mlin: Rethinking and Rebooting `gprof` for the Multicore Era

nino Garcia, Donghwan Jeon, Chris Louie, Michael B. Taylor

outer Science & Engineering Department
rsity of California, San Diego

tivating a "`gprof` for parallelization"

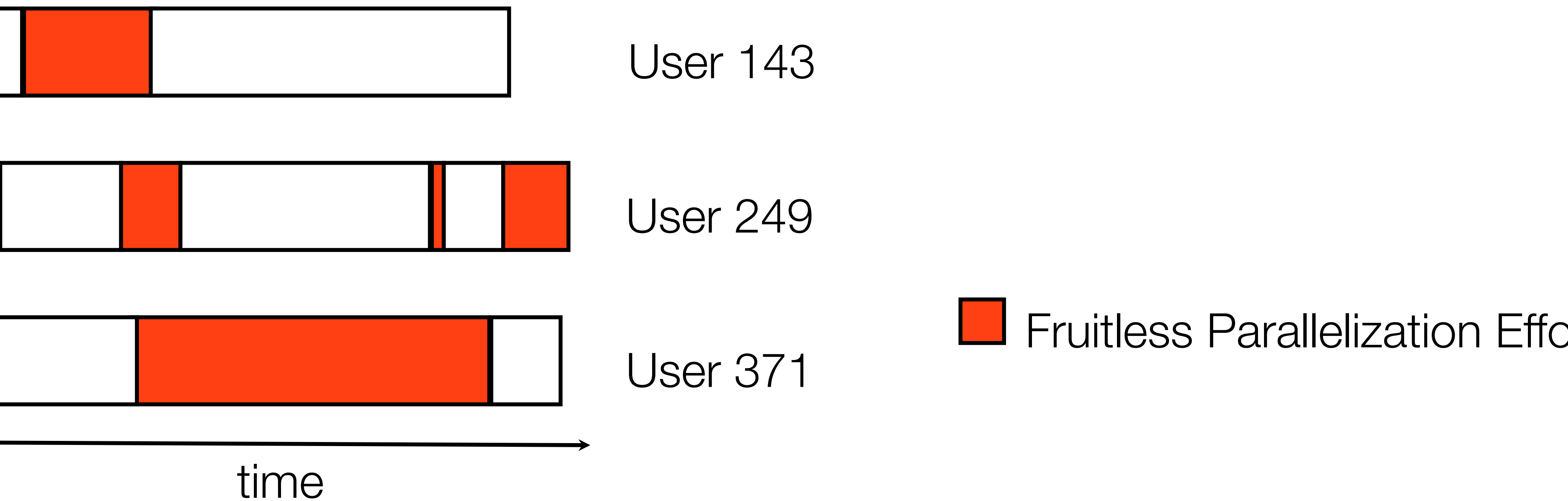
w effective are programmers at picking the right parts of a gram to parallelize?

- User study* we performed at UC San Diego (UCSD IRB #100056)
- First and second year CS graduate students
- Users parallelize their programs and submit to job queue for timing
- 32-core AMD machine, Cilk++, access to gprof
- Students were graded based on effectiveness of their parallel spee students told serial optimization would not help their grade

**Disclaimer: No graduate students w*

er Study: Results

amined student's activities to determine result of efforts



nificant fraction of fruitless effort because of three basic probl

Low Parallelism: Region was not parallel enough

Low Coverage: Region's execution time was too small

gprof answers the question:

"What parts of this program should I spend time ***optimizing****?"*

Kremlin answers the question:

"What parts of this program should I spend time ***parallelizing****?"*

mlin's Usage Model

age model inspired by gprof

1. Compile instrumented binary

```
ake CC=kremlin-cc
```

2. Profile with sample input

```
/tracking lolcats
```

3. Run analysis tool to create

```
remlin tracking --personality=openmp
```

ile (lines)	Self-P	Cov (%)
mageBlur.c (49-58)	145.3	9.7
mageBlur.c (37-45)	145.3	8.7
etInterpPatch.c (26-35)	25.3	8.9
alcSobel_dX.c (59-68)	126.2	8.1
alcSobel_dX.c (46-55)	126.2	8.1

mlin's Key Components

al
c
de

- *Hierarchical Critical Path Analysis* (HCPA)
  - Quantifies self-parallelism in each program region
- *Self-Parallelism*
  - Estimates ideal parallel speedup of a specific region
- *Planning Personalities*
  - Incorporates target specific constraints in parallelization

Parallelism Discovery

"What's the potential parallel speedup of each part of this program?"

Parallelism Planning

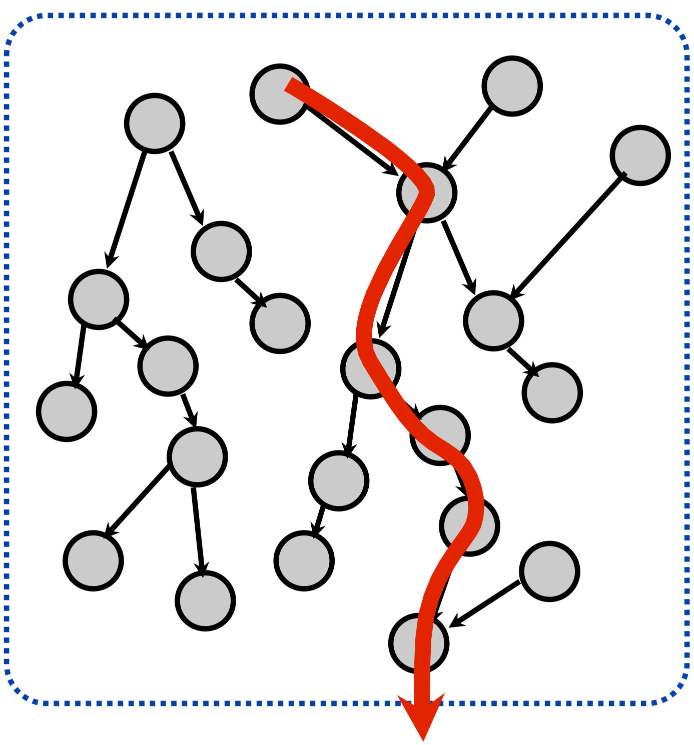
"What regions must I parallelize to get the maximum benefit on this system?"

Parallelization

veloping an Approach for Parallelism Discove

sting Technique: 1980’s-era Critical Path Analysis (CPA)

- Finds critical path through the dynamic execution of a program
- Mainly used in research studies to quantify limits of parallelism



critical path (cp)

$$\textbf{parallelism} = \frac{\textbf{work}}{\textbf{critical path length}}$$

work ~= # of instrs

instruction

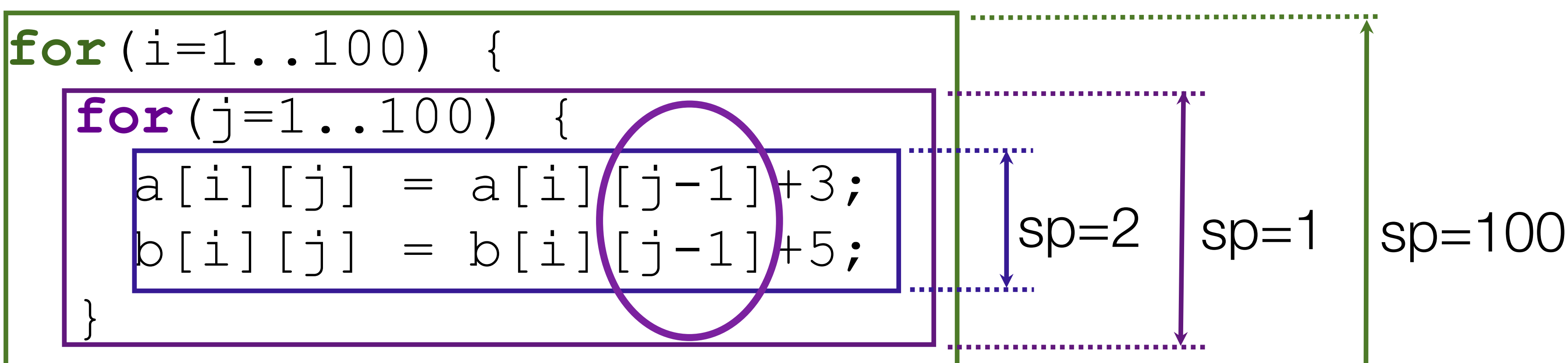
data or control dependence

efits of CPA as a Basis for a Parallelism Disc

- uates program's potential for parallelization under relatively nistic assumptions
  - oser approximation to what human experts can achieve
  - rsus pessimistic static analysis in automatic parallelizing compilers
- lt is predictive of parallelism after typical parallelization formations
  - g., Loop interchange, loop fission, locality enhancement

roving CPA with Hierarchical CPA (HCPA)

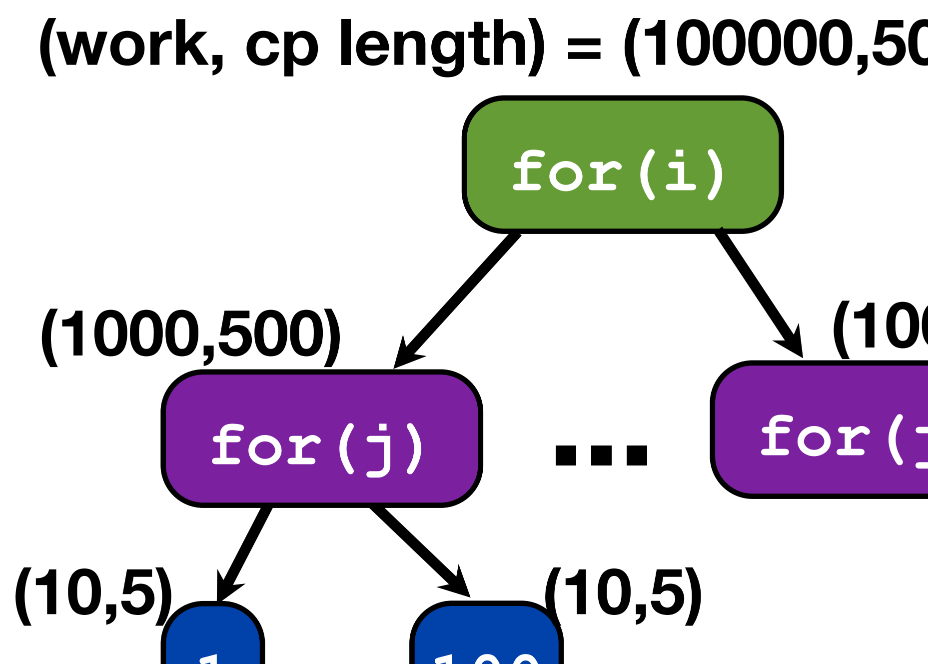
- A is typically run on an entire program
  - Not helpful for identifying specific regions to parallelize
  - Doesn't help evaluate execution time of a program if only a subset he program is parallelized
- rarchical CPA is a region-based analysis
  - Self-Parallelism (sp) identifies parallelism in specific regions
  - Provides basis for estimating parallel speedup of individual regions



PA Step 1: Hierarchically Apply CPA

- **al**: Introduce localization through *region*-based analysis
- adow-memory based implementation
  - Performs CPA analysis on every program region
  - Single pass: Concurrently analyzes multiple nested regions

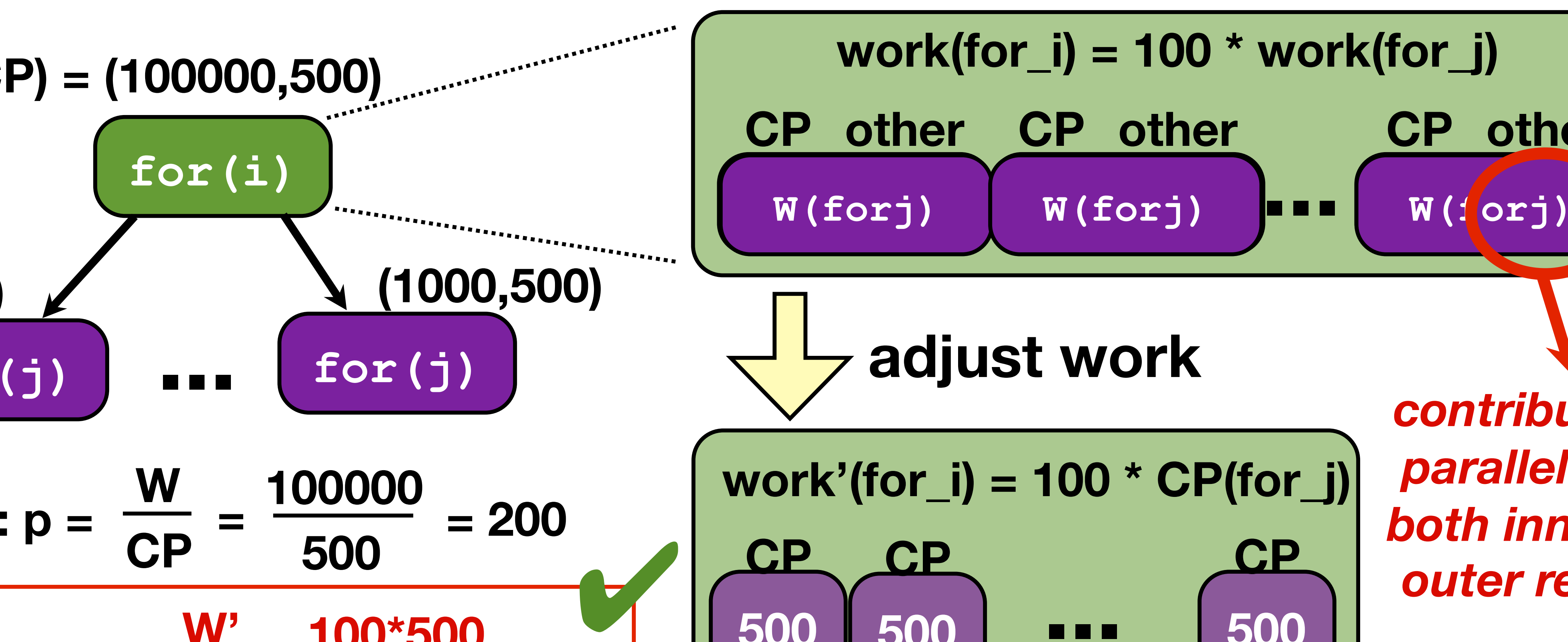
```
1..100) {
j=1..100) {
i][j] = a[i][j-1]+3;
i][j] = b[i][j-1]+5;
```



$$p = \frac{W}{CP} = \frac{100000}{500} = 200$$ ✗

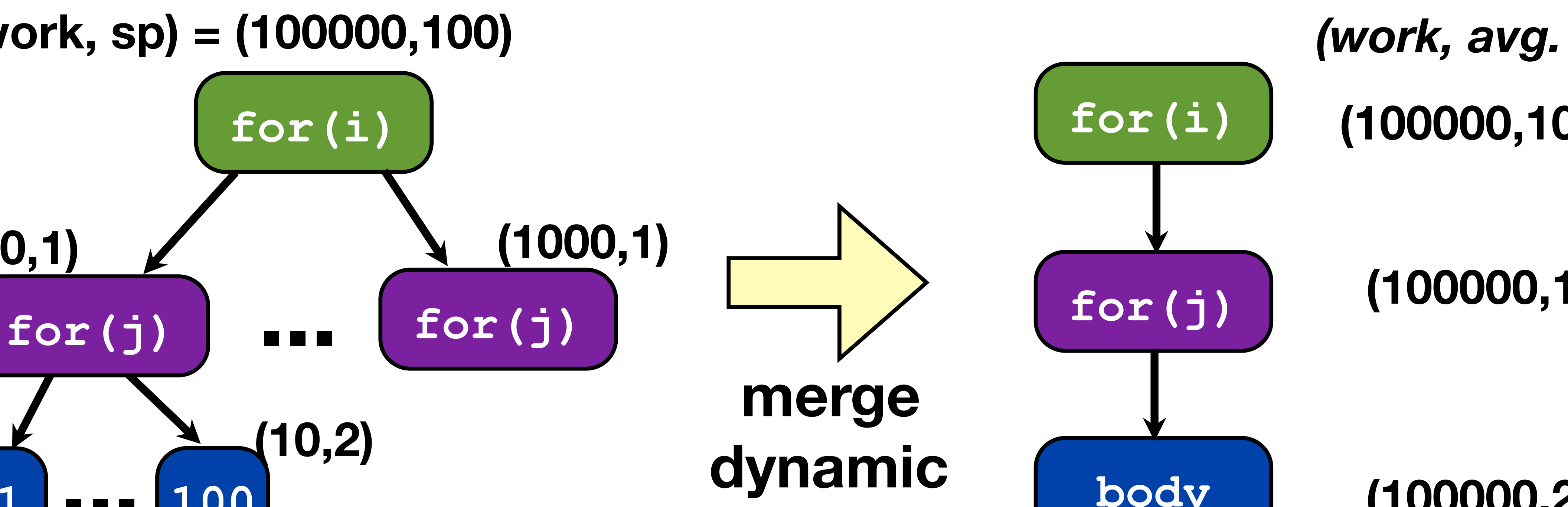
PA Step 2: Calculate Self-Parallelism

- al: Eliminate effect of nested parallelism in parallelism calculat
- proximate self-parallelism using only HCPA output
  - "Subtracts" nested parallelism from overall parallelism



PA Step 3: Compute Static Region Data

- **al:** Convert dynamic region data to static region output
- rge dynamic nodes associated with same static region
  - Work: *Sum* of work across dynamic instances
  - Self-Parallelism: *Weighted Average* across dynamic instances



ther Details on Discovery in Our Paper

- mlin handles much more complex structures than just neste loops: finds parallelism in arbitrary code including recursion

- f-parallelism metric is defined and discussed in detail in the per

- mpression technique used to reduce size of HCPA output

ating a Parallelization Plan

al: Use HCPA output to select best regions for target syster

nning personalities allow user to incorporate system constra

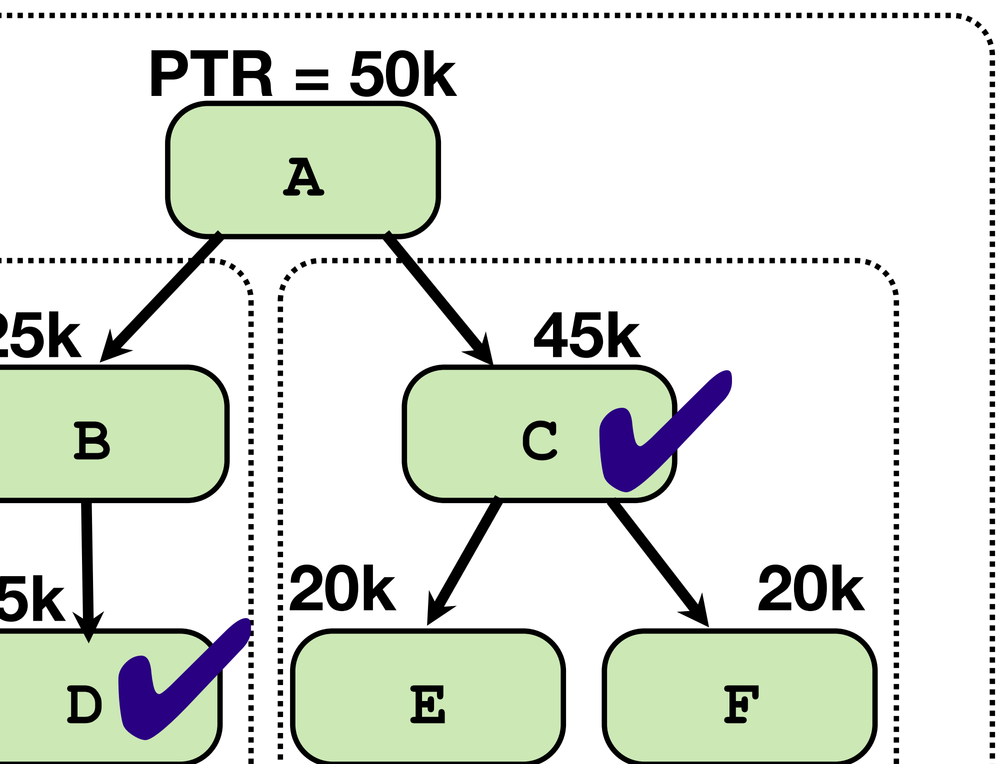
Software constraints: What types of parallelism can I specify?

Hardware constraints: Synchronization overhead, etc.

Planning algorithm can change based on constraints

OpenMP Planner

- sed on OpenMP 2.0 specification
  - Focused on loop-level parallelism
  - Disallows nested parallelism because of overhead
  - Planning algorithm based on dynamic programming



parallelized time reduction = W - (W/

Region	Work	SP
A	100k	2
B	50k	2
C	50k	10
D	50k	10
E	25k	5

luation

thodology:

- Ran Kremlin on serial versions; targeting OpenMP
- Parallelized according to Kremlin's plan
- Gathered performance results on 8 socket AMD 8380 Quad-core
- Compared against third-party parallelized versions (3rd Party)

nchmarks: NAS OpenMP and SpecOMP

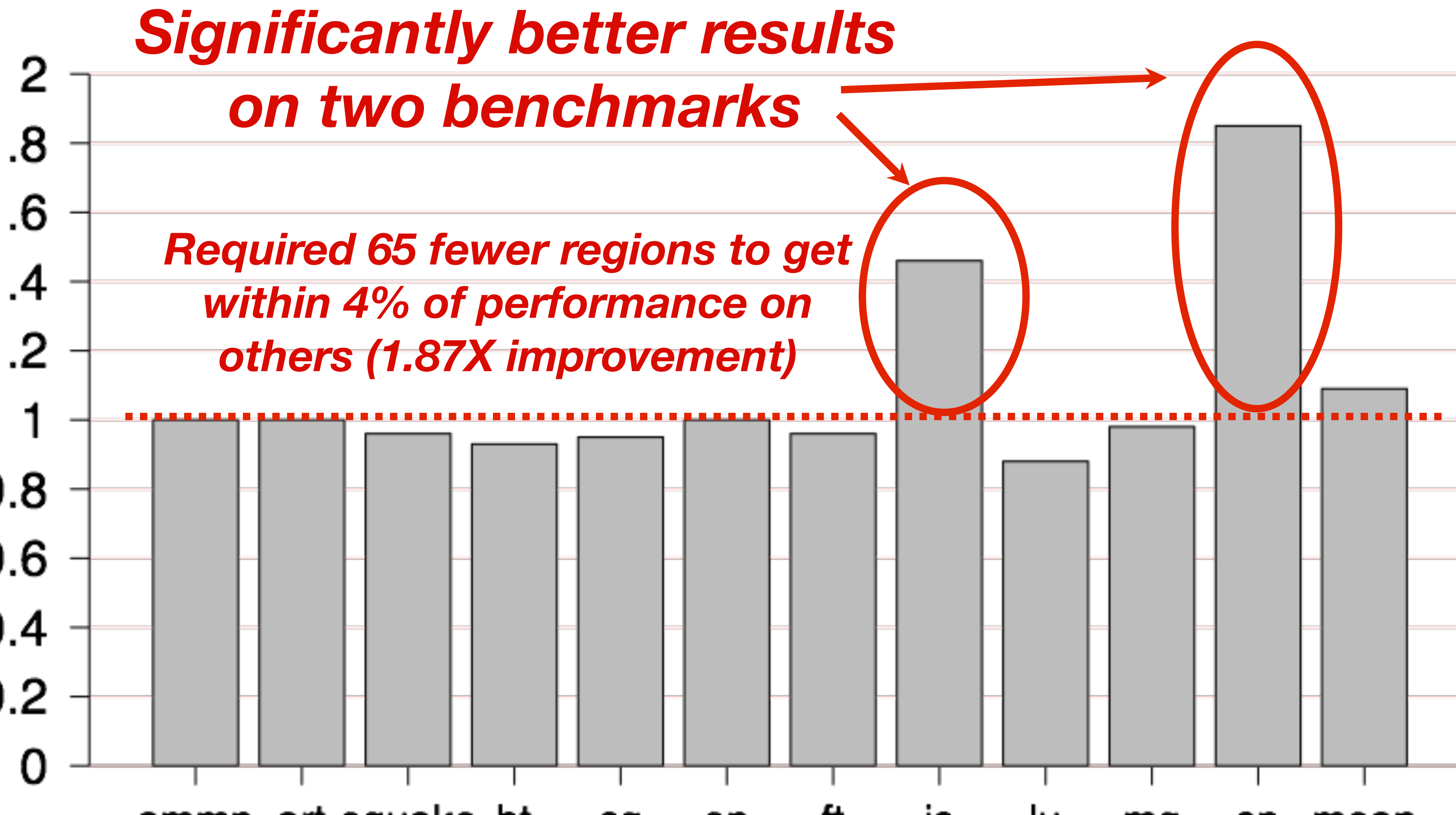
- Have both serial and parallel versions
- Wide range of parallel speedup (min: 1.85x, max: 25.89x) on 32 cor

v much effort is saved using Kremlin?

		# of Regions Parallelized		
Suite	Benchmark	3rd Party	Kremlin	**Reduction**
ecOMP	art	3	4	0.75x
	ampp	6	3	2.00x
	equake	10	6	1.67x
NPB	ep	1	1	1.00x
	is	1	1	1.00x
	ft	6	6	1.00x
	mg	10	8	1.25x
	cg	22	9	2.44x
	lu	28	11	2.55x
	bt	54	27	2.00x
	sp	70	58	1.21x
	Overall	**211**	**134**	**1.57x**

v good is Kremlin-guided performance?

mpared performance against expert, third-party version



es Kremlin pick the best regions first?

ermined what % of speedup comes from first {25,50,75,100
ecommended regions

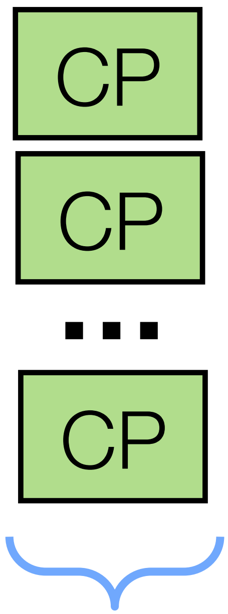
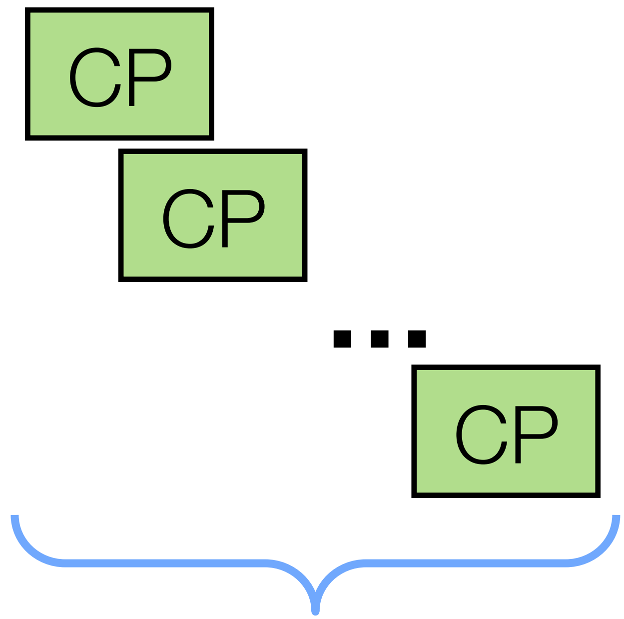
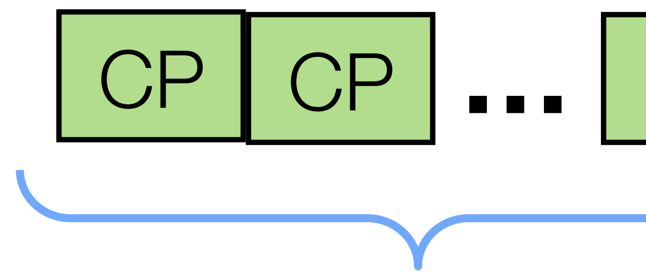
	Fraction of Kremlin Plan Applied			
	First 25% of regions	**Second 25%** of regions	**Third 25%** of regions	**Last 25%** regions
Marginal benefit (% ax speedup) (avg)	56.2%	30.2%	9.2%	4.4%

nclusion

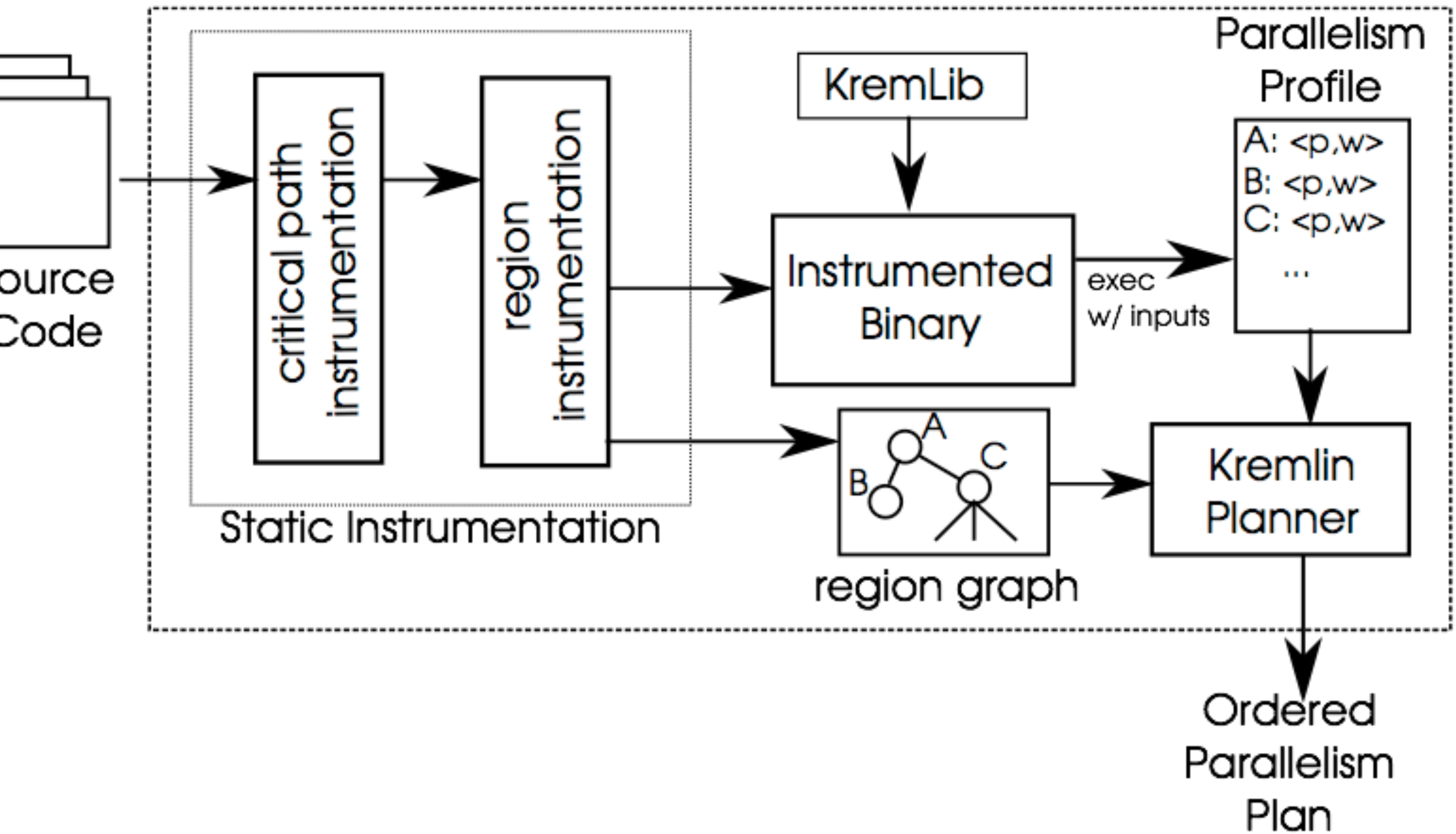
- mlin helps a programmer determine:
  - *"What parts of this program should I spend time parallelizing*
- ee key techniques introduced by Kremlin
  - *Hierarchical CPA*: How much total parallelism is in each region?
  - *Self-Parallelism*: How much parallelism is only in this region?
  - *Planning Personalities*: What regions are best for my target system
- mpelling results
  - .57x average reduction in number of regions parallelized
  - Greatly improved performance on 2 of 11 benchmarks; very close o others

***

Parallelism for Three Common Loop Types

ype	DOALL	DOACROSS	Serial
ath CP)	 CP	 (N/2) * CP	 N * CP
–')	N * CP	N * CP	N * CP
m	$\frac{N * CP}{CP}$ = **N**	$\frac{N * CP}{(N/2) * CP}$ = **2.0**	$\frac{N * CP}{N * CP}$ = **1.**

mlin System Architecture



rpreting the Parallelism Metric