Making Autoparallelizers Mainstream Tools

Rudi Eigenmann
Purdue University

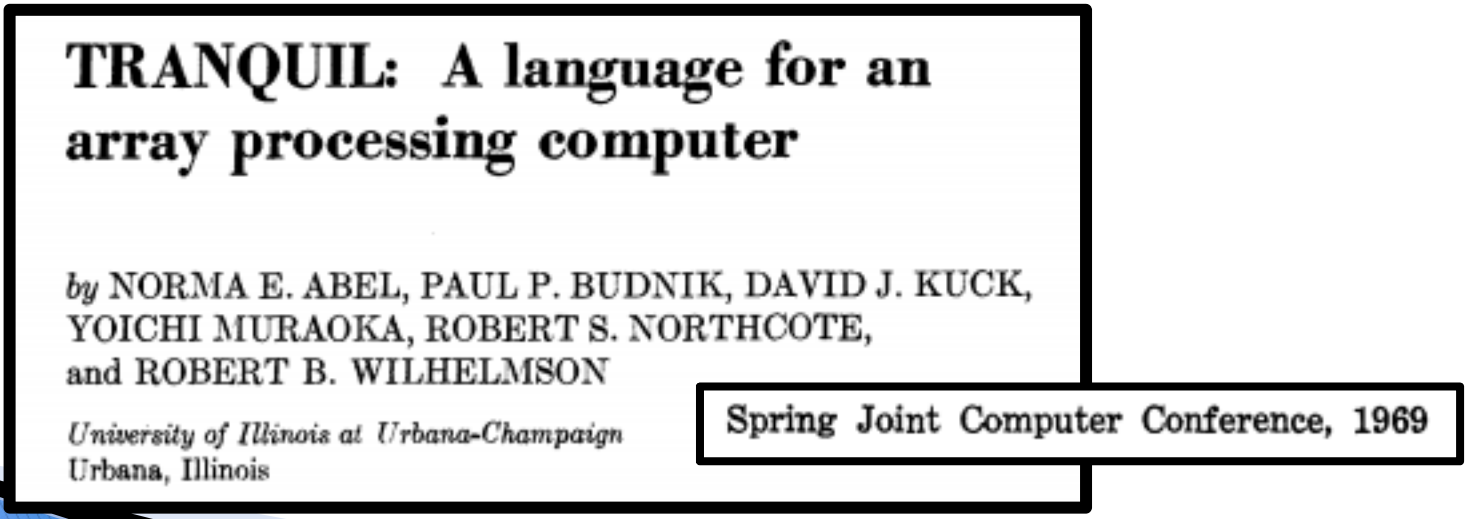
Another Way of Asking

After 30+ years of autoparallelization research

Have we done something useful?

Remember ?

- The 80s: foundational
  - Kuck, Kennedy, Banerjee, Padua, Muraoka
  - Wolfe’s Parafrase I, PFC



TRANQUIL: A language for an array processing computer

by NORMA E. ABEL, PAUL P. BUDNIK, DAVID J. KUCK, YOICHI MURAOKA, ROBERT S. NORTHCOTE, and ROBERT B. WILHELMSON

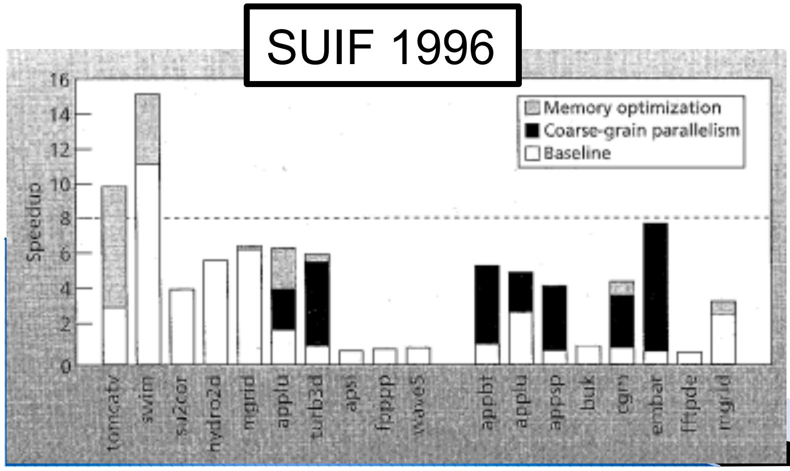
University of Illinois at Urbana-Champaign
Urbana, Illinois

Spring Joint Computer Conference, 1969

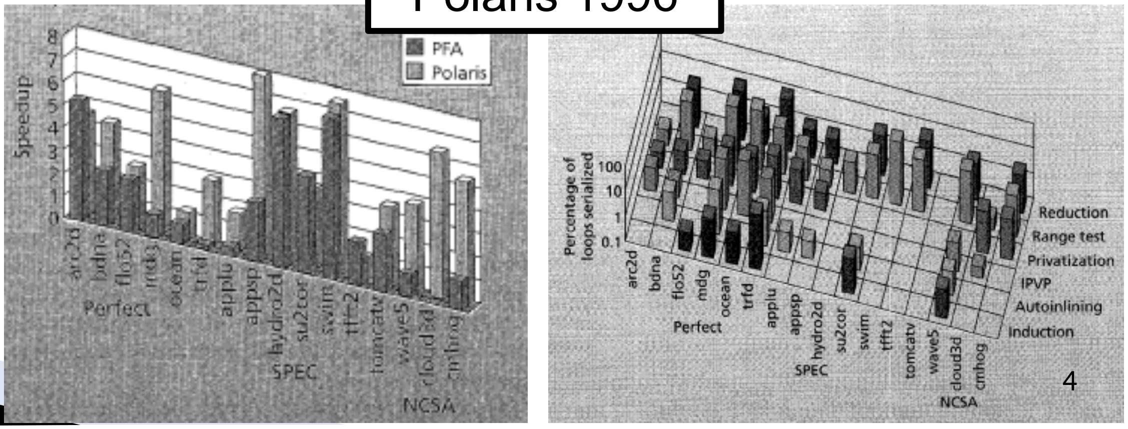
Remember ?

- The 90s: excitement and frustration
  - Success on real benchmarks
  - Polaris
  - SUIF, Oscar, Parascope, HPF . . .
  - National Compiler Infrastructure
  - Success or Failure?
    - “-O is too much user interaction with the compiler”
    - “the only impact of paralleliers is to train programmers…”

SUIF 1996

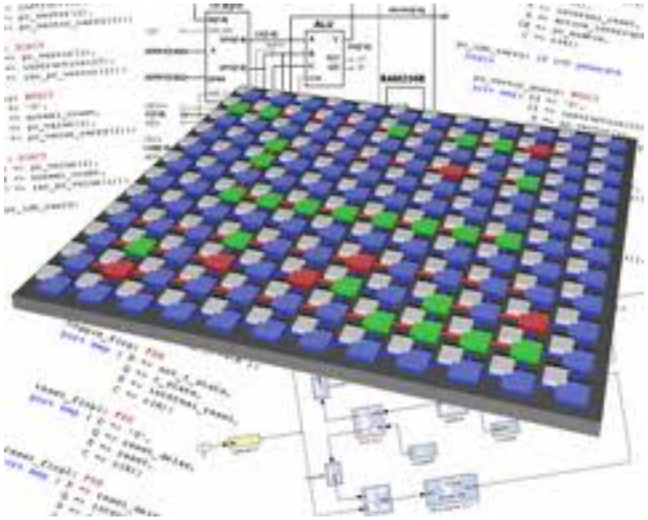


Polaris 1996



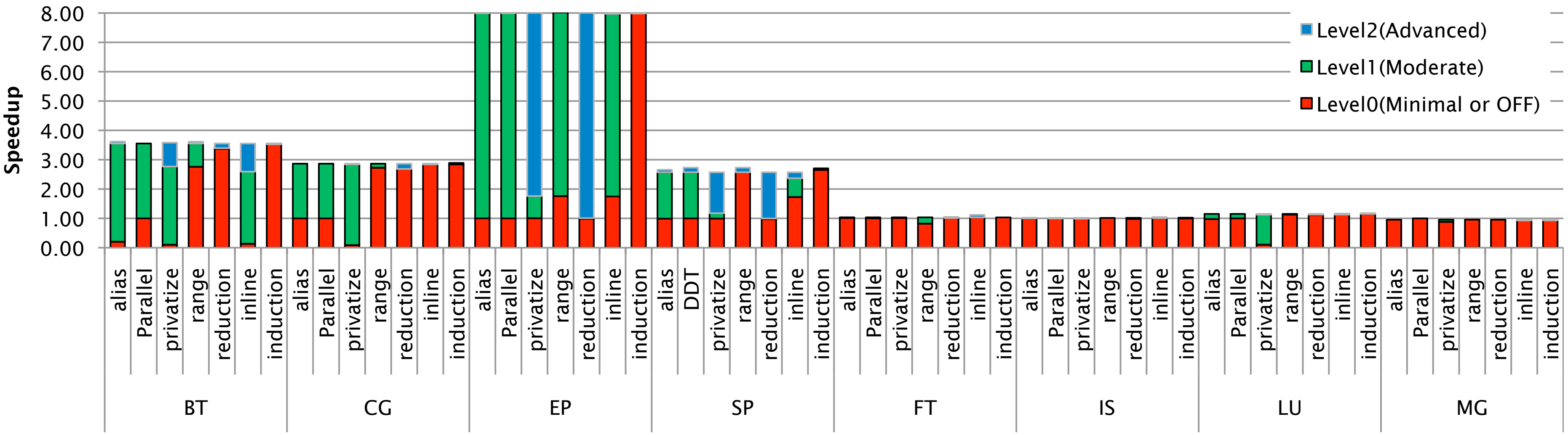
Remember ?

- The New Millennium: renewed interest
  - Multicore is game changer
  - Memory wall growing
  - Cetus, Rose, OpenUH, …
  - Can we deliver?



State of Today’s Autoparallelizers

What’s in a parallelizer?



State of Today's Autoparallelizers

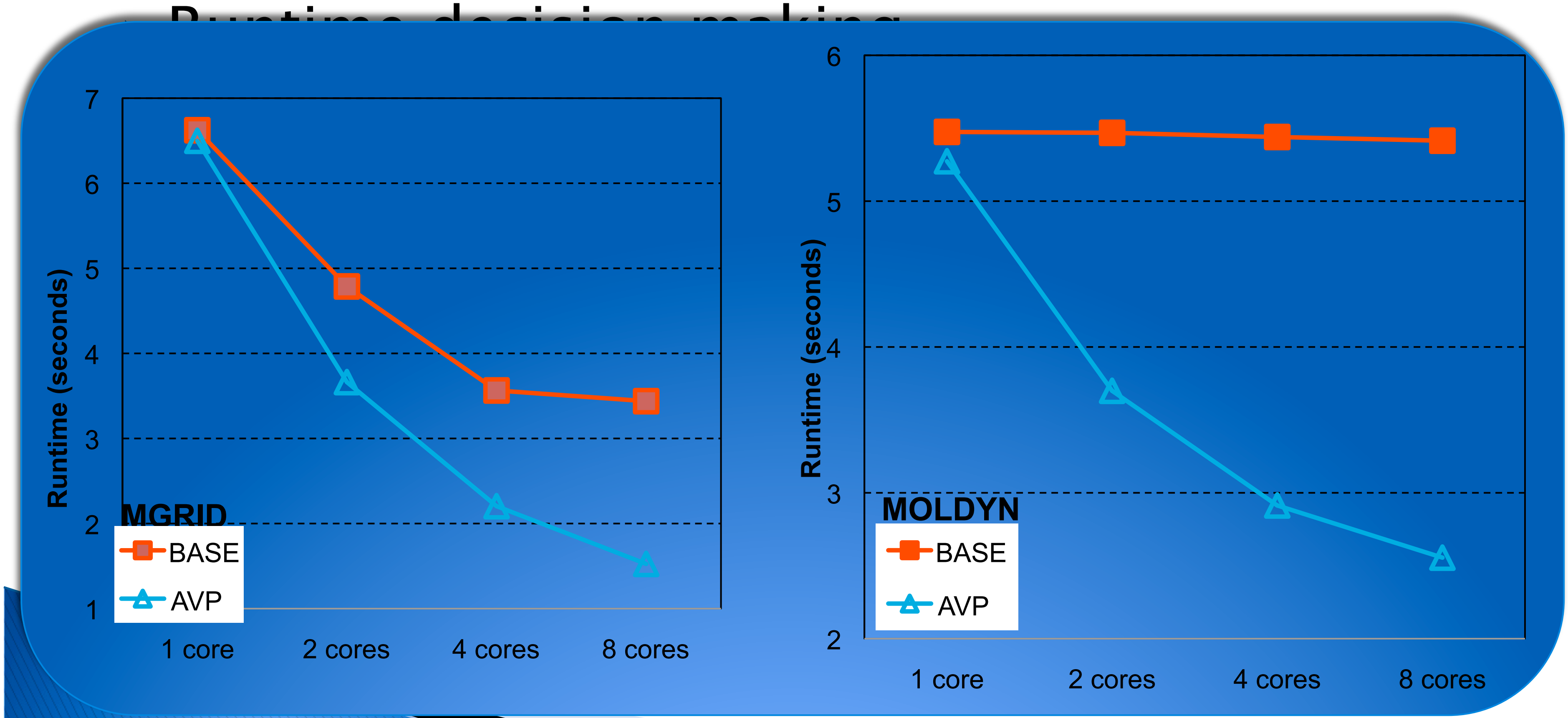
- There are "autopar" compiler options
  - They are not the default
  - Parallelization may *degrade* performance
  - You have to experiment to see if they are useful
  - Do industrial compilers include advanced parallelization techniques?
  - Are research compilers any better?
  - 50% success in numerical apps

=> autoparallelizers are not mainstream tools

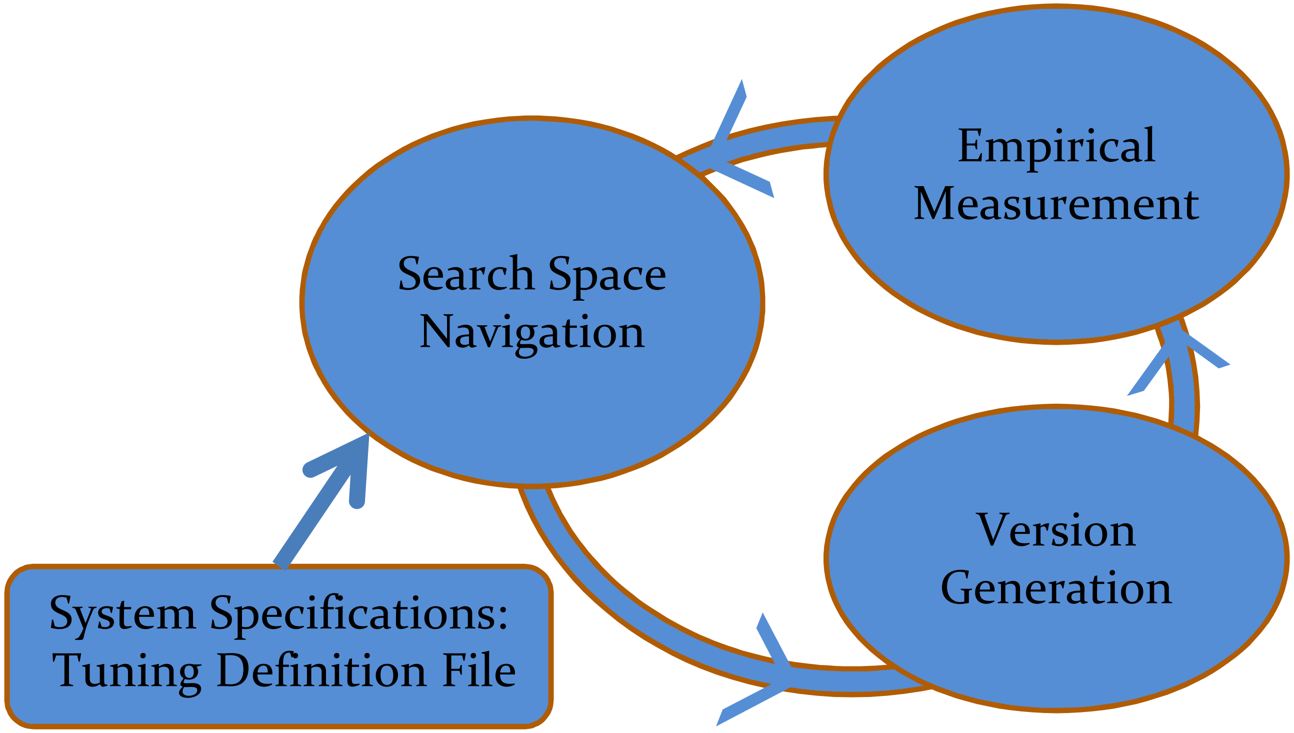
What Stands in the Way

of making parallelizers mainstream tools?

- Advanced techniques
  - E.g., Symbolic array value analysis
- Runtime decision making



Model of Empirical Tuning



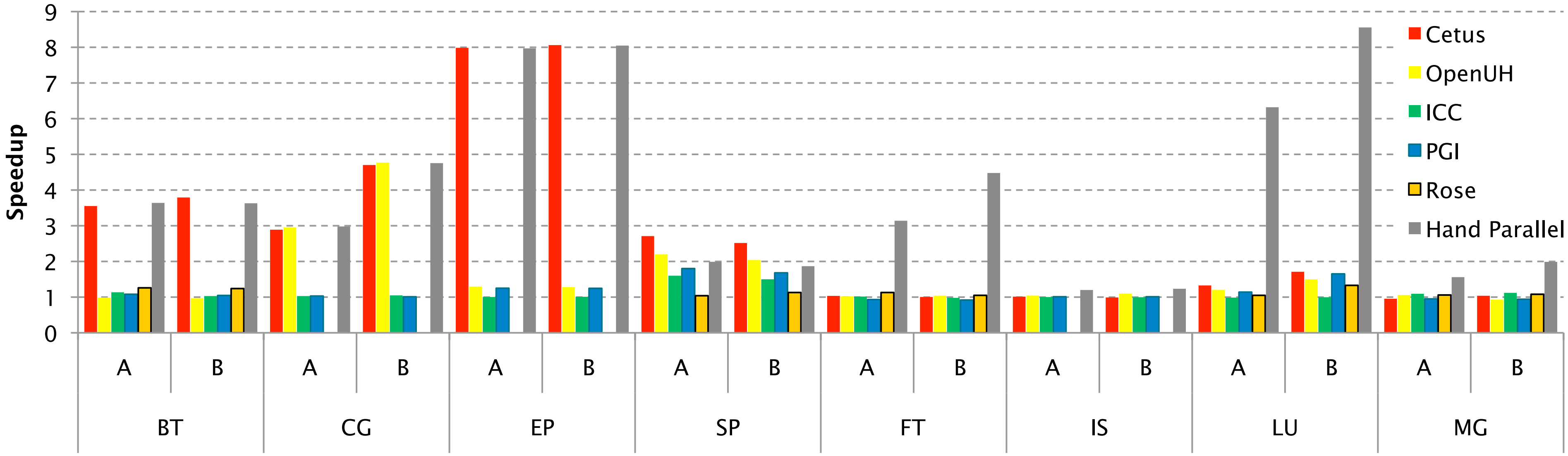
Plugging in a Compiler

```
!Loop-Level Optimization Options:
loop_tile   1              tile_size    [4:256:*=4]
loop_unroll 1 unroll_size [2:16:*=2]
loop_parallelize 0
vec          1 vec_threshold [50:100:+=10]

!Program level Optimization Options
reduction 0

!Options’ Dependencies
loop_tile  loop_parallelize

!Windowing Strategy
fixed 3

!Environment Variables
OMP_NUM_THREADS [1:8]

!Make Definition
. . .
```

Tuning Definition File

=> Turn a compiler into a tuning tool with a few 10s of lines

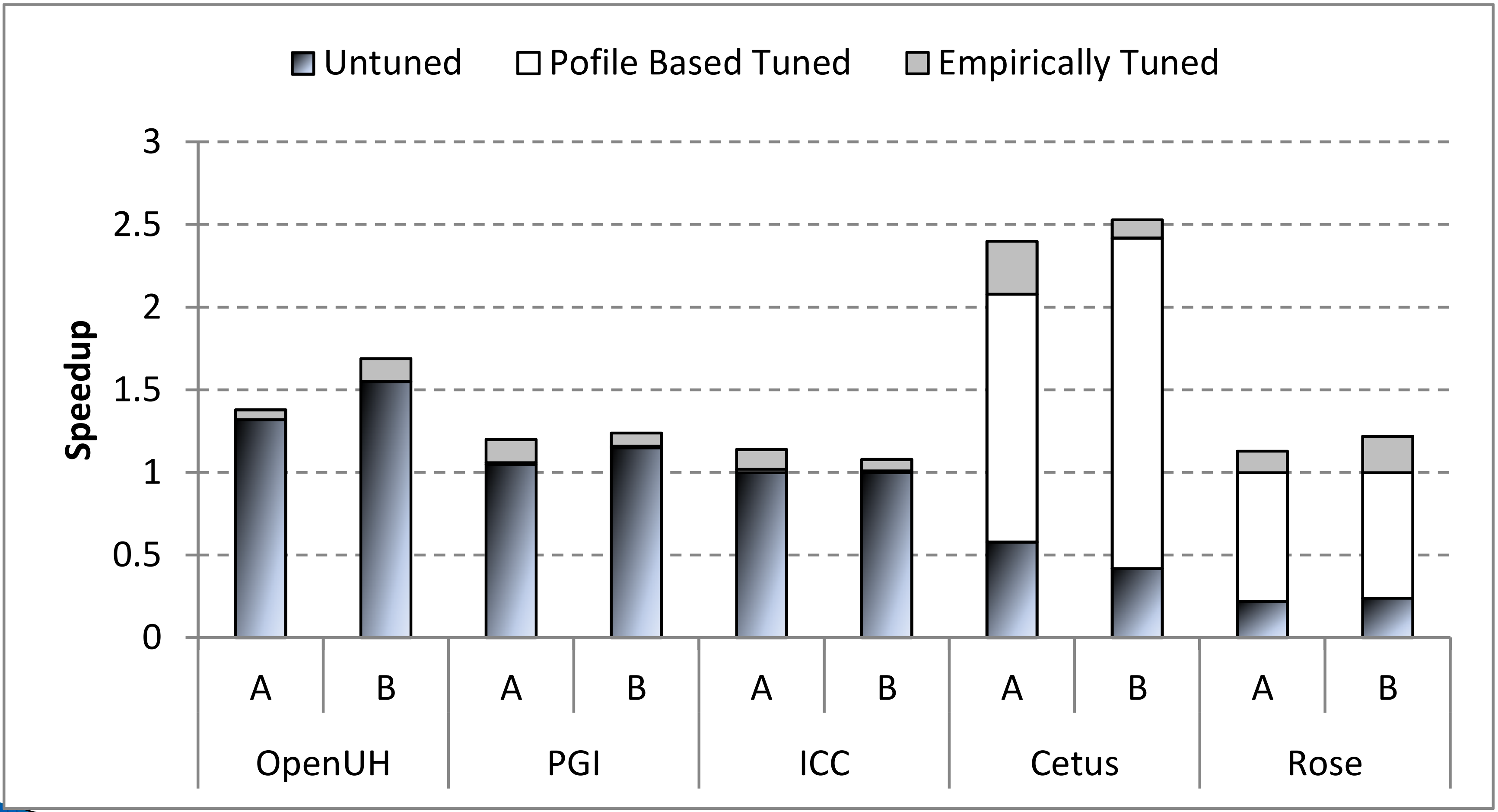
Overall Performance



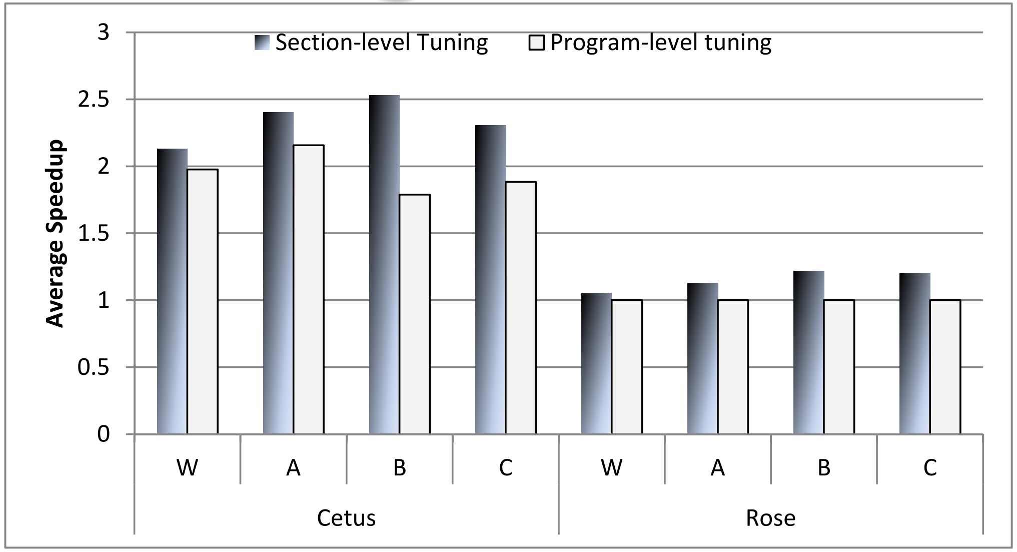
Tuned autoparallelized performance is always >= original performance

=> Can leave autoparallelization ON by default!

Tuning Makes the Key Difference

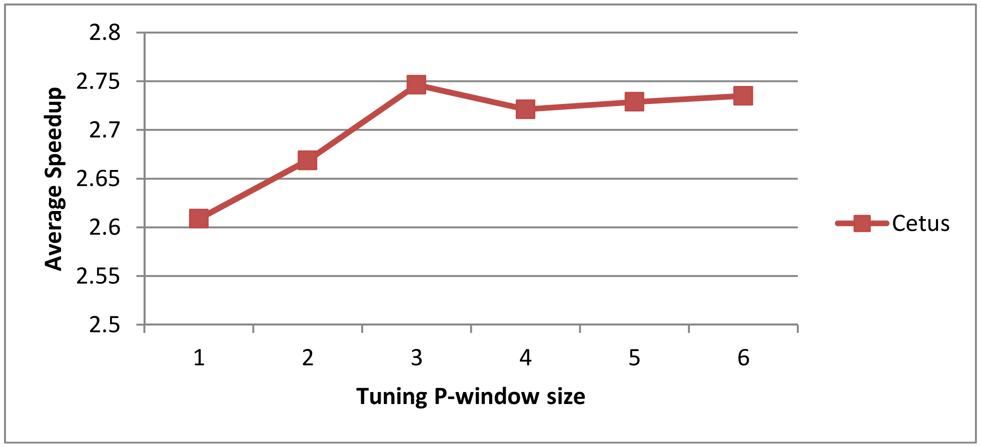


Section-level vs Program-level Tuning



Challenge:
Drastic increase in search space
=> Excessive tuning times

Idea:
Ignore interactions between optimizations of distant program sections



Conclusions

- 30+ years of research have delivered sophisticated tools
- Autoparallelization is not turned on by default, even in today's multicores
- Automatic performance tuning can ensure that performance never degrades
- Tuning can be made portable and section-level tuning makes a significant performance difference.