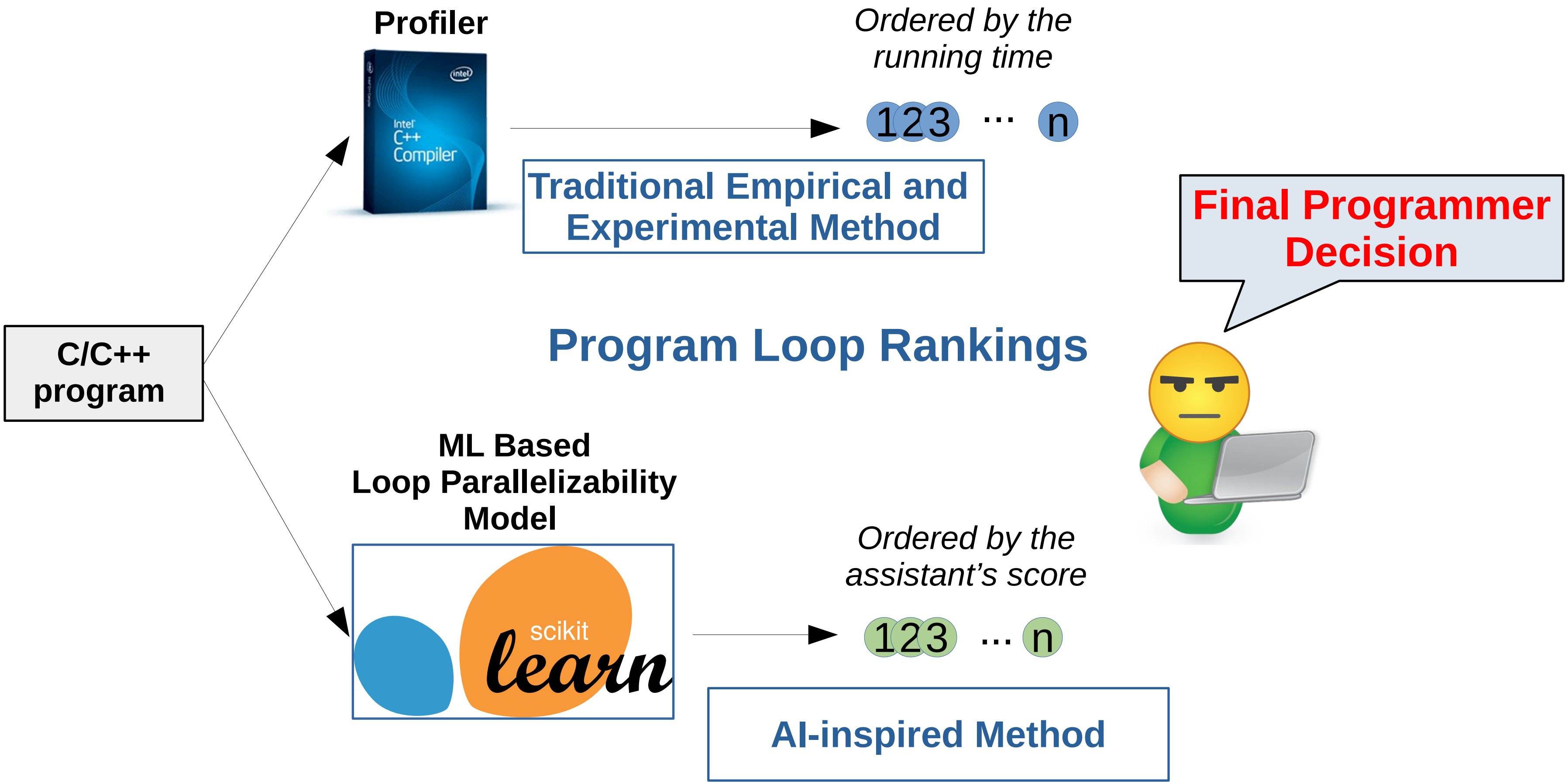
Tool

Methodology

&

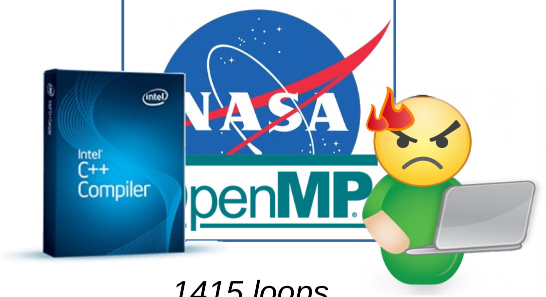


How to use the assistant



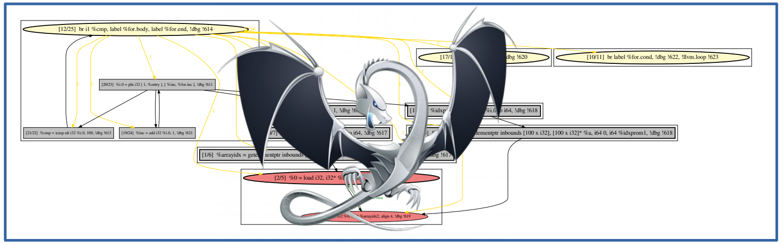
Solution Scheme [1]

SNU NPB



1415 loops

Loop Labels



Loop Features

Training Data Set

Parallelizable loops

for (int i=...) { … } [$f_1, f_2, \ldots, f_n$]
…
for (int j=...) { … } [$f_1, f_2, \ldots, f_n$]

*995 loops **

Non-parallelizable loops

while (true) { … } [$f_1, f_2, \ldots, f_n$]
…
for (int k=...) { … } [$f_1, f_2, \ldots, f_n$]

420 loops

** Intel Compiler succeeds in parallelizing 812 loops*

ML Based Loop Parallelizability Model

scikit learn

Loop features are based on static structural properties of loop program depencence 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.

Solution Scheme [2]

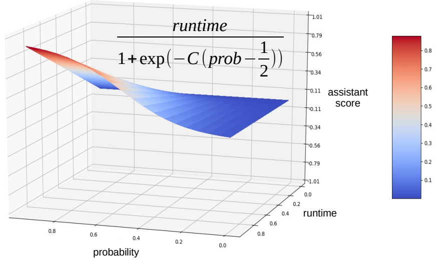
Profiler

Potential Contribution to Speedup

ML Based Loop Parallelizability Model

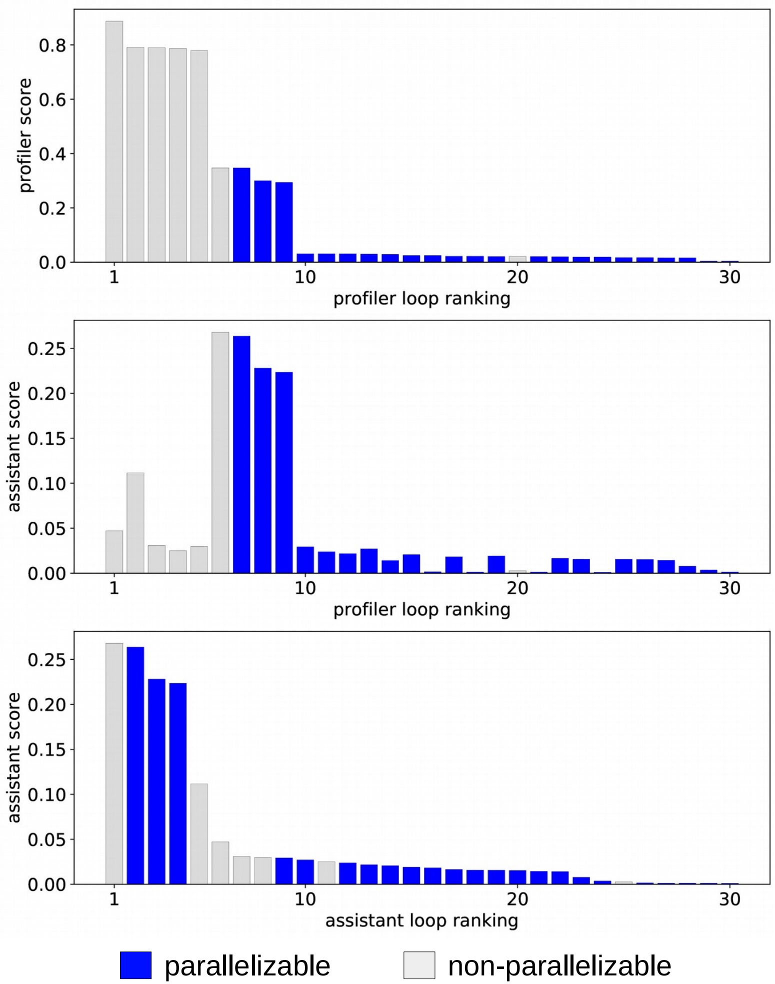
scikit learn

Predicted Probability of Parallelization



Improved Loop Ranking

Solution Scheme [2]

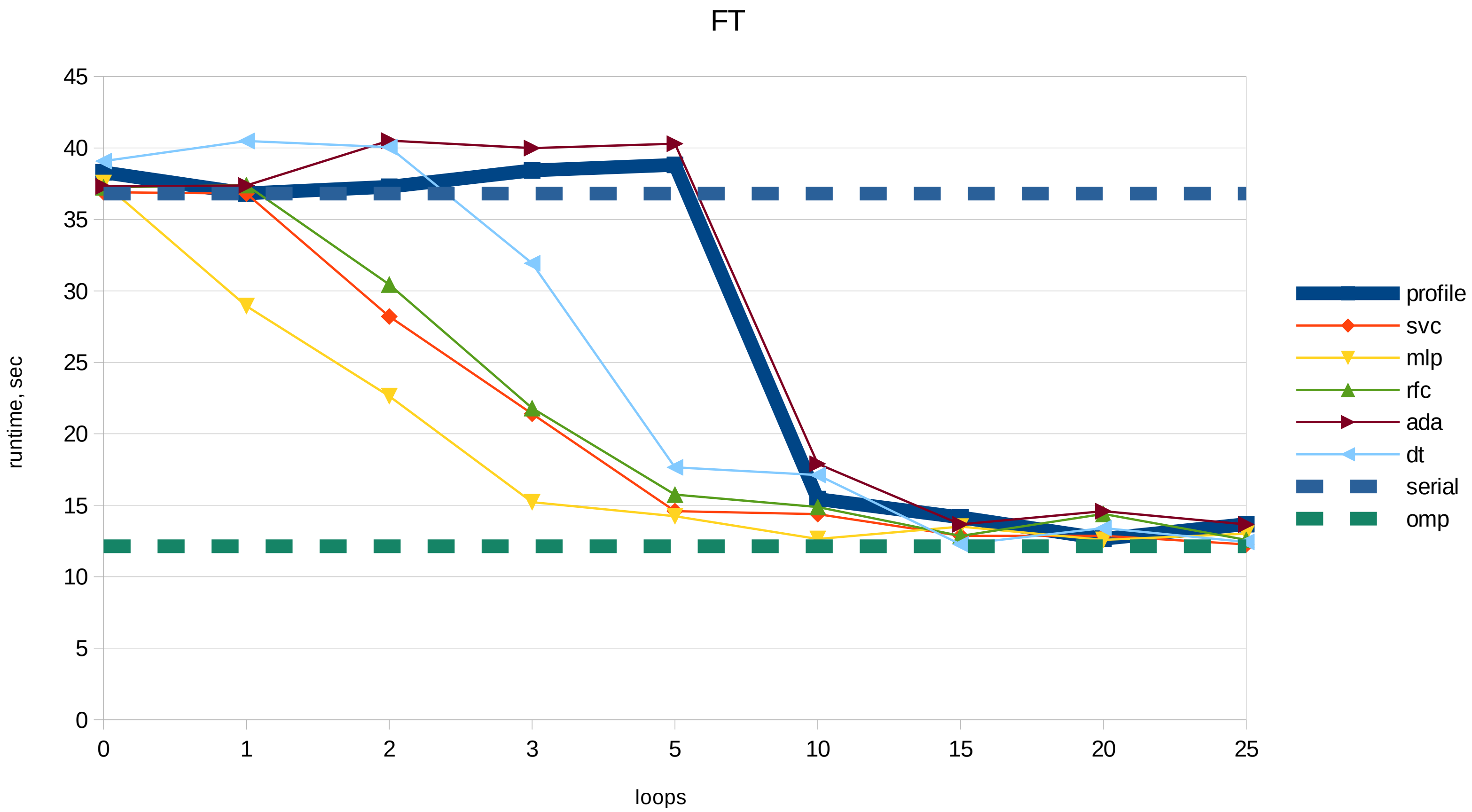


Results

Predictive performance of our ML based loop parallelizability model

Deployment of our assistant on SNU NPB benchmarks

Assistant Deployment

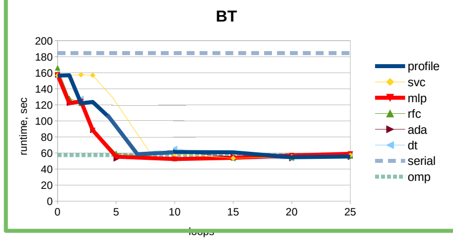


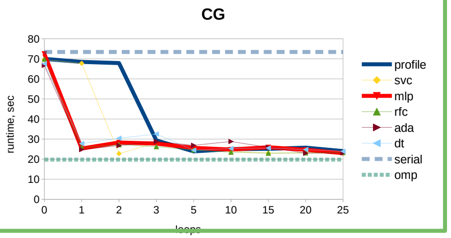
Manual software parallelization faster

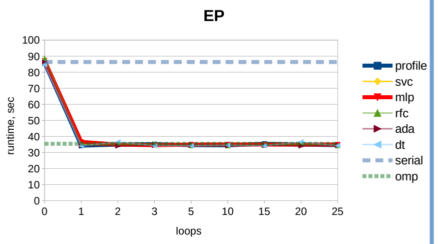
(examine and parallelize 20% fewer Lines Of Code (LOC) to get to the best achievable performance)

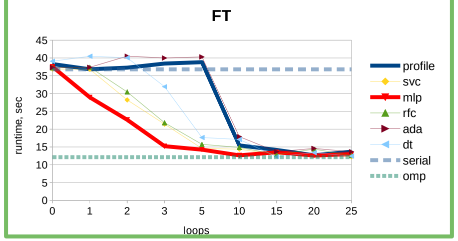
Assistant Deployment

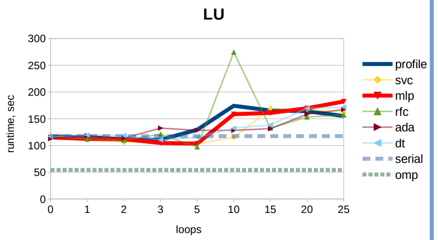
Improvement in 4 benchmarks

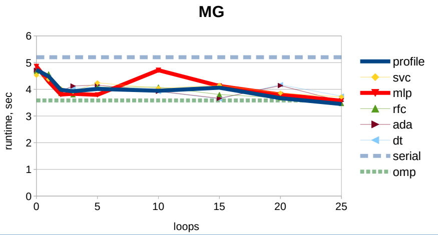


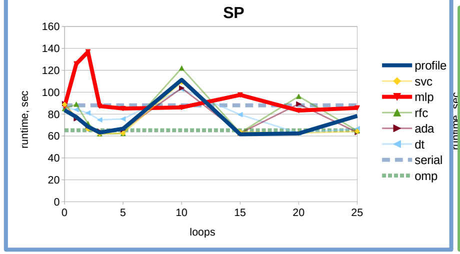


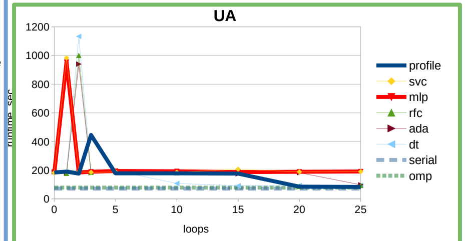








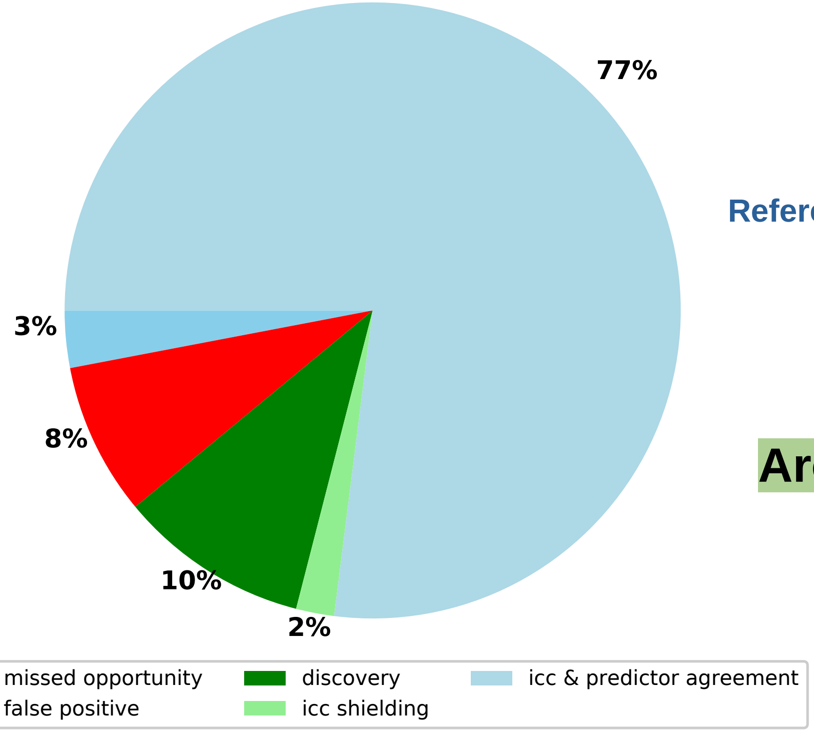




No change in 4 other benchmarks

Predictive Performance

ML Based Loop Parallelizability Model



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

Agreement rate of 80%

Discovery rate of 10%

False positive rate of 8% is not critical

Summary & Conclusions

- Despite decades of research into parallelizing compiler technology human experts still play the major role in the task
- Loop parallelizability is learnable property (we created ML model and trained it to work with the accuracy of above 90%)
- ML model of loop parallelizability has been harnessed into an assistant scheme guiding a programmer towards the best achievable performance
- Deployed against SNU NPB benchmarks our assistant showed a faster convergence: 20% Lines Of Code (LOC) reduction

Thank you!

Motivating example

SNU NPB Conjugate Gradient (CG) benchmark

Profiler's Loop Ranking
1 cg.c:326
2 cg.c:484
3 **cg.c:509**

VS.

Assistant's Loop Ranking	Assistant's Parallel Loop Probability
1 **cg.c:509**	**[85%]**
2 cg.c:326	[29%]
3 cg.c:484	[8%]

```
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,&rnorm);
  if (timeron) timer_stop(T_conj_grad);
  ...
  printf("    %5d       %20.14E%20.13f\n", it,
    rnorm, zeta);
  ...
}
```

cg.c:326