Hybrid Analysis
and its Application to Thread Level Parallelization

Lawrence Rauchwerger

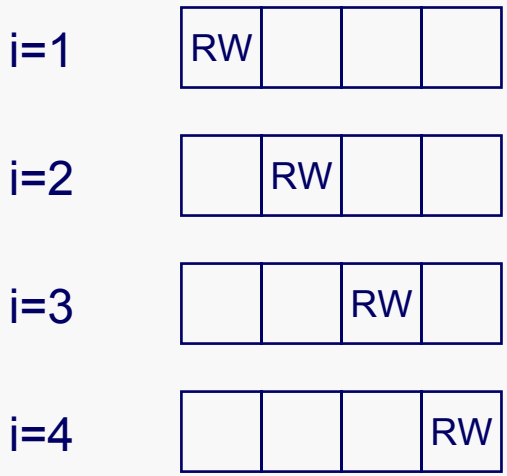
Thread (Loop) Level Parallelization

- Thread level Parallelization
  - Extracting parallel threads from a sequential program
- Manual
  - Very expensive for sequential legacy code: MPI - OpenMP
- Automatic
  - So far not very successful due to:
    - weak static analysis
    - Decisions depend on dynamic values (**input dependent**)

What is a parallel loop ?

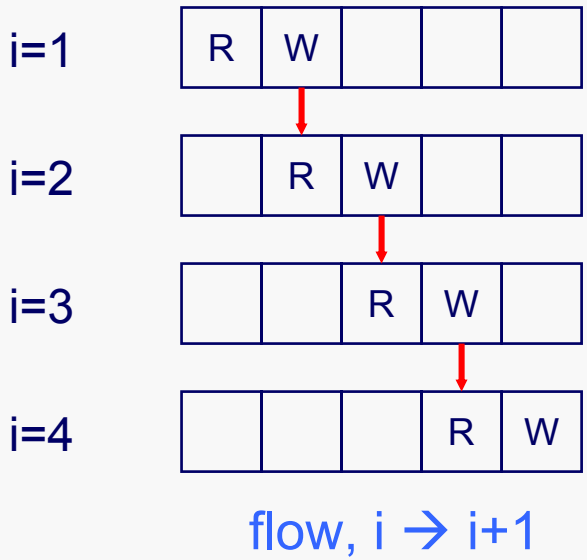
Parallelizable loop

```
DO i = 1, 100
  a[i] = a[i] + b[i]
ENDDO
```



Sequential loop

```
DO i = 1, 100
  a[i+1] = a[i] + b[i]
ENDDO
```



flow, i → i+1

Static Data Dependence Analysis: Linear Reference Patterns

Parasol

```
DO j = 1, 10
 a(j) = a(j+40)
ENDDO
```

No cross iteration dependences ⟺ No integer solutions:

$$\begin{cases} 1 \le j_w \le 10 \\ 1 \le j_r \le 10 \\ j_w \ne j_r \\ j_w = j_r + 40 \end{cases}$$

- Geometric view: Polytope model
  - Some convex body contains no integral points
- Simplified solutions: GCD Test, Banerjee Test etc
  - Potentially overly conservative
- General solution: Presburger formula decidability
  - Omega Test: Precise, potentially slow

Restricted to linear addressing and control: mostly small kernels

Static Data Dependence Analysis: Nonlinear Reference Patterns

Parasol

```
DO j = 1, 10
 IF (x(j)>0) THEN
  a(f(j)) = ...
 ENDIF
ENDDO
```

No known solution in general

Common cases:

- Indirect access (subscripted subscripts)
- Control dependence on data values
- Recurrence without closed form

– Linear Approximation
  - Maslov '94; Creusillet, Irigoin '96
  - Range Test (Blume, Eigenmann '94)
  - Monotonicity (Wu, Cohen, Padua '01)

– Symbolic Unknowns
  - Uninterpreted Function Symbols (Pugh, Wonnacott '95)
  - Fuzzy Array Dataflow (Barthou, Collard, Feautrier '95)

– User assistance
  - Uninterpreted Function Symbols (Pugh, Wonnacott '95)
  - SUIF Explorer (Liao '00)

Most nonlinear cases not solvable statically: need run time analysis

Alternative: Run-time Analysis

Parasol

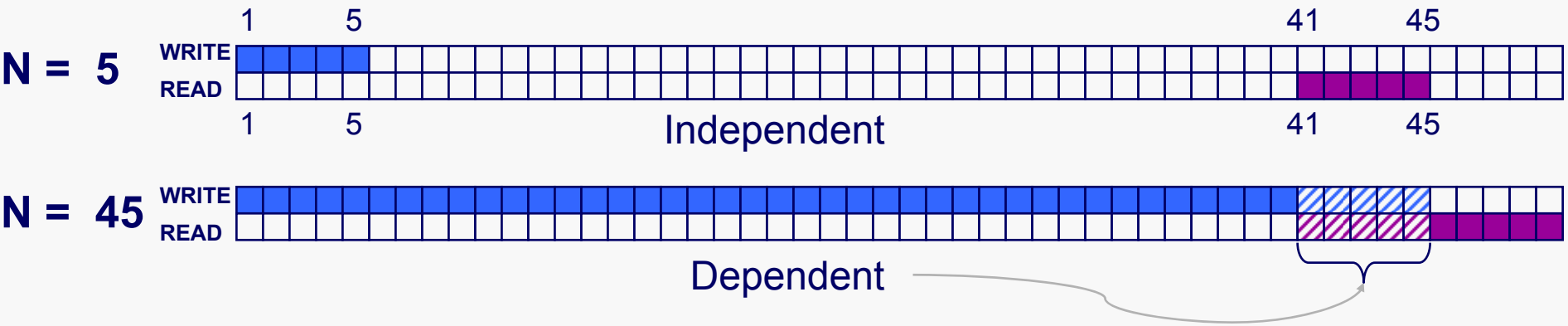
```
READ *, N
DO j=1,N
 a(j)=a(j+40)
ENDDO
```

Linear, very simple, but not decidable statically!

Solution: **LRPD Run-time Test** (Rauchwerger and Padua '95)

Instrument all relevant memory references

Analyze the resulting trace at run time



Accurate, but overhead proportional to the dynamic memory reference count

Minimum necessary information for parallelization: N ≤ 40

Compile-time vs. Run-time

Parasol

Compile Time

- PROs
  - No run-time overhead
- CONs: too conservative when
  - Input/computed values
    - Indirection
    - Control
  - Weak symbolic analysis
    - Complex recurrences
  - Impractical symbolic analysis
    - Combinatorial explosion

Run-time, reference by reference

- PROs
  - Always finds answers
- CONs
  - Run-time overhead proportional to dynamic memory reference count
  - Unnecessary work: Ignores partial compile-time results

Why Did We Fail ?

- Static Analysis cannot be sufficient (weak and/or input sensitive)
- Dynamic Analysis (misunderstood as only speculation) is not a substitute for Static Analysis (too costly)
- No program level representation suitable for representing statically un-analyzable code

Hybrid Analysis of memory Reference Patterns (Rus ICS'02, 07)

Parasol

	Compile-time Analysis	**Hybrid Analysis**	Run-time Analysis
STATIC	Symbolic analysis	Symbolic analysis Extract conditions	
DYNAMIC		Evaluate conditions	Full reference-by-reference analysis

Framework: Hybrid memory reference analysis

Application: Automatic parallelization

Hybrid Analysis

```
DO j=1,N
 a(j)=a(j+40)
ENDDO
```

Parasol

4.a) If we can prove $10 \le N \le 30$, generate parallel.loop

4.b) If **N** is unknown, Extract run-time test.

N≤40

Compile Time

Run Time

Parallel Loop

```
DO PARALLEL j=1,N
 a(j)=a(j+40)
ENDDO
```

No run-time tests performed if not necessary!

Run-time Test

```
IF (N≤40) THEN
 DO PARALLEL j=1,N
  a(j)=a(j+40)
 ENDDO
ELSE
 DO j=1,N
  a(j)=a(j+40)
 ENDDO
ENDIF
```

Parallel Loop

Sequential Loop

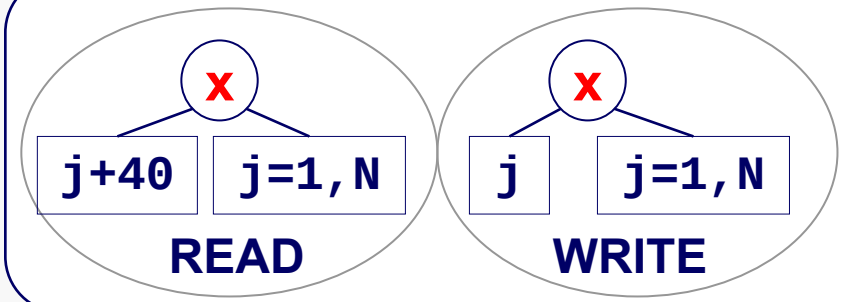
Hybrid Analysis

Compile-time Phase

```
DO j=1,N
 a(j)=a(j+40)
ENDDO
```

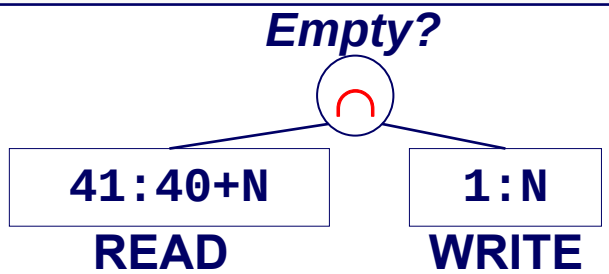
Under what conditions can the loop be executed in parallel?

Parasol



1. Collect and classify memory references.

2. Aggregate them symbolically



3. Formulate independence test.

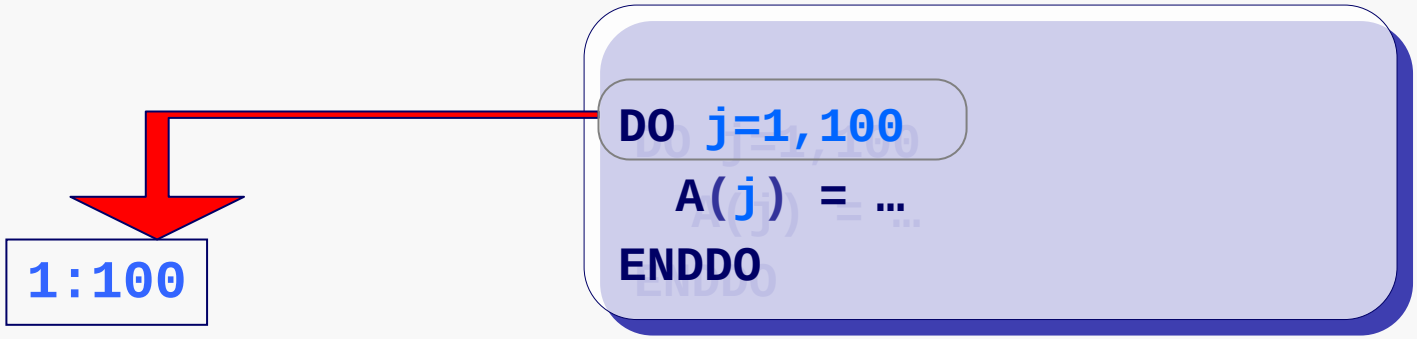
4.a) If we can prove $10 \le \mathbf{N} \le \mathbf{30}$, Declare loop parallel.

4.b) If **N** is unknown, Extract run-time test.

N≤40

Aggregation of Linear References Across an Iteration Space

WF pattern for **A**:

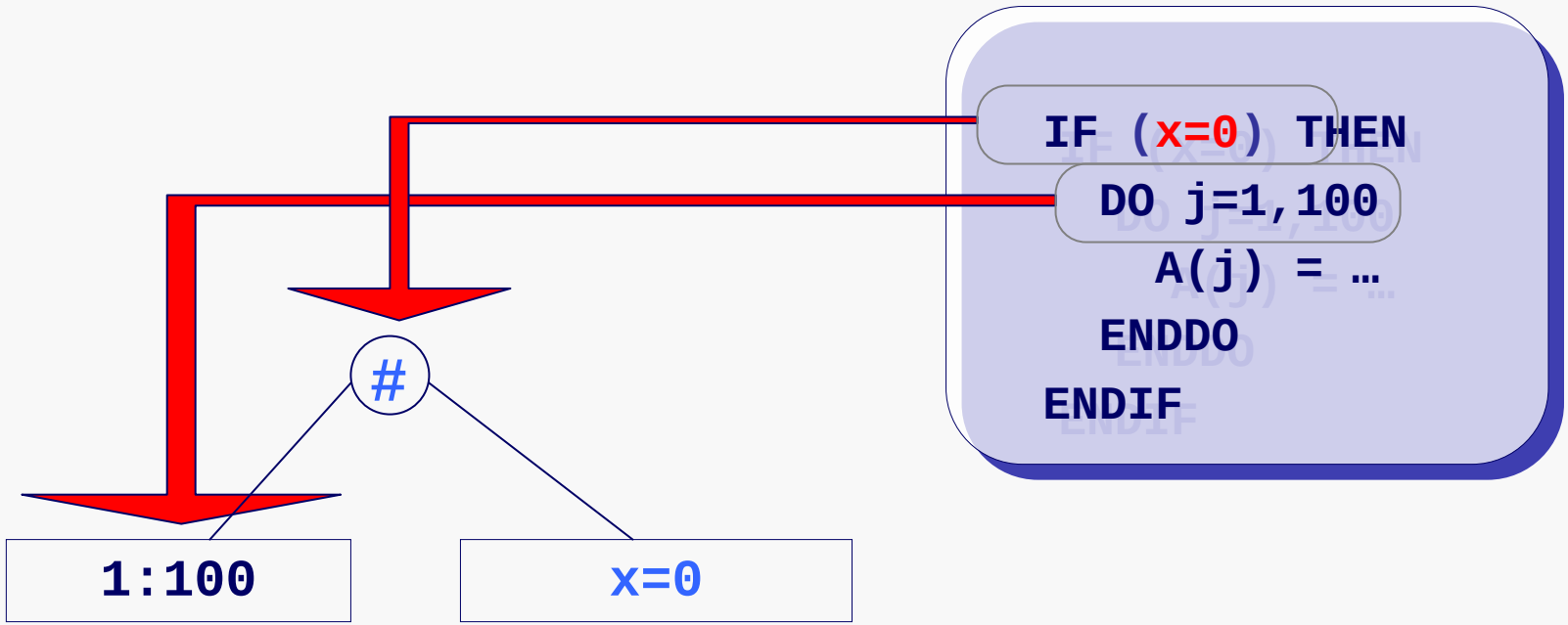


LMAD = Linear Memory Access Descriptor (Hoeflinger ’98)
Multidimensional, strided intervals

Gate Operator

Postpone Analysis Failure

The truth value of (`x=0`) is not known

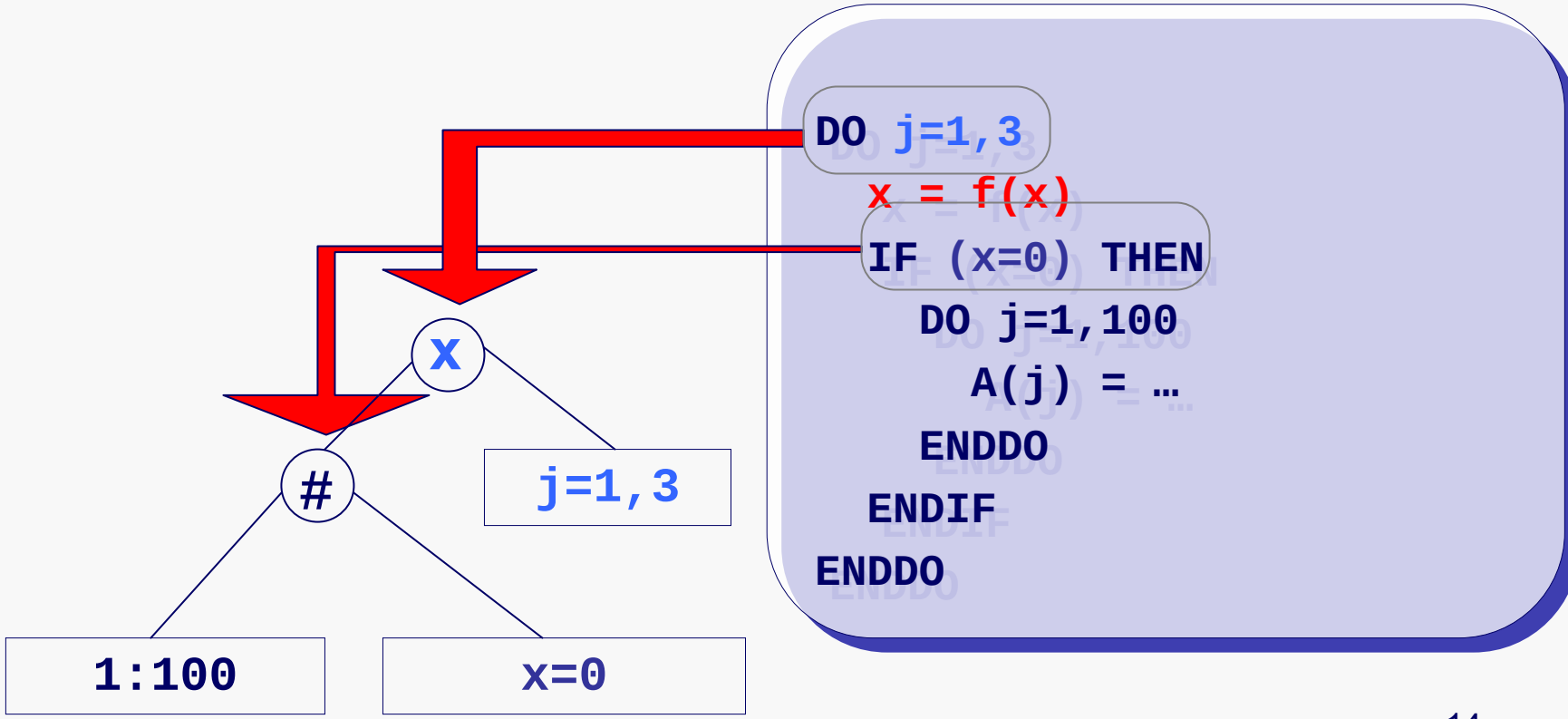


Recurrence Operator

Postpone Static Analysis Failure due to Nonlinear Reference Pattern

Parasol

Recurrence with no closed form on **x**



Uniform Set of References (USR)

Parasol

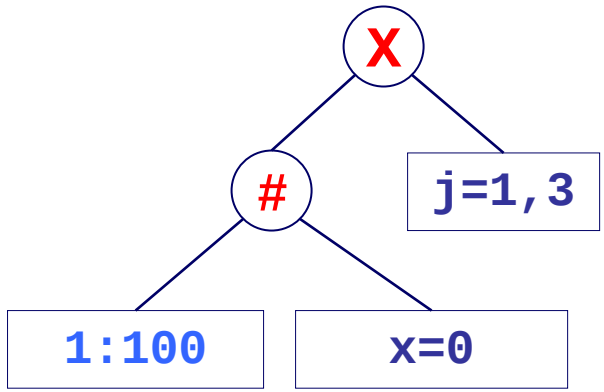
Closed under composition at program level

T = { **LMAD**, ∩, ∪, –, **(**, **)**, #, ⊗, **ϴ**, **Gate**, **Recurrence**, **Call Site**}

N = { USR }

S = USR

P = { USR → **LMAD** | **(**USR **)**
USR → USR ∩ USR
USR → USR ∪ USR
USR → USR – USR
USR → **Gate #** USR
USR → **Recurrence** ⊗ USR
USR → **Call Site ϴ** USR }

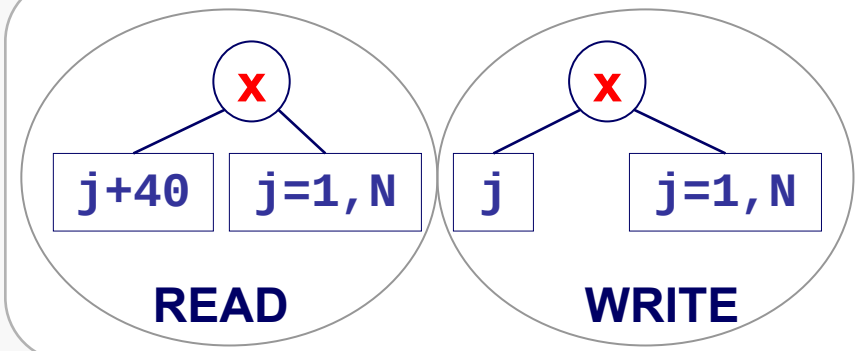


LMAD = Start + [Stride$_1$:Span$_1$, Stride$_2$:Span$_2$, ...]

```
READ *, N
DO j=1,N
 a(j)=a(j+40)
ENDDO
```

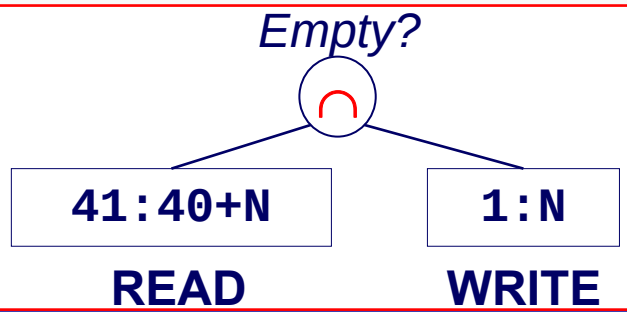
Are there any cross-iteration dependences?

Parasol



1. Collect references.

2. Aggregate them symbolically & Classify Reference Type



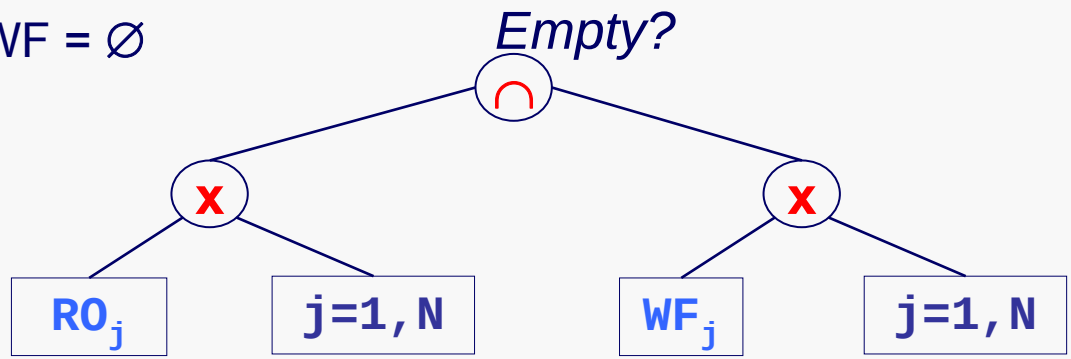
3. Formulate independence test.

4. Extract lowest-cost runtime test.

Data Dependences

Parasol

- Given:
  - Loop expression: **`j = 1,N`**
  - Per-iteration aggregated descriptors $\mathbf{RO_j}$, $\mathbf{WF_j}$, $\mathbf{RW_j}$
- Solve equation RO ∩ WF = ∅



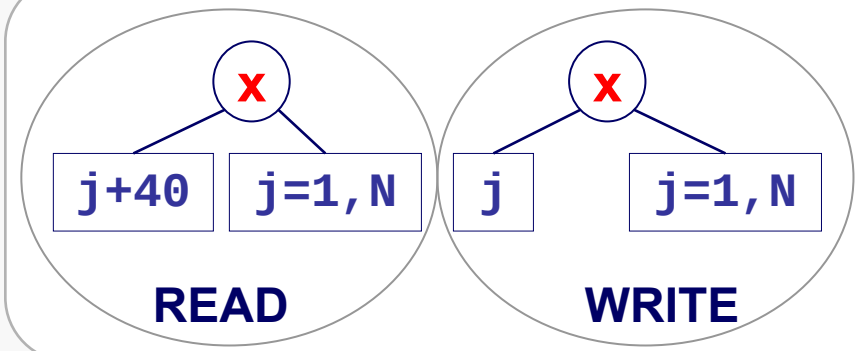
- At compile-time:
  - RO ∩ WF evaluates to ∅ ⇒ independent
  - RO ∩ WF evaluates to a set that is not empty ⇒ dependent
  - All other cases: run-time dependence test

Similar equations for privatization and reduction recognition

```
READ *, N
DO j=1,N
 a(j)=a(j+40)
ENDDO
```

Are there any cross-iteration dependences?

Parasol

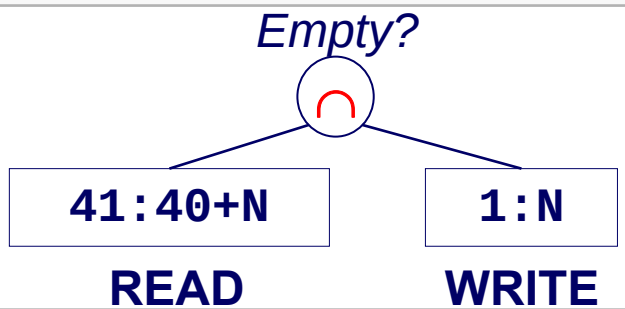


1. Collect references.

41:40+N — READ

1:N — WRITE

2. Aggregate them symbolically & Classify Reference Type



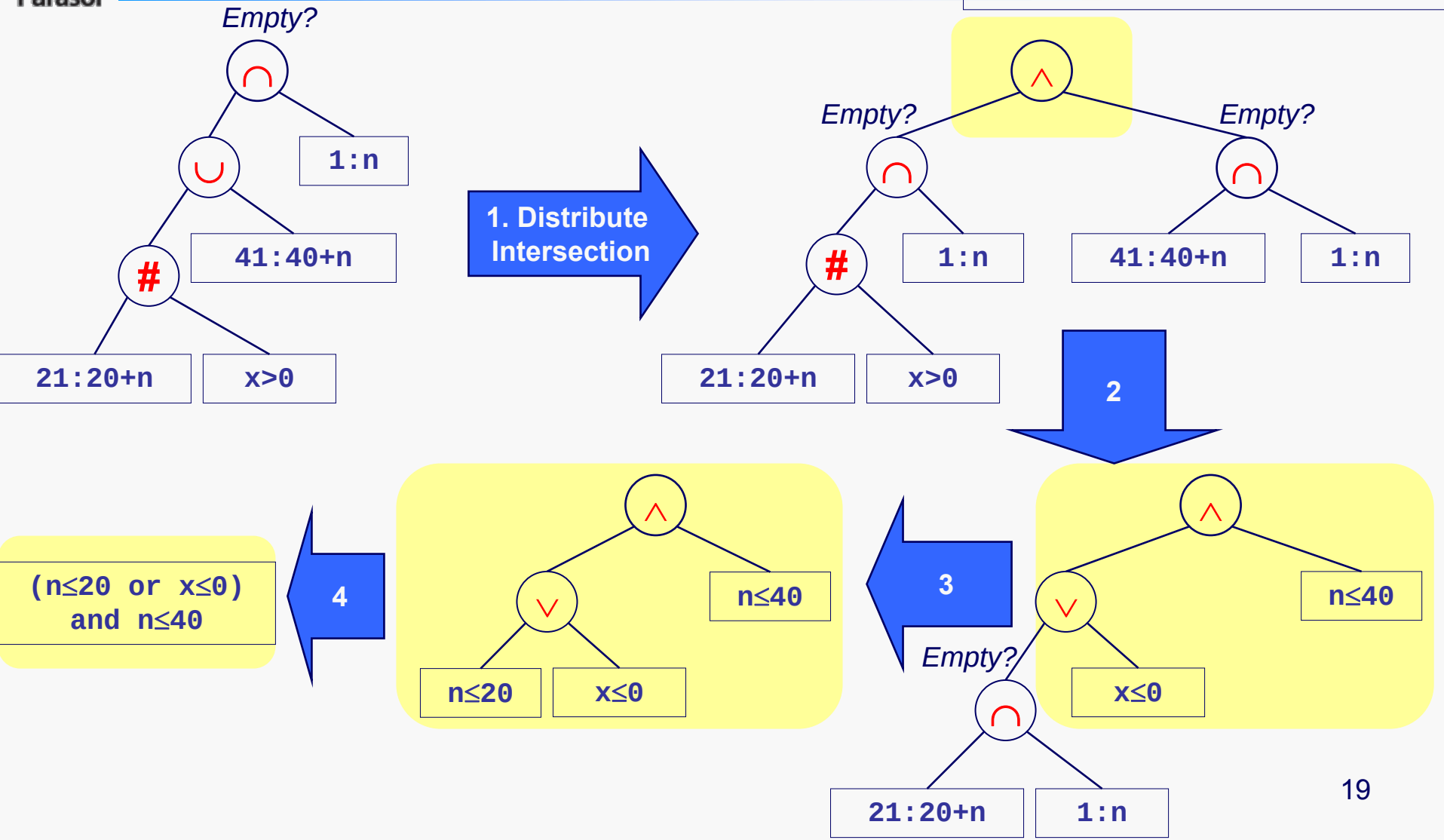
3. Formulate independence test.

N≤40

4. Extract lowest-cost runtime test.

Algorithm:
Recursive Descent

```
DO j = 1, n
  A(j) = A(j+40)
  IF (x>0) THEN
    A(j) = A(j) + A(j+20)
  ENDIF
ENDDO
```



Novel Static/Dynamic Interface: Predicate DAG

Parasol

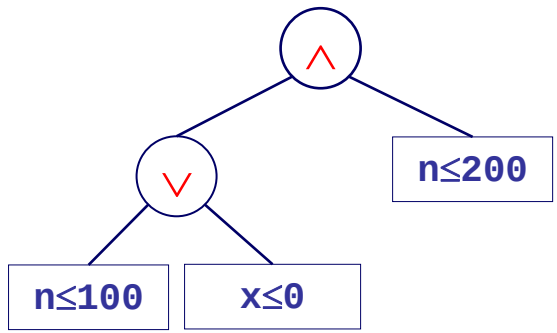
Represents any dynamic condition for loop parallelization

T = { Logical Expression, $\wedge$, $\vee$, $\otimes_{\wedge}$, $\otimes_{\vee}$, Θ, Recurrence, Call Site, Library Routine}

N = { PDAG }

S = PDAG

P = { PDAG $\rightarrow$ Logical Expression
PDAG $\rightarrow$ PDAG $\wedge$ PDAG
PDAG $\rightarrow$ PDAG $\vee$ PDAG
PDAG $\rightarrow$ PDAG $\otimes_{\wedge}$ Recurrence
PDAG $\rightarrow$ PDAG $\otimes_{\vee}$ Recurrence
PDAG $\rightarrow$ PDAG Θ Call Site
PDAG $\rightarrow$ Library Routine }



- Expressive: arbitrary dependence questions
- Inexpensive: evaluates quickly at run-time

PDAG Extraction

Parasol

Input: USR equation $D = \varnothing$

Output: PDAG P

Such that:

$$P \iff D = \varnothing$$

Optimistic: Sufficient predicate: $P \Rightarrow D = \varnothing$

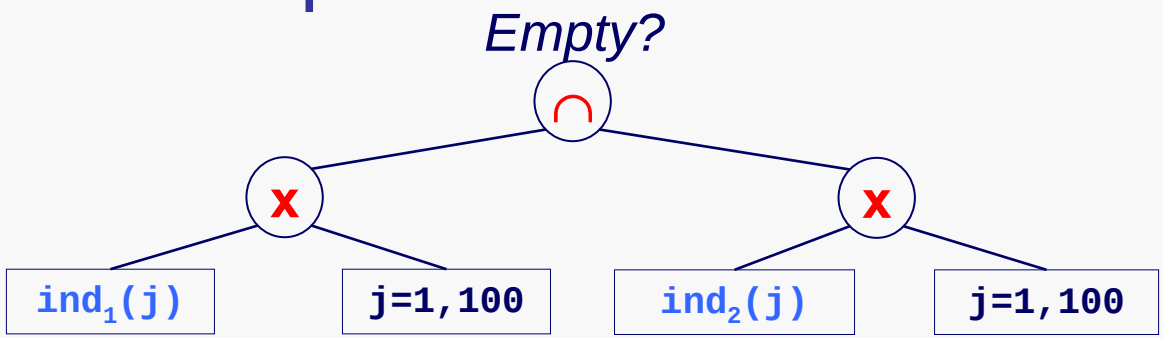
Pessimistic: Necessary predicate: $D = \varnothing \Rightarrow P$, or $\overline{P} \Rightarrow D \neq \varnothing$

```
DO j=1,100
  A(ind1(j)) = A(ind2(j))
ENDDO
```

Fallback

Parasol

Not all equations can be reversed

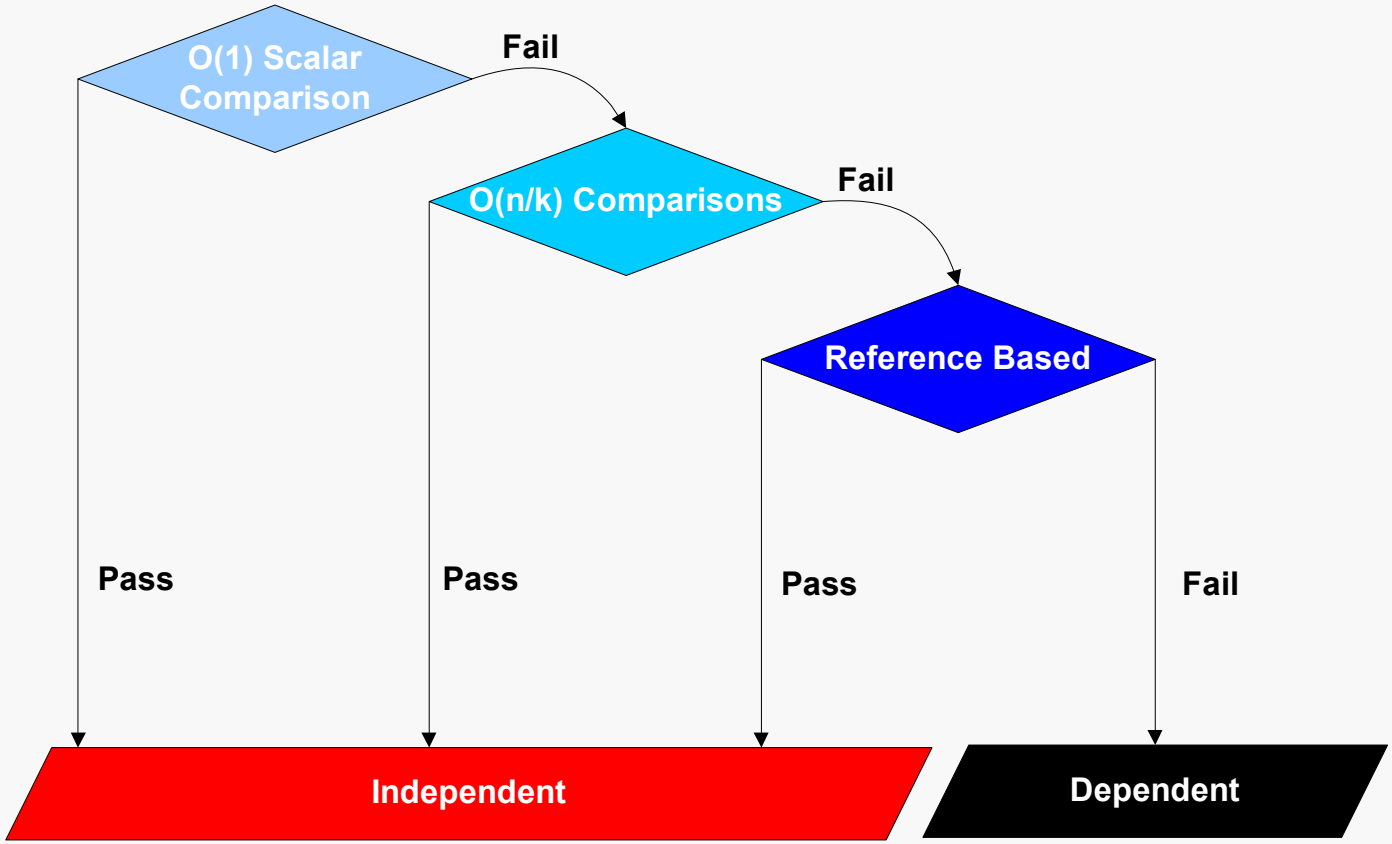


Fallback Solutions

- Trace based
- Speculation

Possibly Aggregated

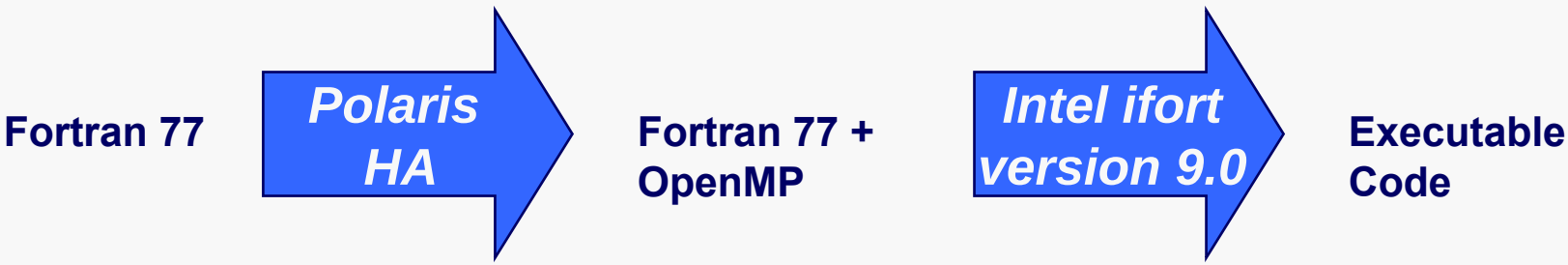
Hybrid Analysis: Run Time



Evaluation Methodology

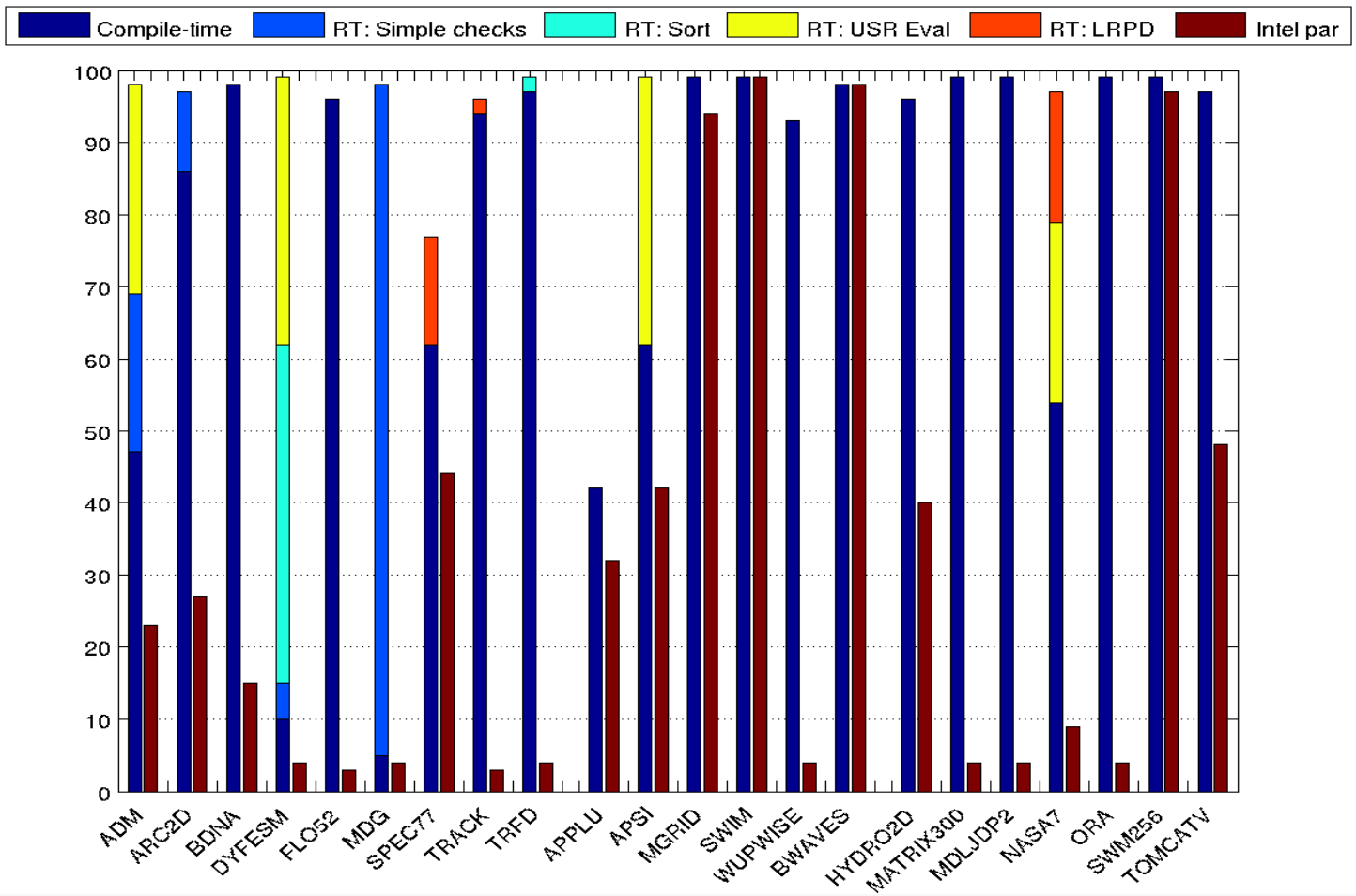
Parasol

- Automatic parallelization using Hybrid Analysis in Polaris
  - Analyses/Transformations: Dependence analysis, privatization, reduction, pushback
  - Candidate loop selection: Profiling
  - No scheduling or memory locality optimization
  - Simple dynamic mechanism to rule out very small loops

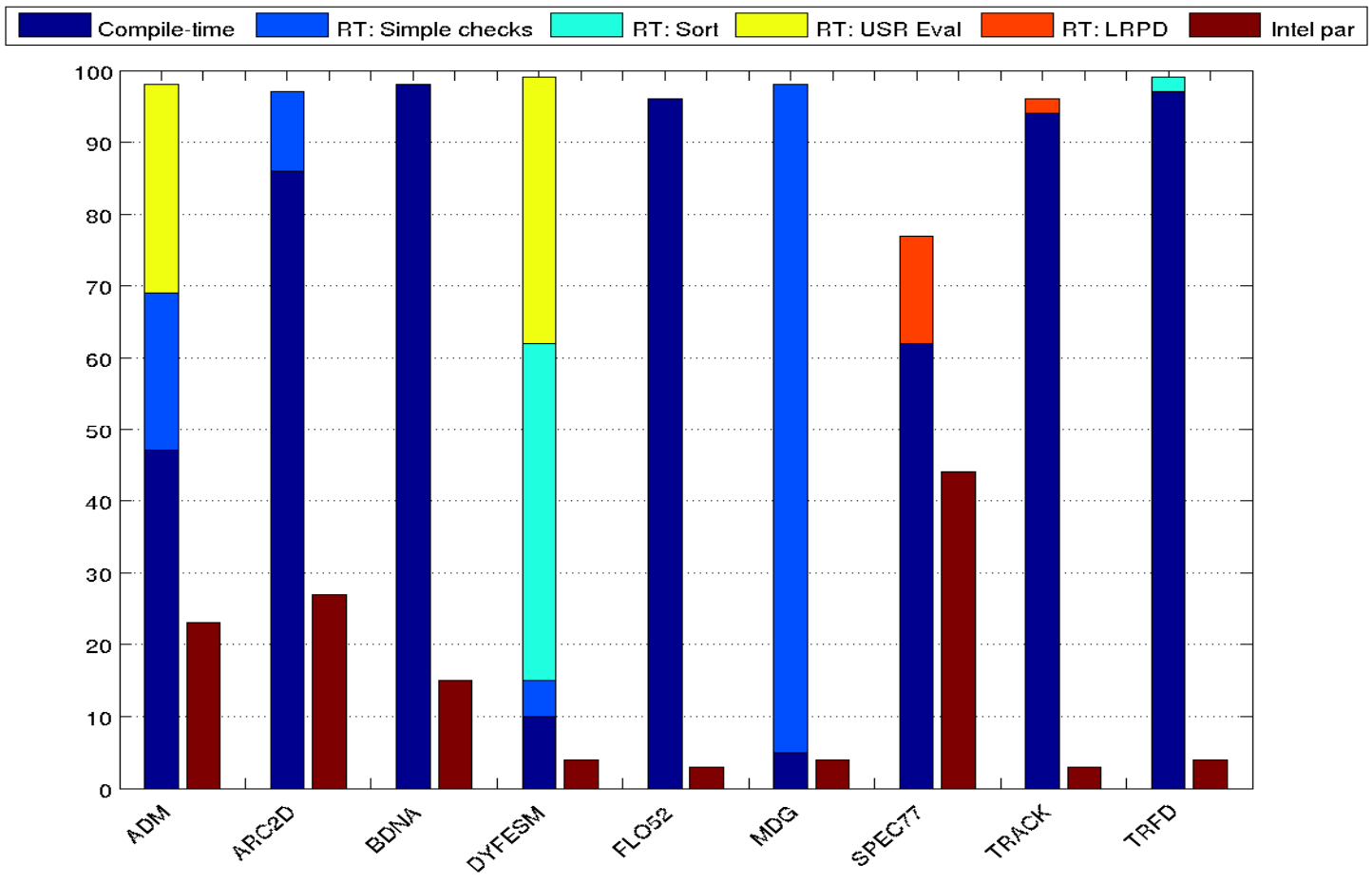


- Experiment Setup
  - SPEC 2000 - 2006, PERFECT, Other SPEC
  - -Dual Core CoreDuo Intel, Lenovo X60s
  - Dual Core AMD, quad socket Sun

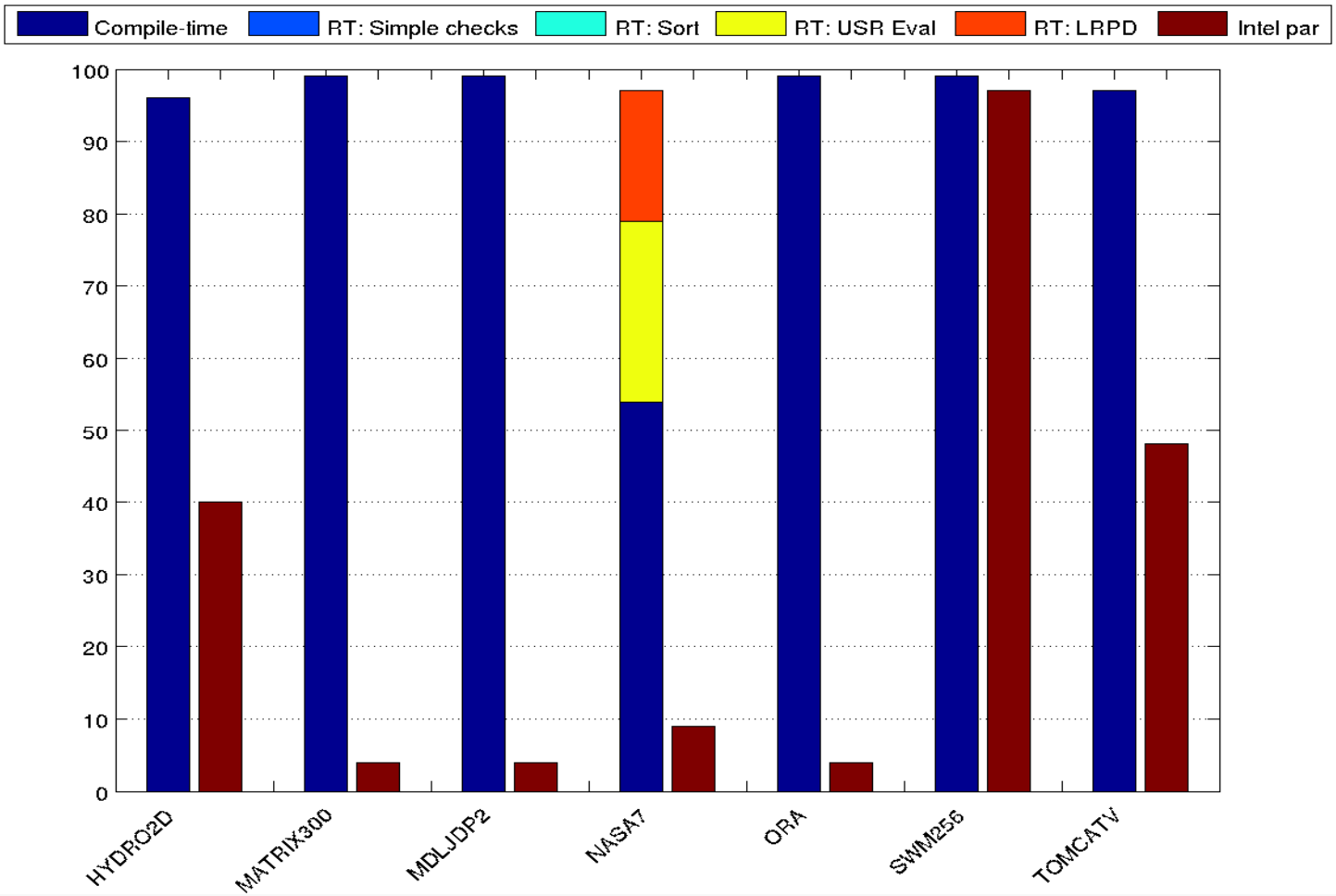
Polaris-HA vs. Ifort Coverage: PERFECT/SPEC89/92



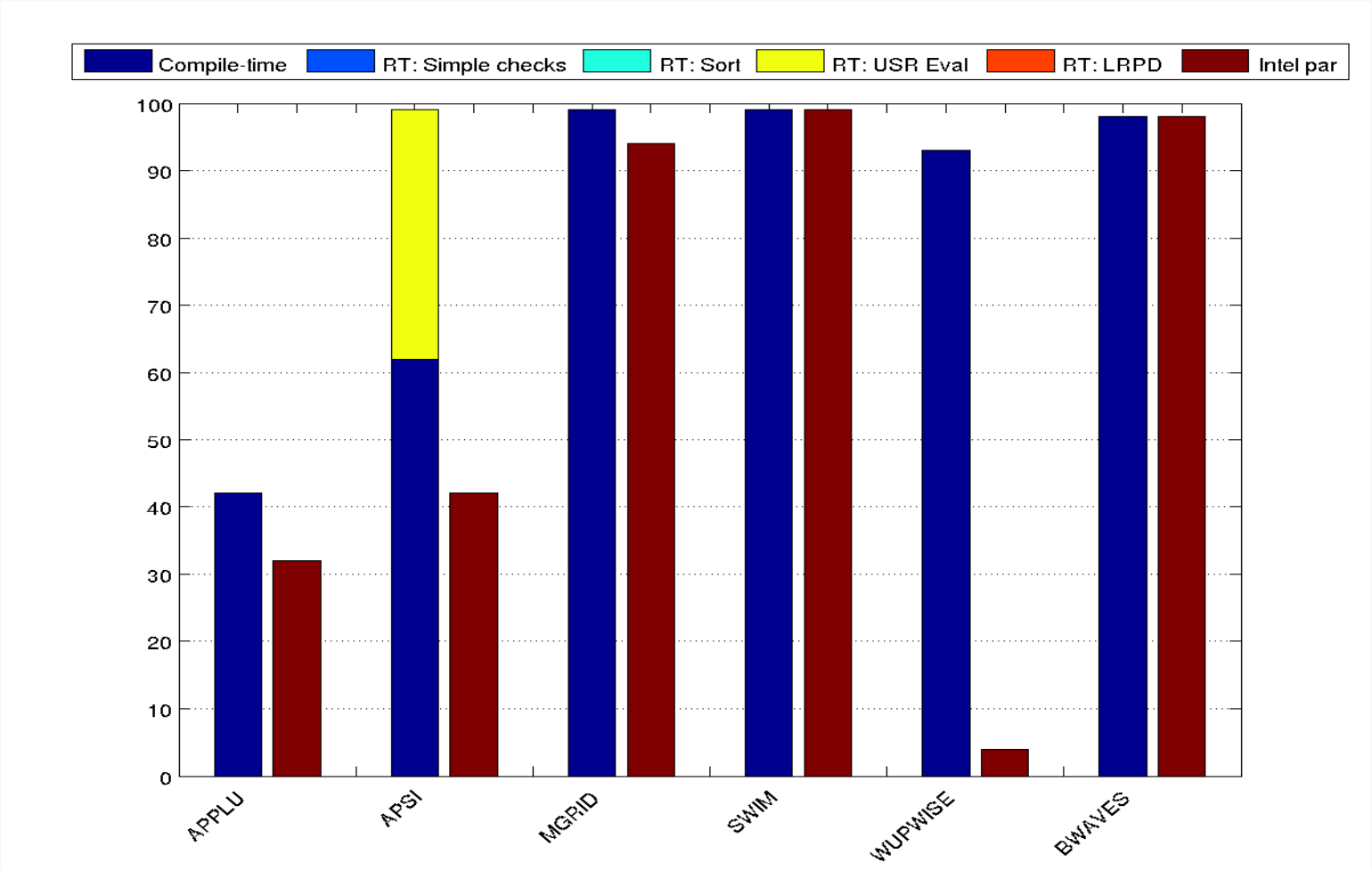
Polaris-HA vs. Ifort Coverage: PERFECT Benchmarks



Polaris-HA vs. Ifort Coverage: SPEC 89/92

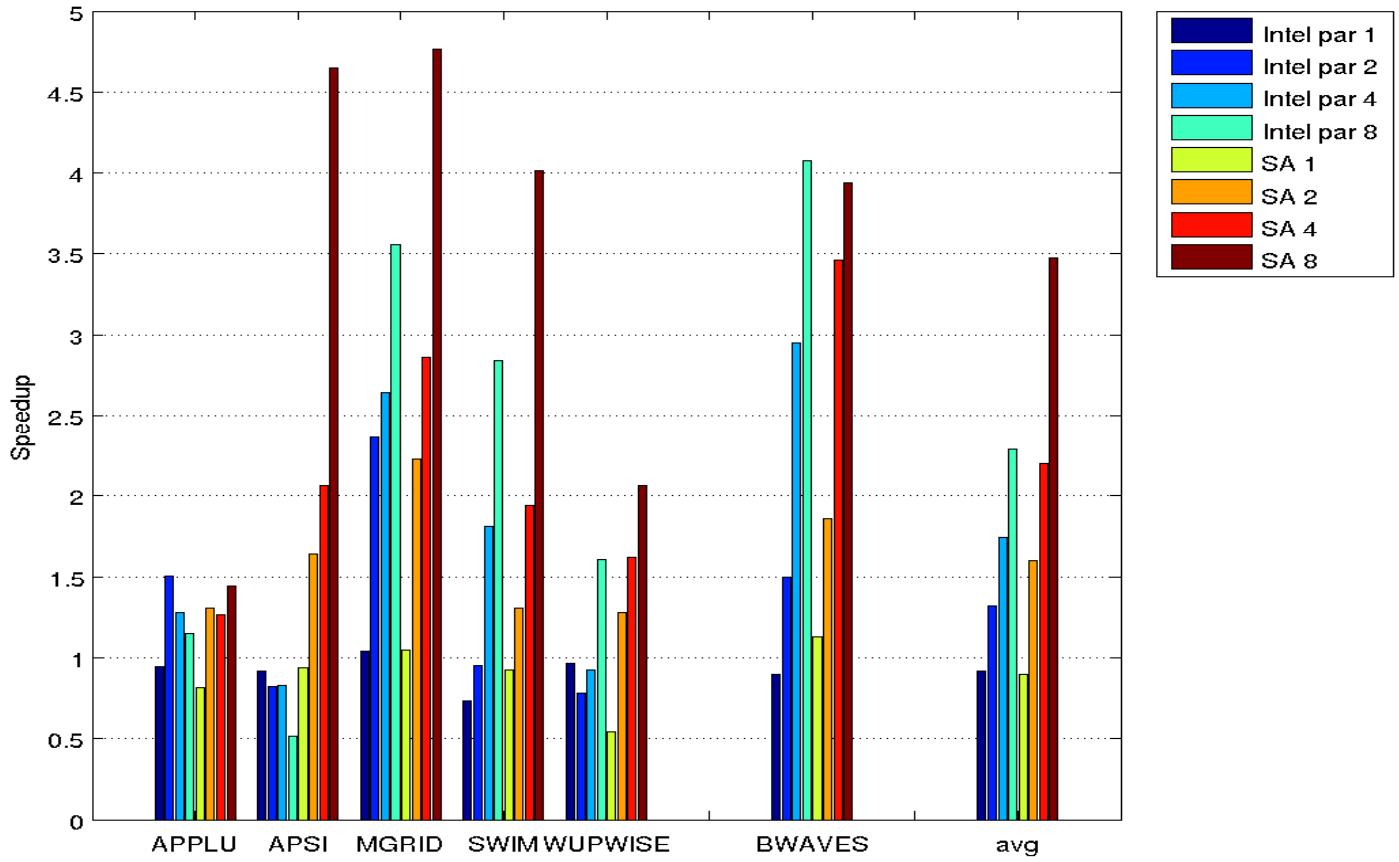


Polaris-HA vs. Ifort Coverage: SPEC2000/06

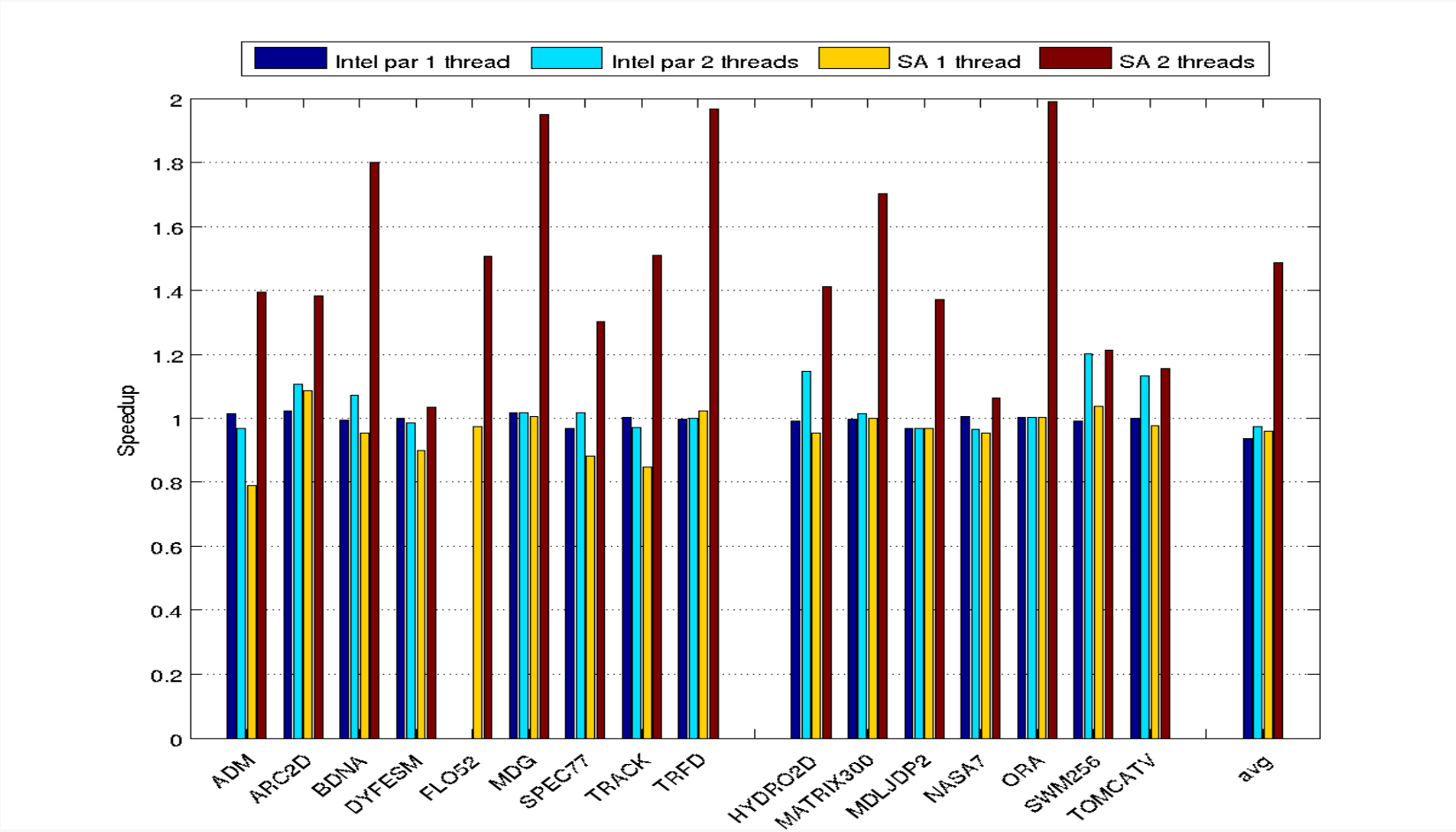


Speedups SPEC2000/06:

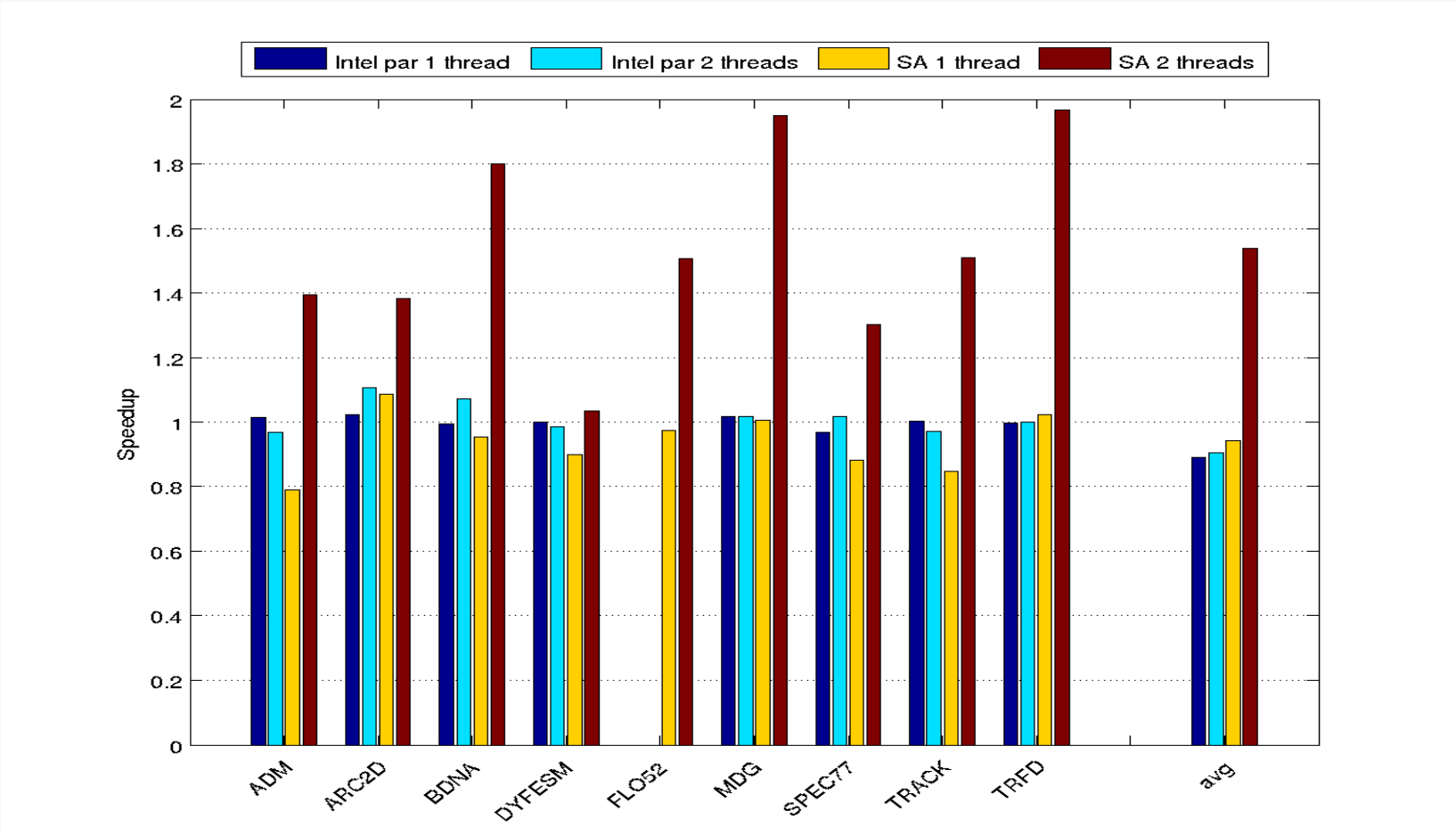
AMD dual core/quad socket



Speedups Perfect/Spec89/92: Intel Core Duo

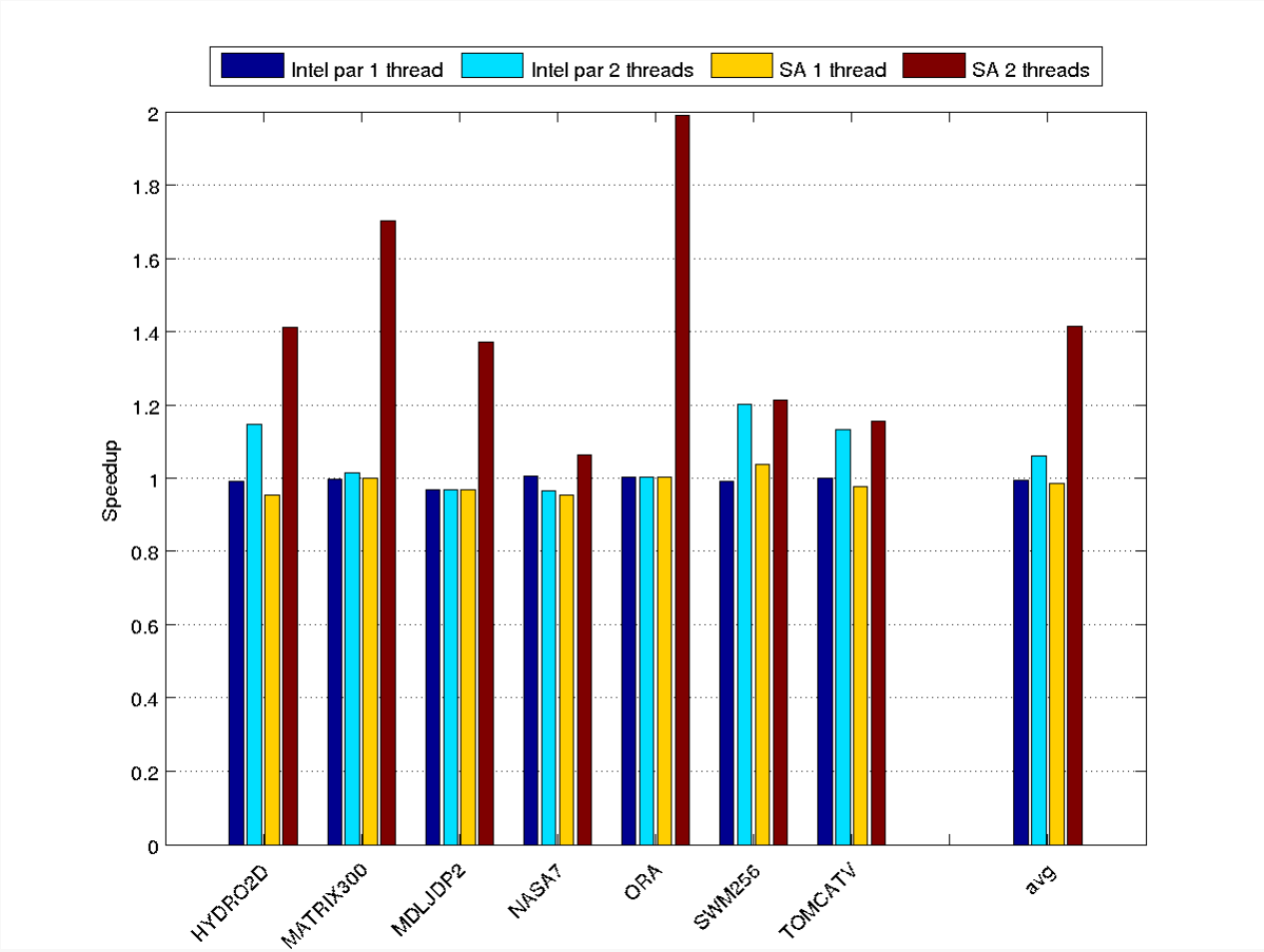


Speedups PERFECT: Intel CoreDuo



Speedups SPEC 89/92: Intel Core Duo

Parasol



Conclusions: HA + Autopar

Parasol

- Hybrid memory reference analysis is a general framework for optimization
- Representation is crucial
  - -- USR
    - Closed-form representation that tolerates analysis failure
  - – PDAG:
    - Input sensitivity of optimization decisions
    - Continuum of compile-time to run-time solutions
- Efficient automatic parallelization
  - – Good Speedups on FP benchmark applications

http://parasol.tamu.edu/compilers/ha