Heuristics for Profile-driven Method-level Speculative Parallelization

John Whaley and Christos Kozyrakis
Stanford University

June 15, 2005

Speculative Multithreading

- Speculatively parallelize an application
  - Uses speculation to overcome ambiguous dependencies
  - Uses hardware support to recover from misspeculation
  - Promising technique for automatically extracting parallelism from programs
- Problem: Where to put the threads?

Method-Level Speculation

- Idea: Use method boundaries as speculative threads
  - Computation is naturally partitioned into methods
  - Execution often independent
  - Well-defined interface
- Extract parallelism from irregular, non-numerical applications

Method-Level Speculation Example

```
main()
{
  work_A;

  foo();

  work_C; // reads *q
}
```

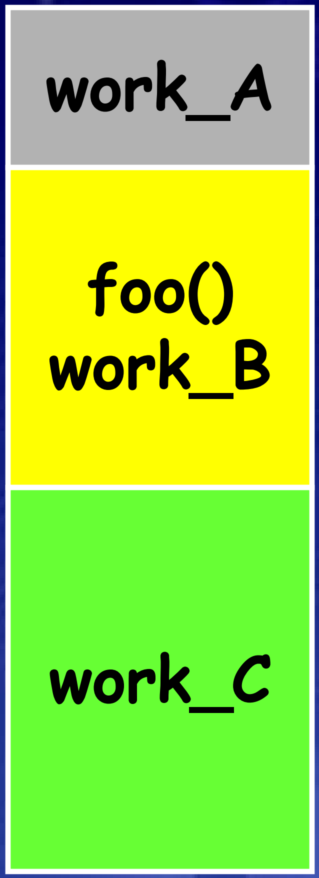
```
foo()
{
  work_B; // writes *p
}
```

Method-Level Speculation Example

```
main()
{
  work_A;
  foo() {
    work_B; // writes *p
  }
  work_C; // reads *q
}
```

Method-Level Speculation Example

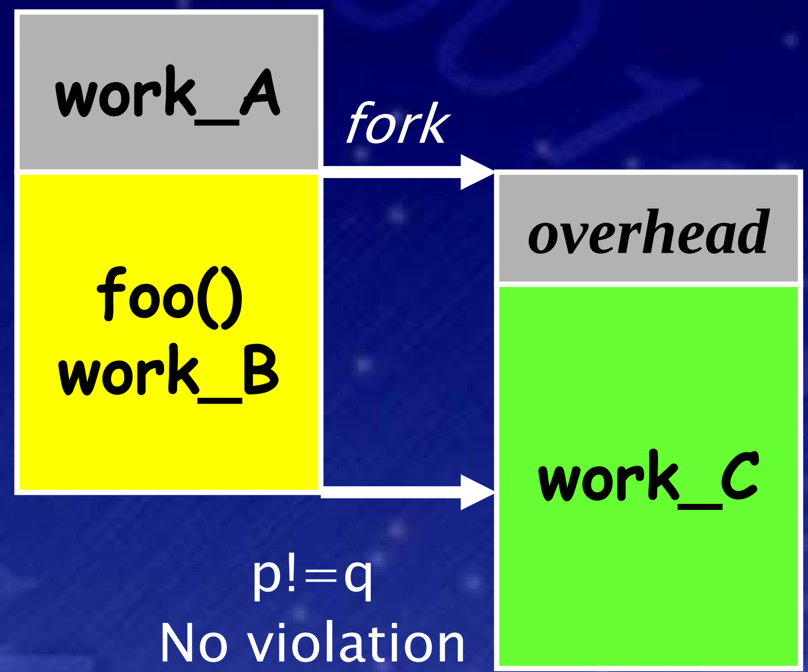
```
main()
{
  work_A;
  foo() {
    work_B; // writes *p
  }
  work_C; // reads *q
}
```



Sequential execution

Method-Level Speculation Example

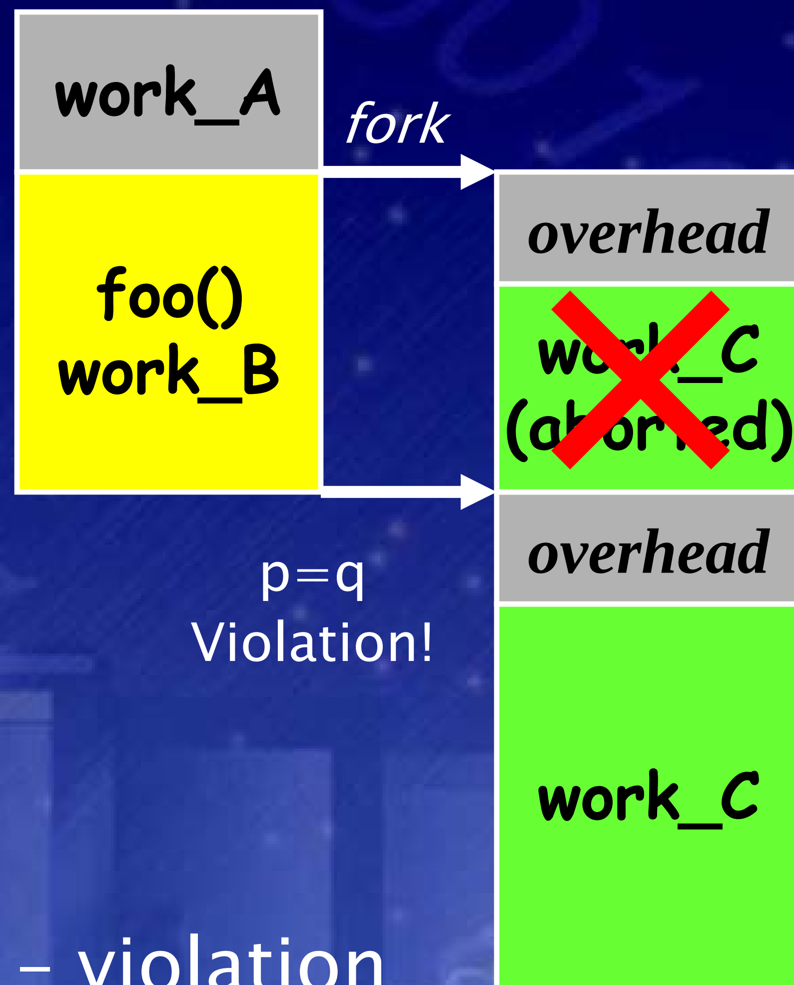
```
main()
{
  work_A;
  foo() {
    work_B; // writes *p
  }
  work_C; // reads *q
}
```



TLS execution – no violation

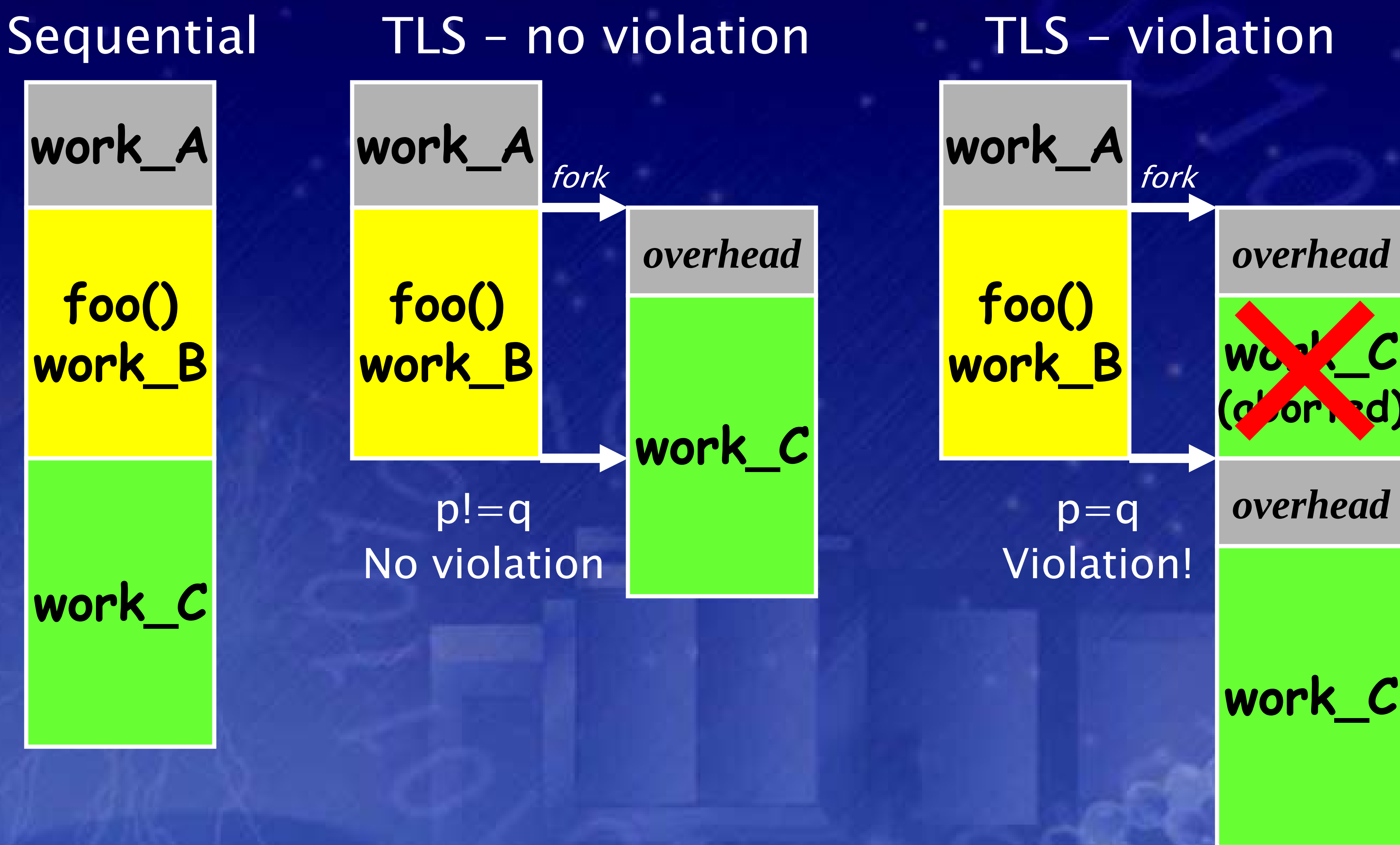
Method-Level Speculation Example

```
main()
{
  work_A;
  foo() {
    work_B; // writes *p
  }
  work_C; // reads *q
}
```

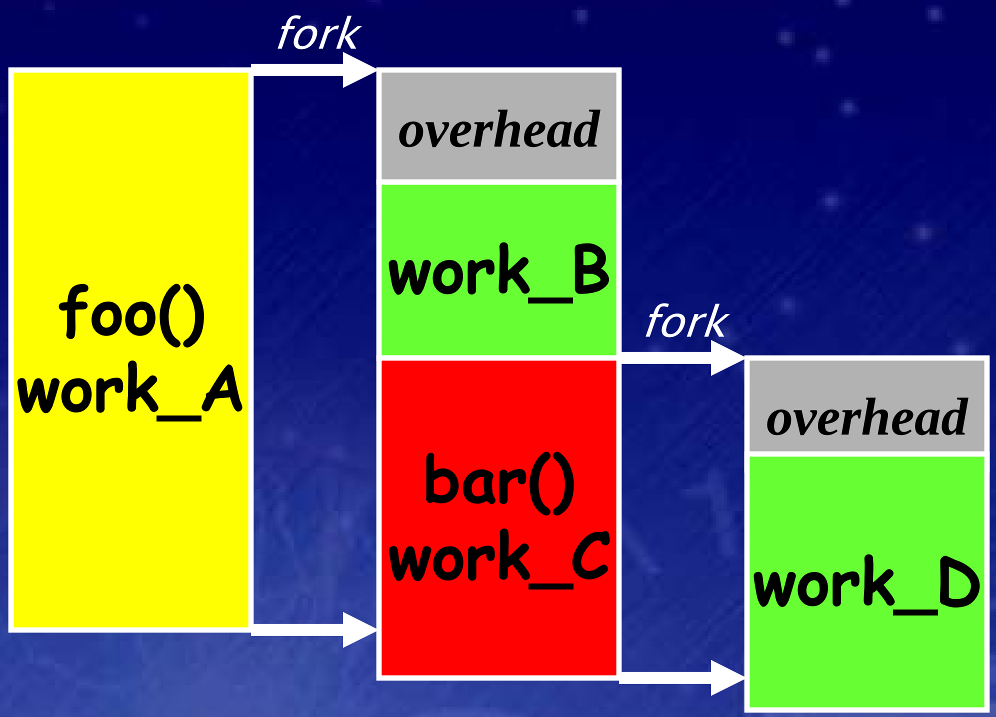


TLS execution – violation

Method-Level Speculation Example



Nested Speculation




```
main()
{
   foo() {
     work_A;
   }
   work_B;
   bar() {
     work_C;
   }
   work_D;
}
```

Sequences of method calls can cause nested speculation.

This Talk: Choosing Speculation Points

- Which methods to speculate?
  - Low chance of violation
  - Not too short, not too long
  - Not too many stores
- Idea: Use profile data to choose good speculation points
  - Used for profile-driven and dynamic compiler
  - Should be low-cost but accurate
- We evaluated 7 different heuristics
  - ~80% effective compared to perfect oracle

Difficulties in Method-Level Speculation

- Method invocations can have varying execution times
  - Too short: Doesn't overcome speculation overhead
  - Too long: More likely to violate or overflow, prevents other threads from retiring
- Return values
  - Mispredicted return value causes violation

Classes of Heuristics

- Simple Heuristics
  - Use only simple information, such as method runtime
- Single-Pass Heuristics
  - More advanced information, such as sequence of store addresses
  - Single pass through profile data
- Multi-Pass Heuristics
  - Multiple passes through profile data

Classes of Heuristics

- Simple Heuristics
  - Use only simple information, such as method runtime
- Single-Pass Heuristics
  - More advanced information, such as sequence of store addresses
  - Single pass through profile data
- Multi-Pass Heuristics
  - Multiple passes through profile data

Runtime Heuristic (SI-RT)

- Speculate on all methods with:
  - MIN < runtime < MAX
- Idea: Should be long enough to amortize overhead, but not long enough to violate
- Data required:
  - Average runtime of each method

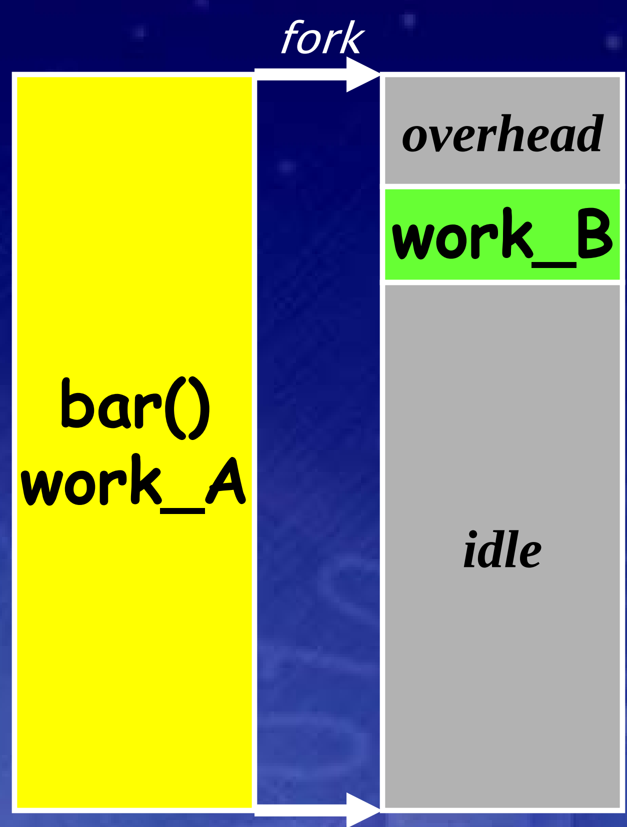
Store Heuristic (SI-SC)

- Speculate on all methods with:
  - dynamic # of stores < MAX
- Idea: Stores cause violations, so speculate on methods with few stores
- Data required:
  - Average dynamic store count of each method

Classes of Heuristics

- Simple Heuristics
  - Use only simple information, such as method runtime
- Single-Pass Heuristics
  - More advanced information, such as sequence of store addresses
  - Single pass through profile data
- Multi-Pass Heuristics
  - Multiple passes through profile data

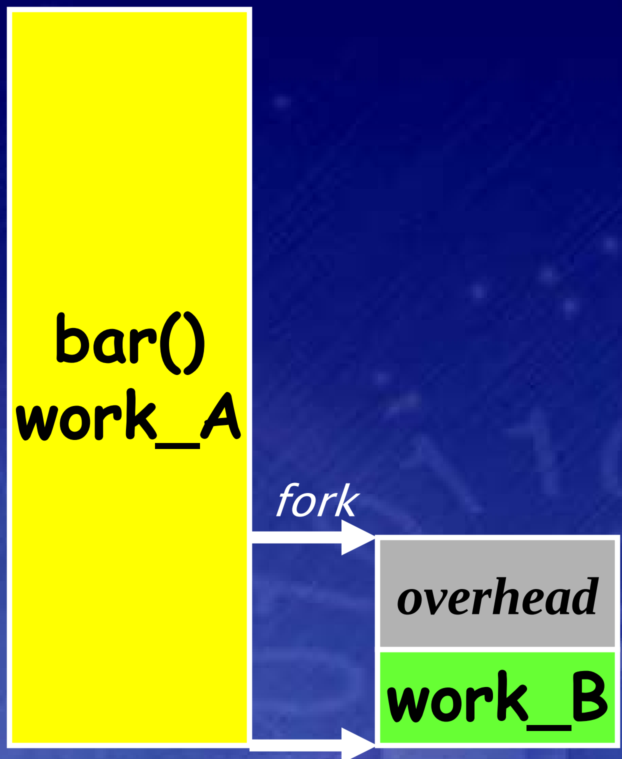
Stalled Threads




```
foo()
{
  bar() {
    work_A;
  }
  work_B;
}
```

Speculative threads may stall while waiting to become main thread.

Fork at intermediate points




```
foo()
{
  bar() {
    work_A;
  }
  work_B;
}
```

Fork at an intermediate point within a method to avoid violations and stalling

Best Speedup Heuristic (SP-SU)

- Speculate on methods with:
  - predicted speedup > THRES
- Calculate predicted speedup by:

$$\frac{\text{expected sequential run time}}{\text{expected parallel run time}}$$

- Scan store stream backwards to find fork point
  - Choose fork point to avoid violations and stalling

Most Cycles Saved Heuristic (SP-CS)

- Speculate on methods with:
  - predicted cycle savings > THRES
- Calculate predicted cycle savings by:

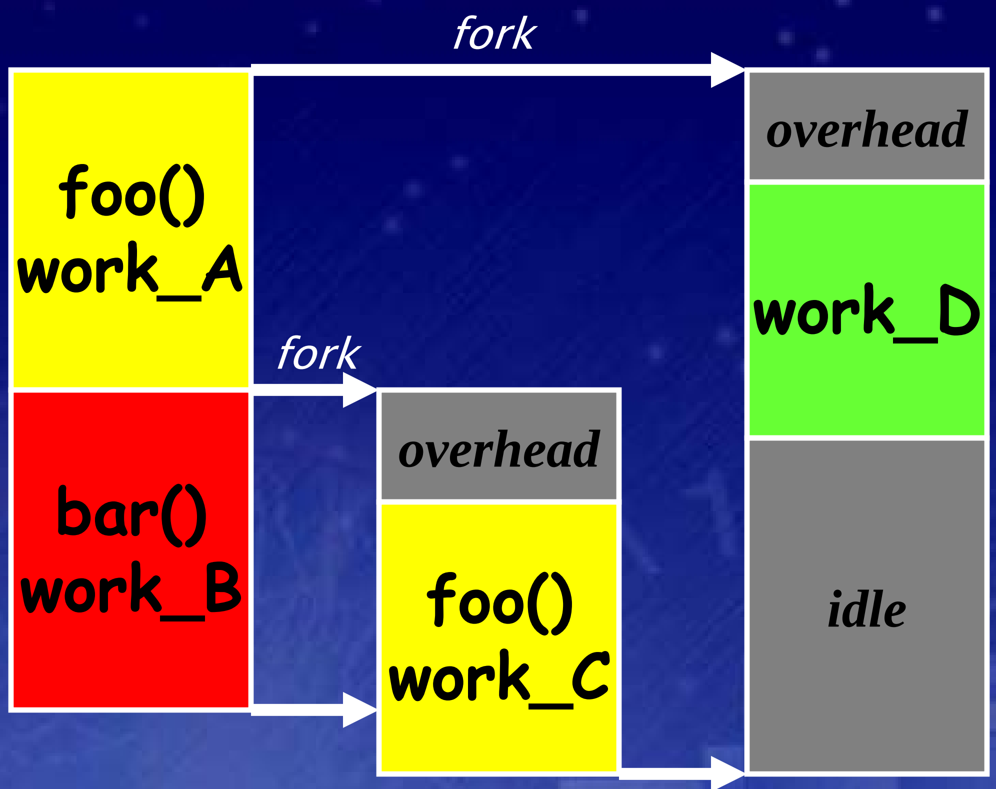
  sequential cycle count – parallel cycle count

- Place fork point such that:
  - predicted probability of violation < RATIO
- Uses same information as SP-SU

Classes of Heuristics

- Simple Heuristics
  - Use only simple information, such as method runtime
- Single-Pass Heuristics
  - More advanced information, such as sequence of store addresses
  - Single pass through profile data
- Multi-Pass Heuristics
  - Multiple passes through profile data

Nested Speculation




```
main()
{
   foo() {
      work_A;
      bar() {
         work_B;
      }
      work_C;
   }
   work_D;
}
```

Effectiveness of speculation choice depends on choices for caller methods!

Best Speedup Heuristic with Parent Info (MP-SU)

- Iterative algorithm:
  - Choose speculation with best speedup
  - Readjust all callee methods to account for speculation in caller
  - Repeat until best speedup < THRES
- Max # of iterations: depth of call graph

Most Cycles Saved Heuristic with Parent Info (MP-CS)

- Iterative algorithm:
  1. Choose speculation with most cycles saved and predicted violations < RATIO
  2. Readjust all callee methods to account for speculation in caller
  3. Repeat until most cycles saved < THRES
- Multi-pass version of SP-CS

Most Cycles Saved Heuristic with No Nesting (MP-CSNN)

- Iterative algorithm:
  - Choose speculation with most cycles saved and predicted violations < RATIO.
  - *Eliminate* all callee methods from consideration.
  - Repeat until most cycles saved < THRES.
- Disallows nested speculation to avoid double-counting the benefits
- Faster to compute than MP-CS

Experimental Results

Trace-Driven Simulation

- How to find the optimal parameters (THRES, RATIO, etc.) ?
- Parameter sweeps
  - For each benchmark
    - For each heuristic
      - Multiple parameters for each heuristic
- For cycle-accurate simulation: >100 CPU years?!
- Alternative: *trace-driven simulation*

Trace-Driven Simulation

1. Collect trace on Pentium III (3-way out-of-order CPU, 32K L1, 256K L2)
   - Record all memory accesses, enter/exit method events, etc.
2. Recalibrate to remove instrumentation overhead
3. Simulate trace on 4-way CMP hardware
   - Model shared cache, speculation overheads, dependencies, squashing, etc.

Spot check with cycle-accurate simulator: Accurate within ~3%

Simulated Architecture

- Four 3-way out-of-order CPUs
  - 32K L1, 256K shared L2
- Single speculative buffer per CPU
- Forking, retiring, squashing overhead: 70 cycles each
- Speculative threads can be preempted
  - Low priority speculations can be squashed by higher priority ones

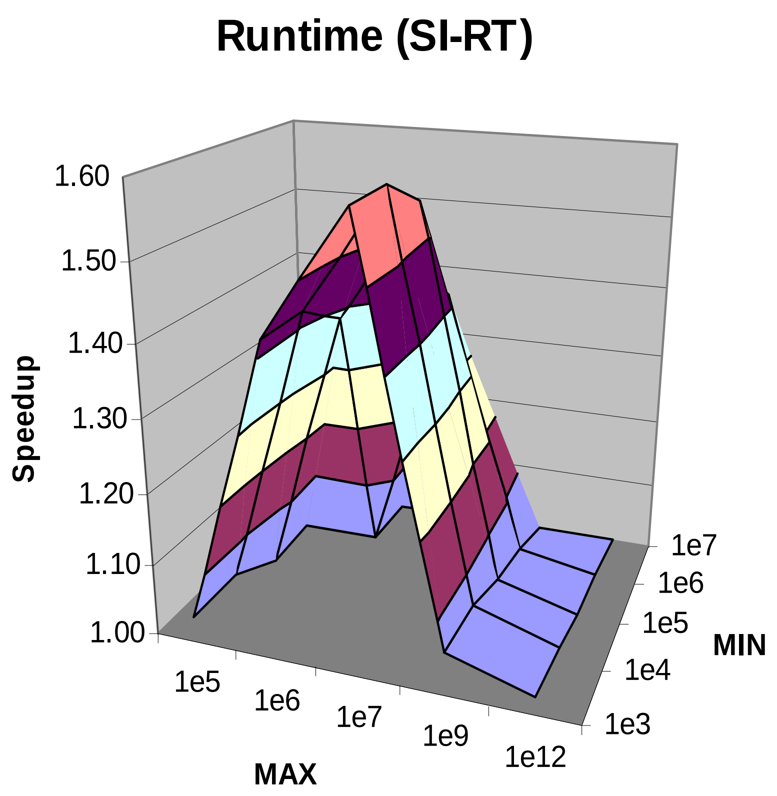
The Oracle

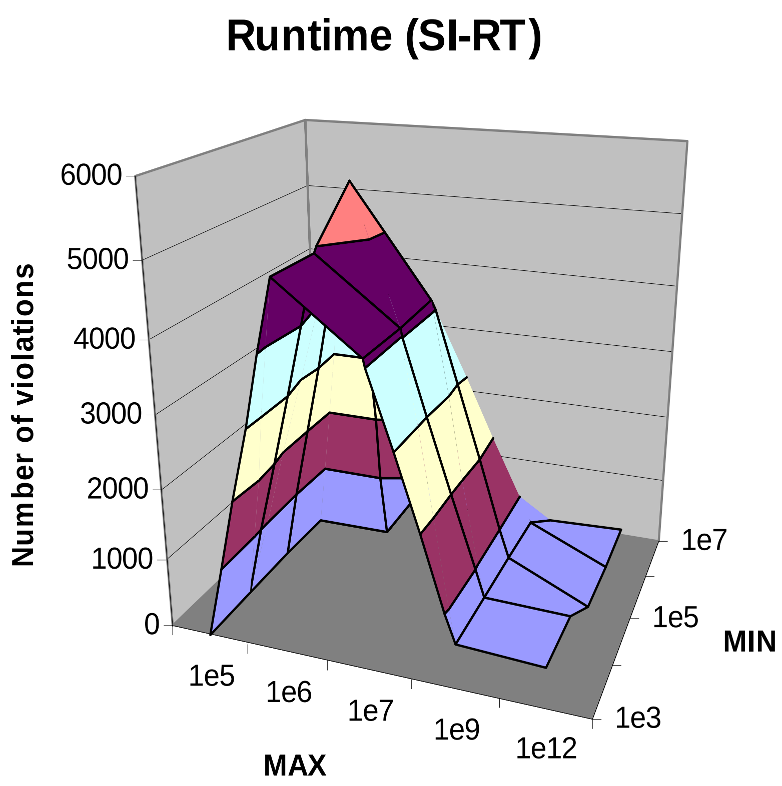
- A “Perfect” Oracle
  - Preanalyzes entire trace
  - Makes a separate decision on every method invocation
  - Chooses fork points to never violate
  - Zero overhead for forking or retiring threads
- *Upper-bound on performance of any heuristic*

Benchmarks

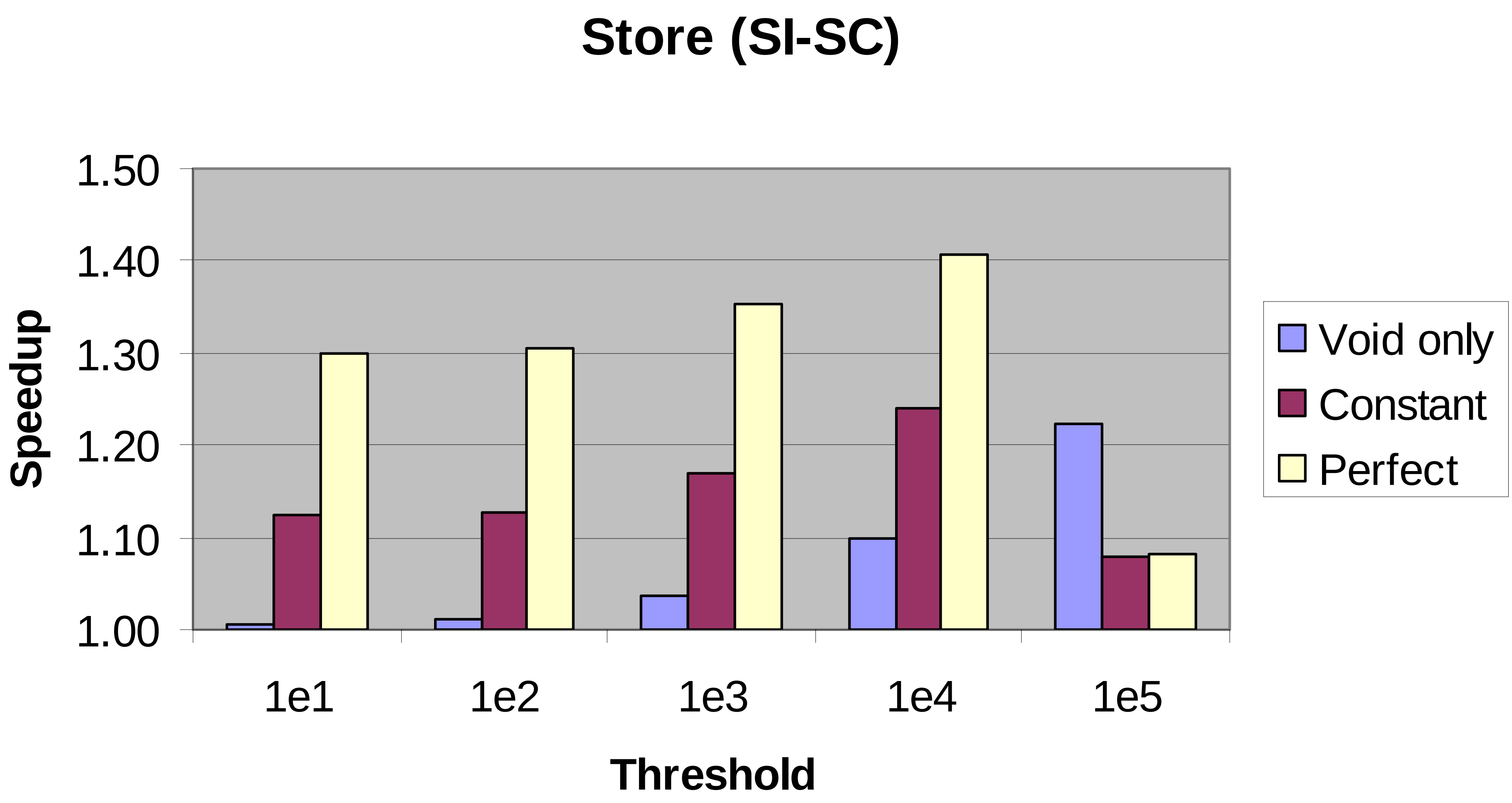
- SpecJVM
  - compress: Lempel-Ziv compression
  - jack: Java parser generator
  - javac: Java compiler from the JDK 1.0.2
  - jess: Java expert shell system
  - mpeg: Mpeg layer 3 audio decompression
  - raytrace: Raytracer that works on a dinosaur scene
- SPLASH-2
  - barnes: Hierarchical N-body solver
  - water: Simulation of water molecules

Heuristic Parameter Tuning

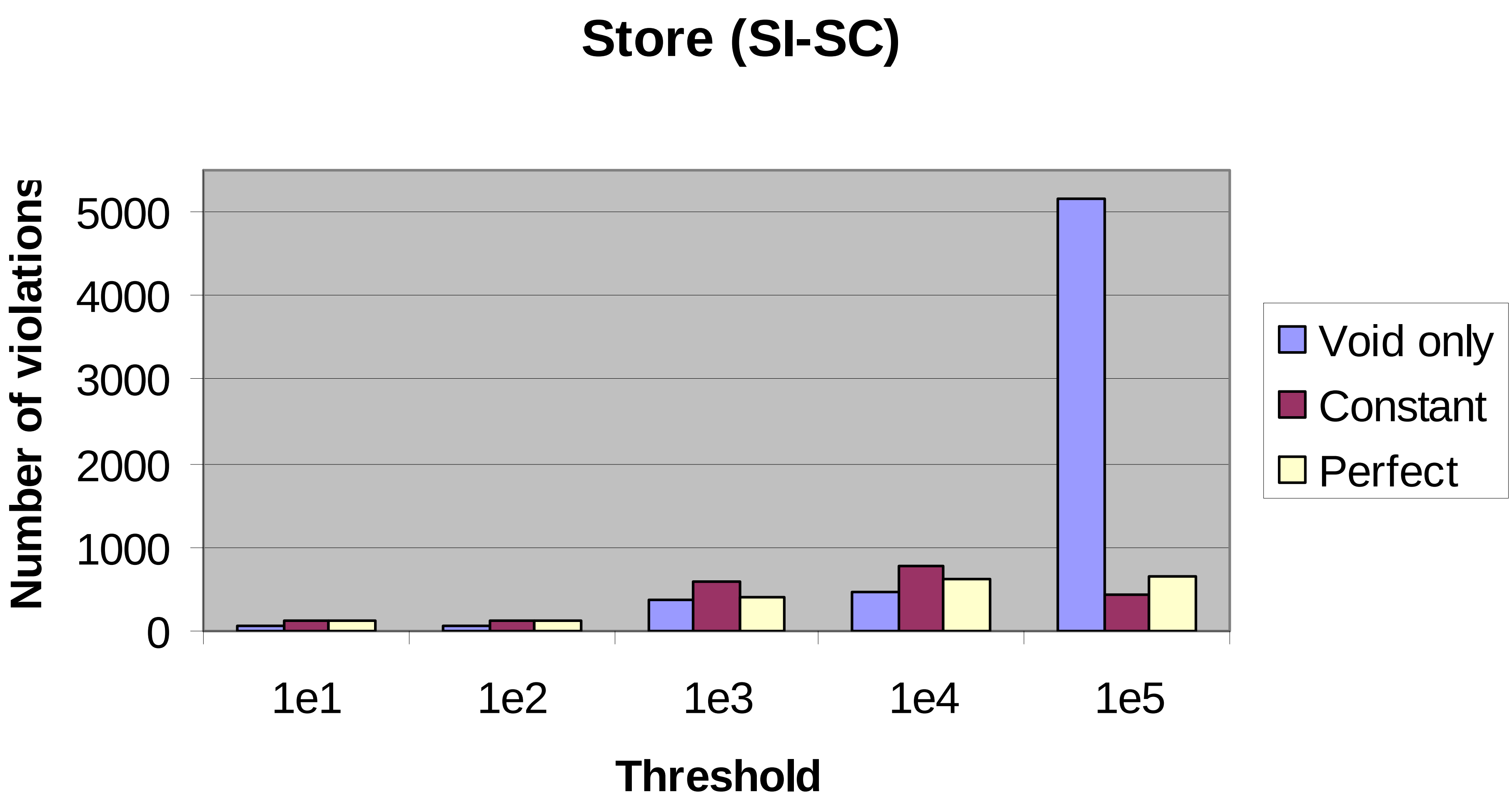




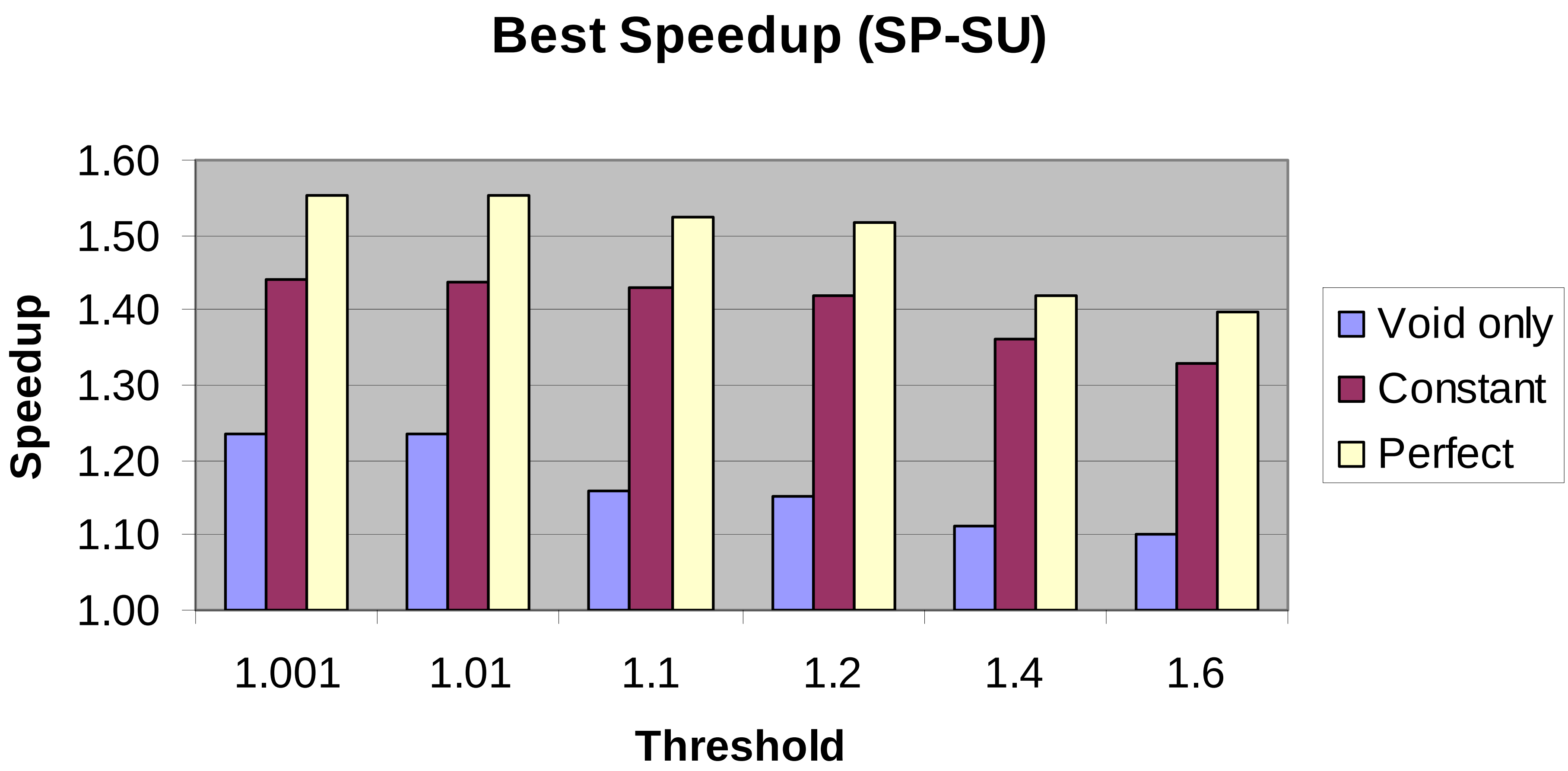
Heuristic Parameter Tuning



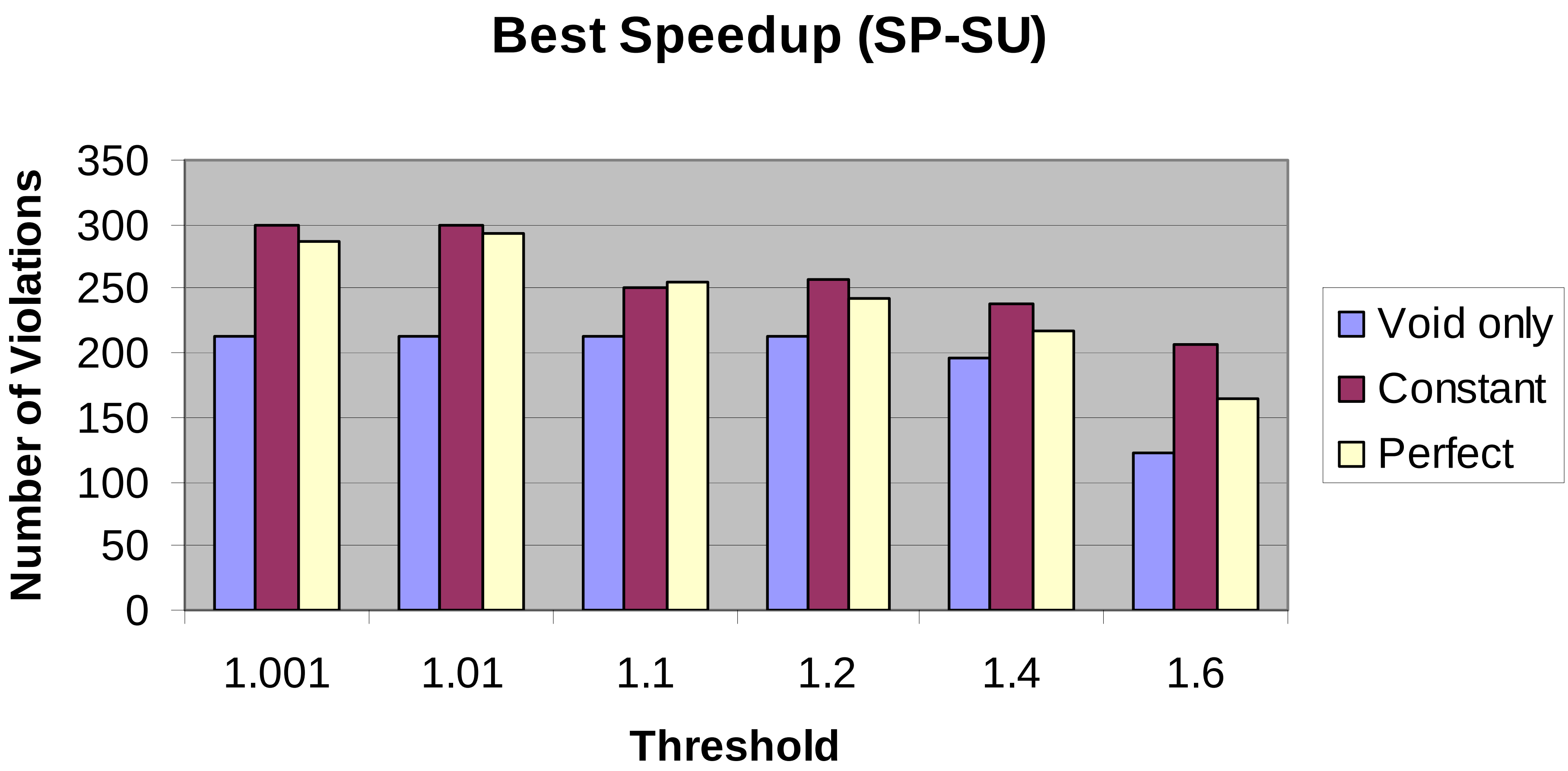
Heuristic Parameter Tuning



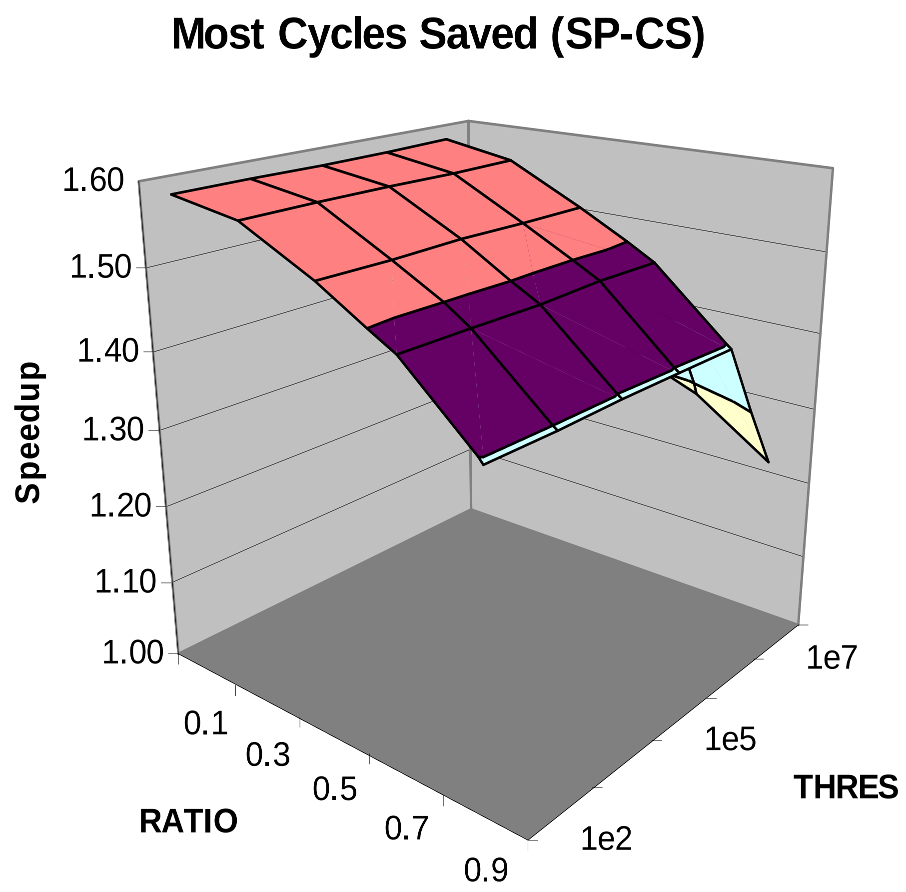
Heuristic Parameter Tuning

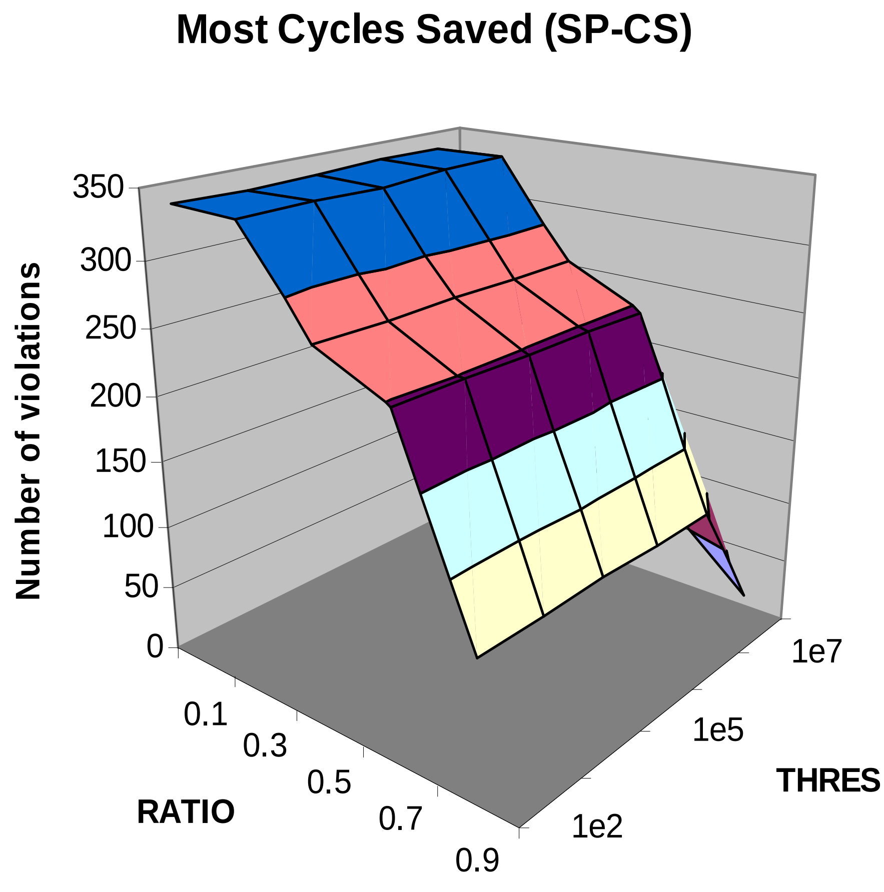


Heuristic Parameter Tuning

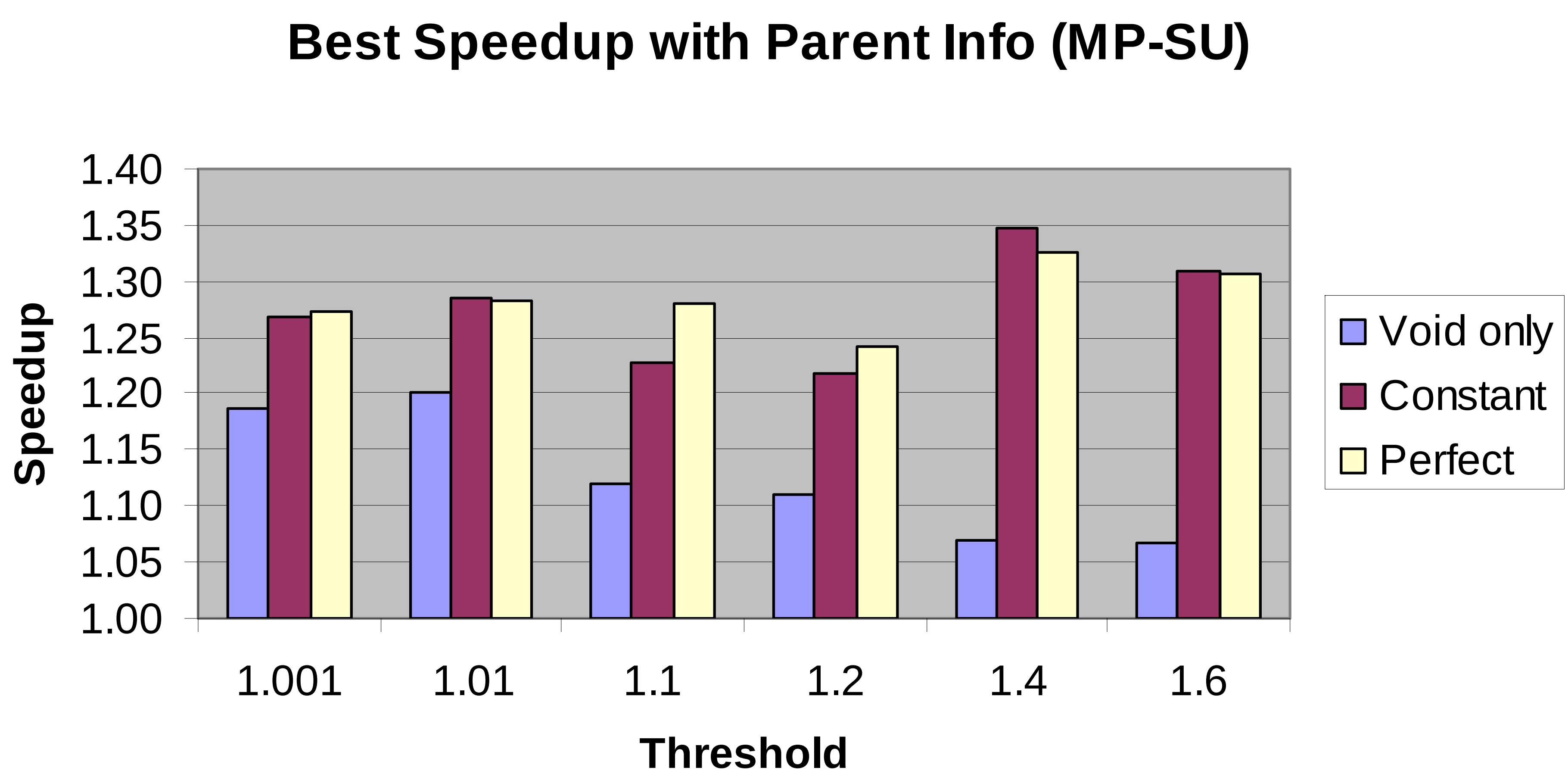


Heuristic Parameter Tuning

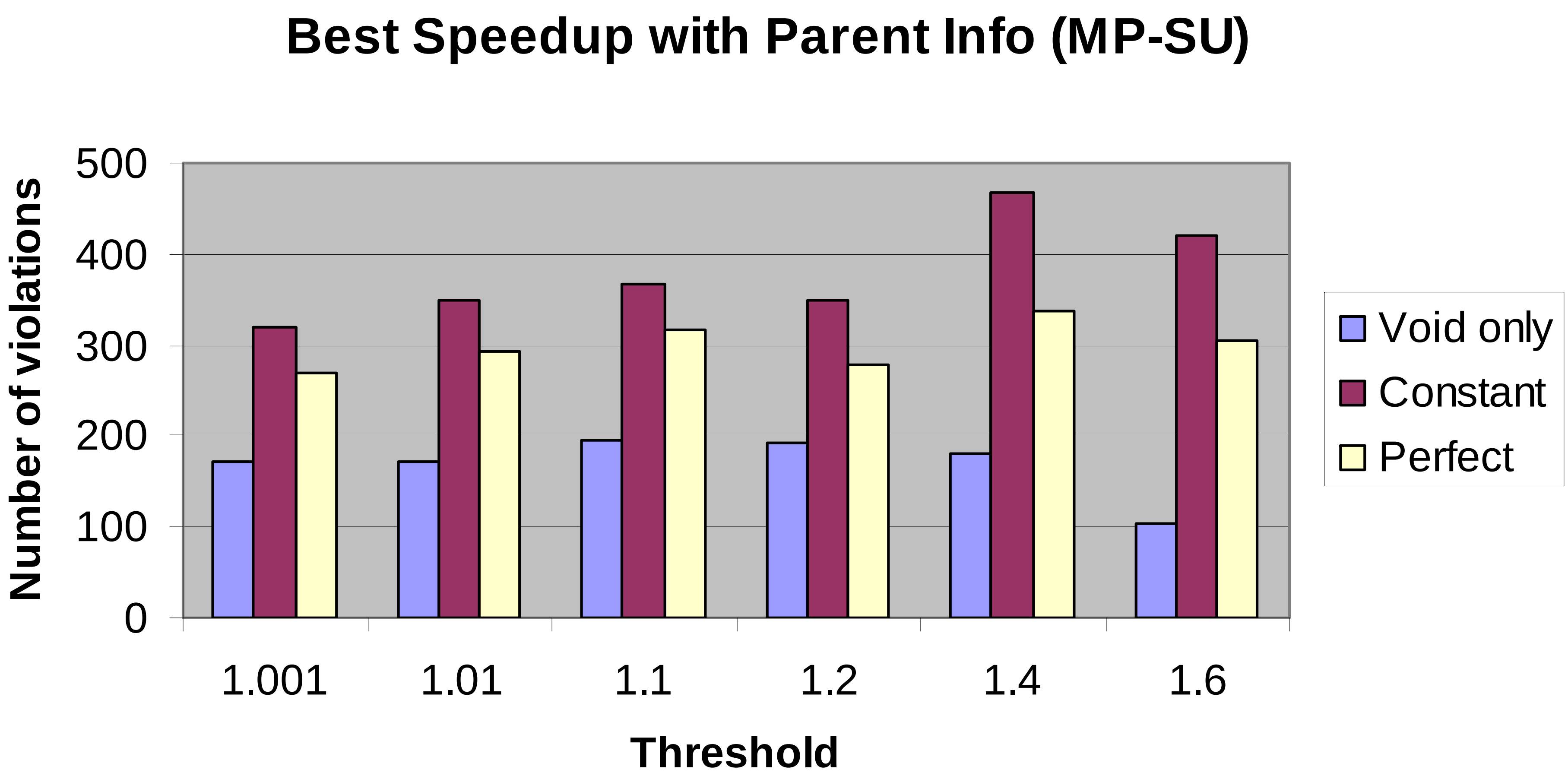




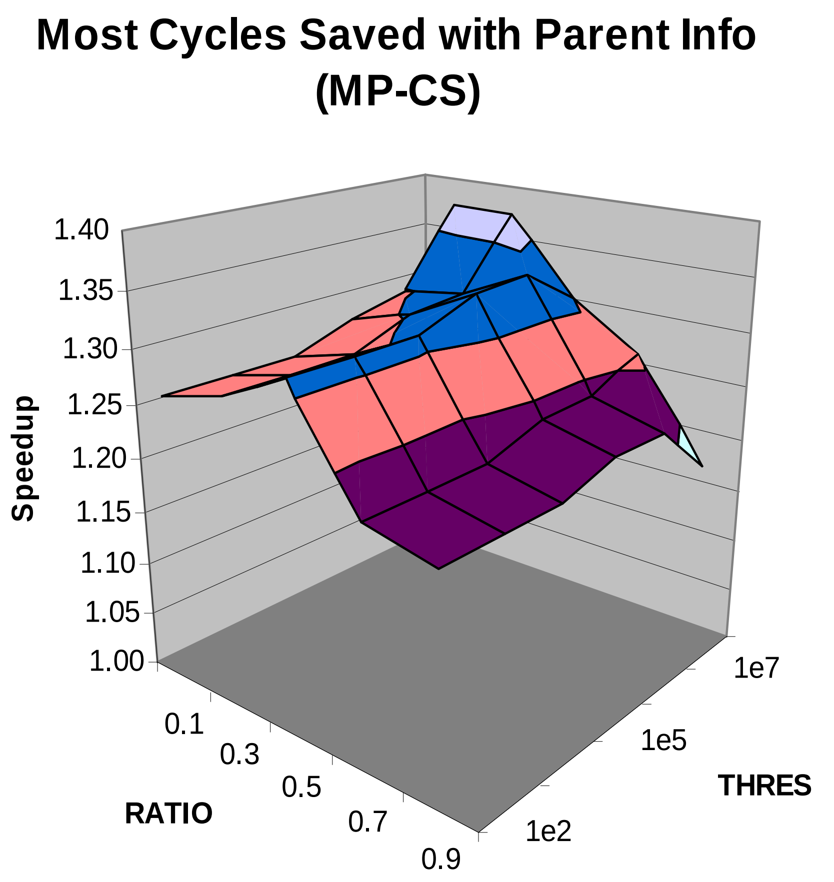
Heuristic Parameter Tuning

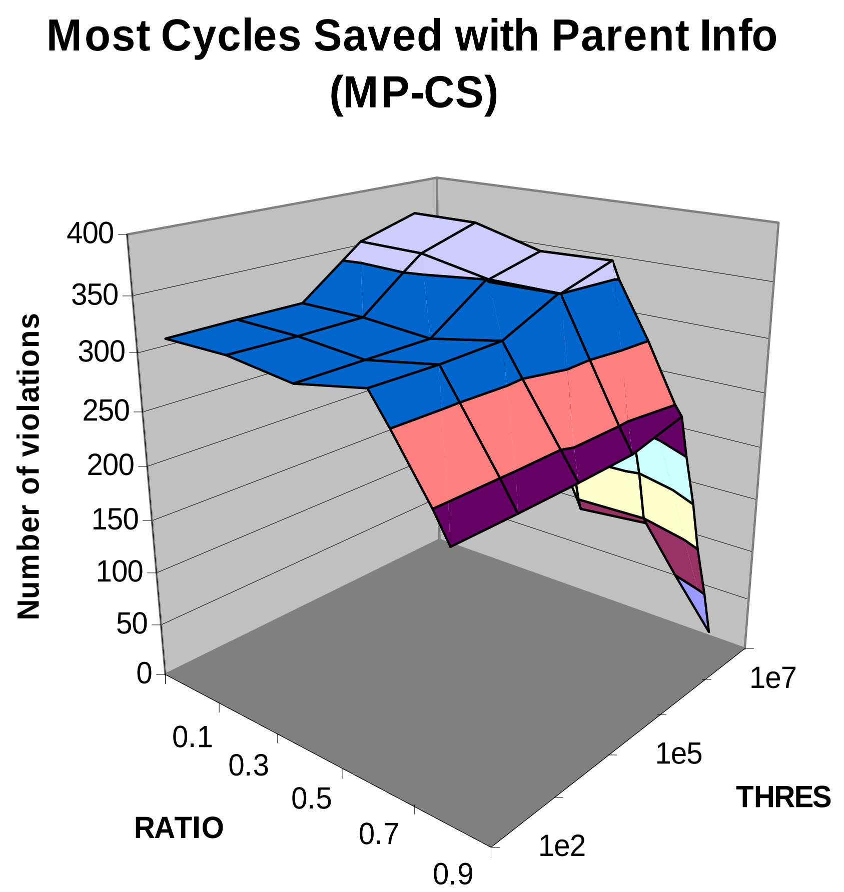


Heuristic Parameter Tuning

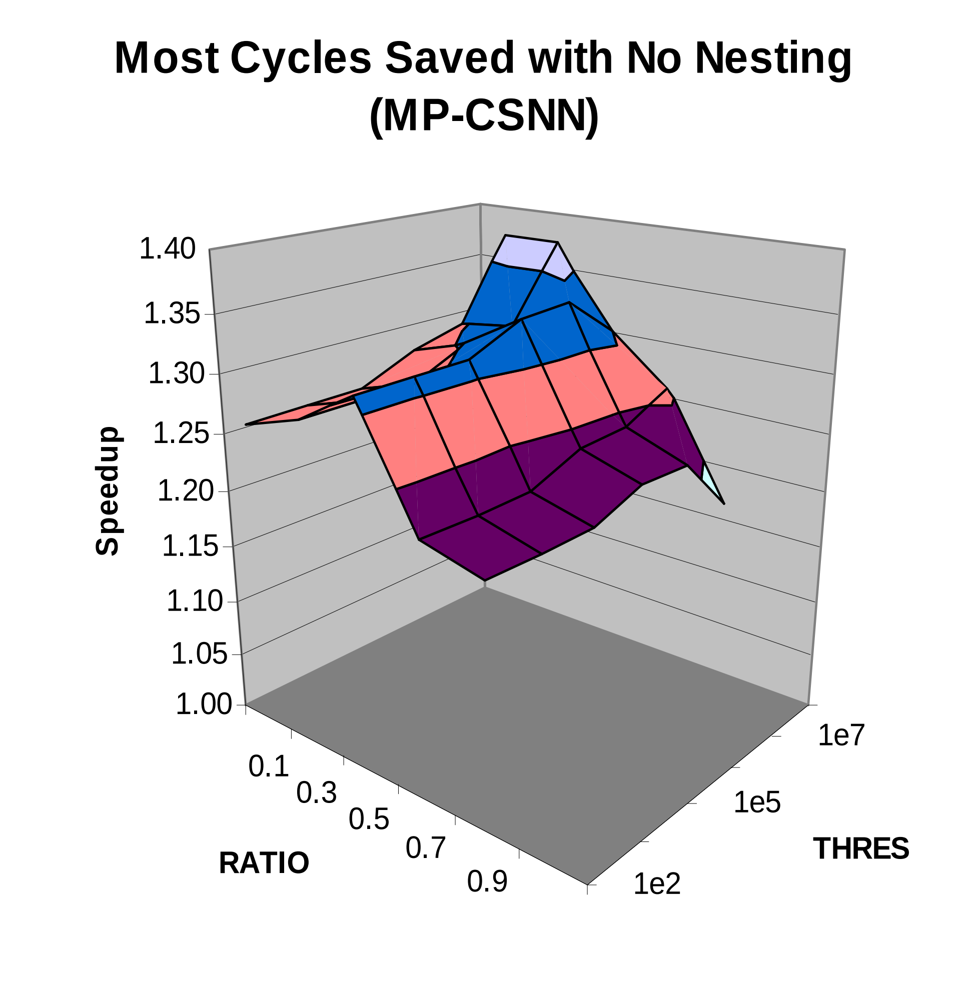


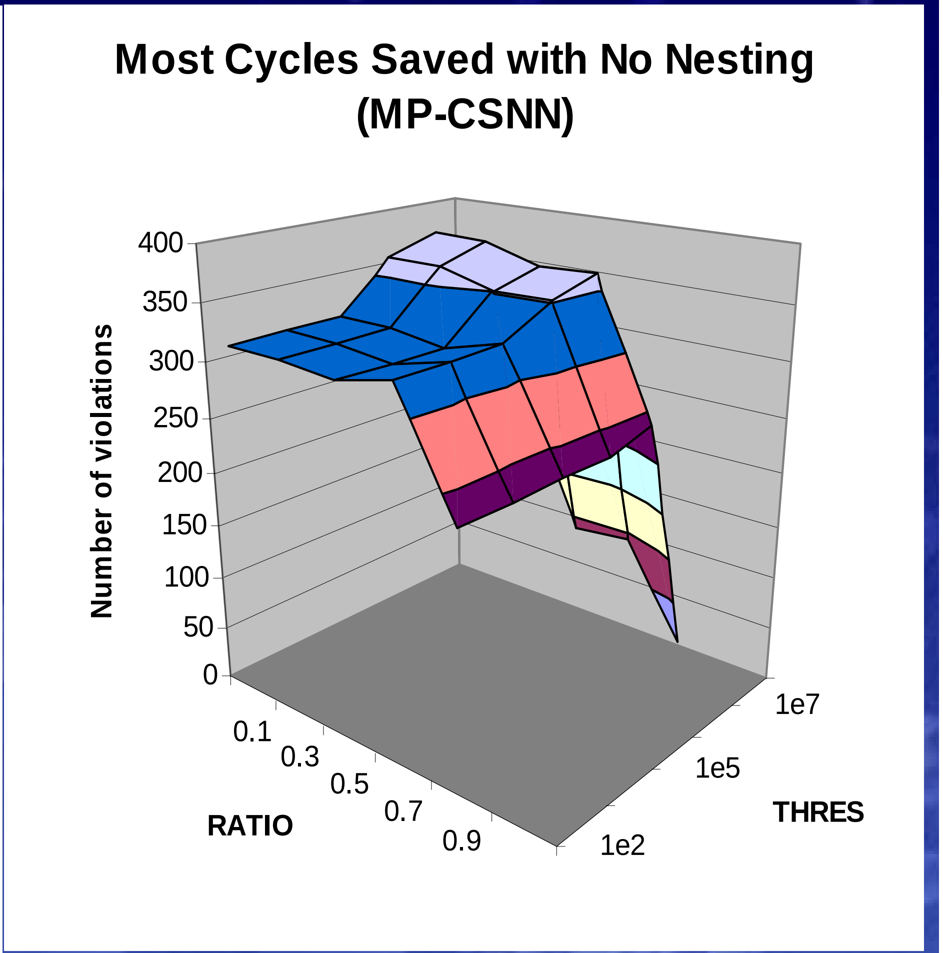
Heuristic Parameter Tuning





Heuristic Parameter Tuning

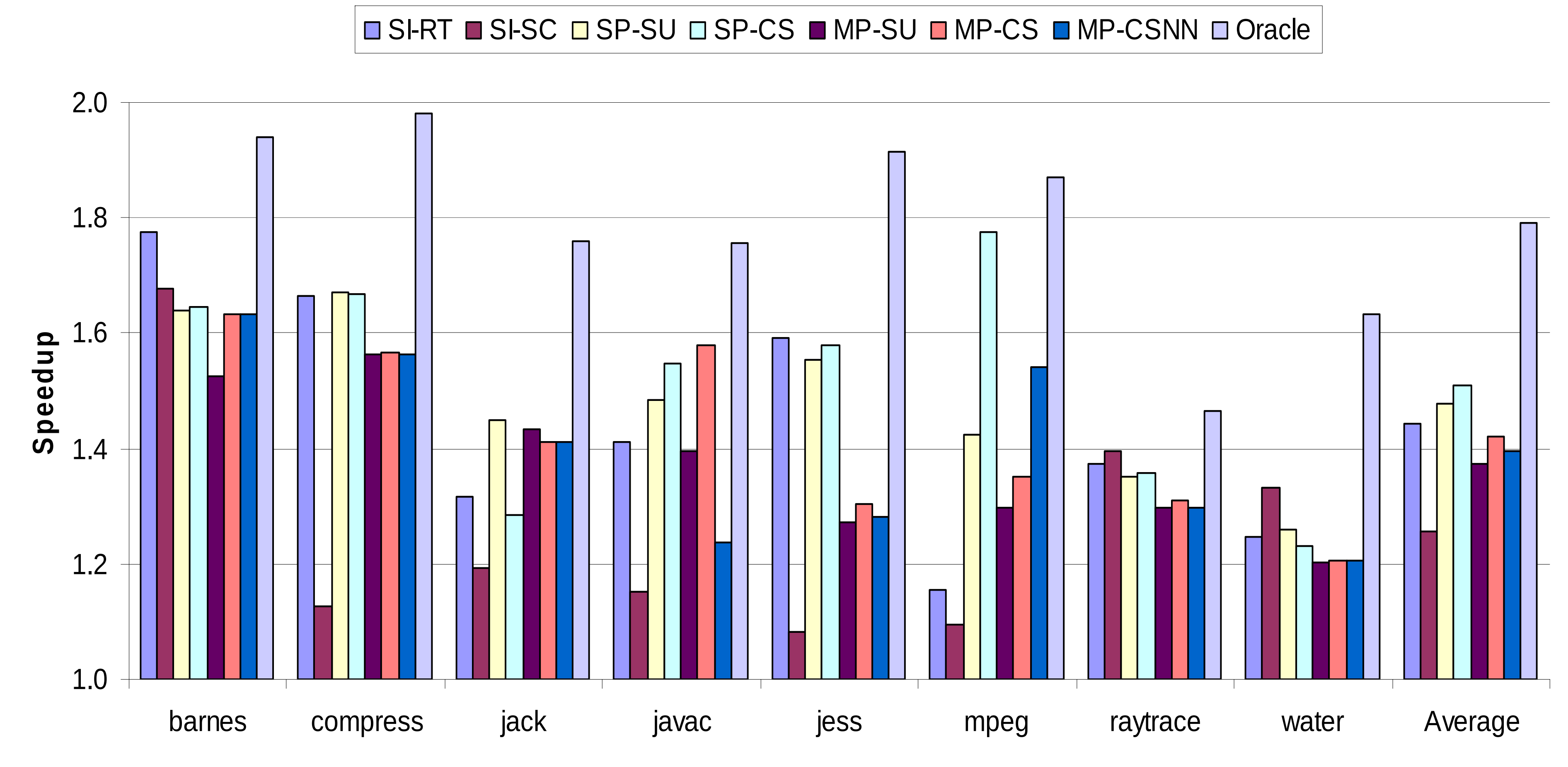




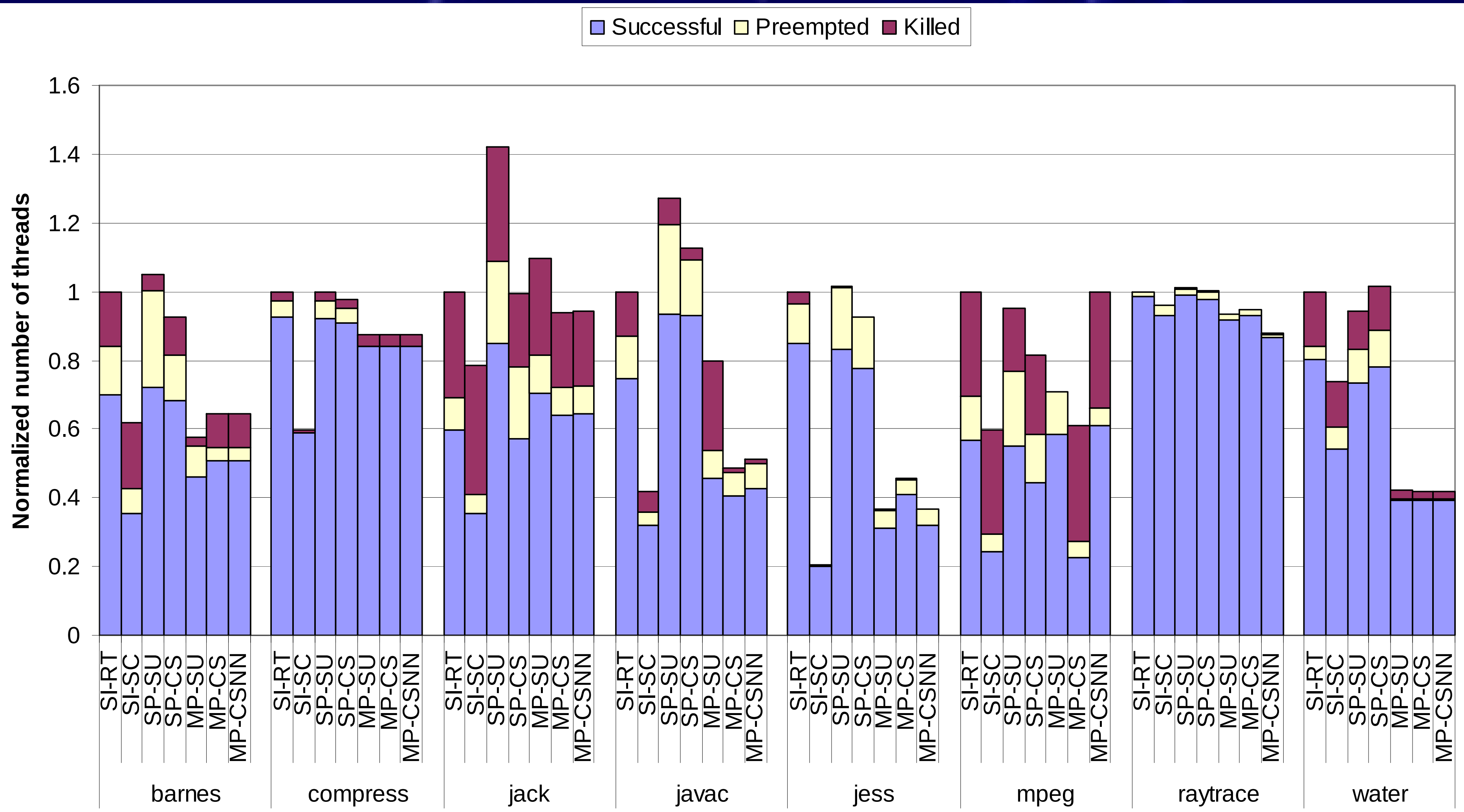
Tuning Summary

- Runtime (SI-RT):
  - MIN = 10^3 cycles, MAX = 10^7 cycles
- Store (SI-SC):
  - MAX = 10^5 stores
- Best speedup (SP-SU, MP-SU):
  - Single pass: MIN = 1.2x speedup
  - Multi pass: MIN = 1.4x speedup
- Most cycles saved (SP-CS, MP-CS, MP-CSNN):
  - THRES = 10^5 cycles saved, RATIO = 70% violation
- Return value prediction:
  - Constant is within 15% of perfect value prediction

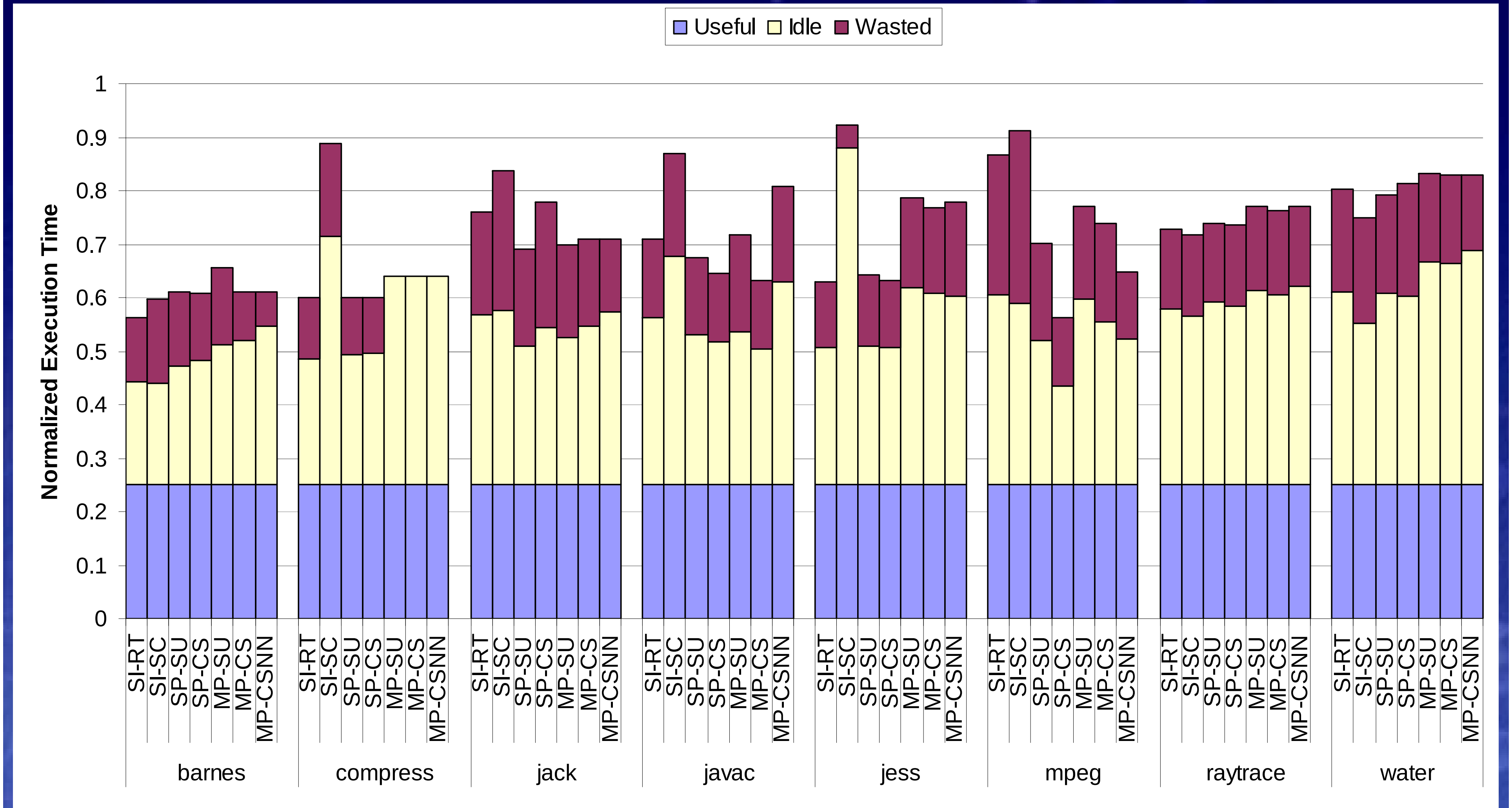
Overall Speedups



Breakdown of Speculative Threads



Breakdown of Execution Time



Speculative Store Buffer Size

	barnes	comp	jack	javac	jess	mpeg	rtrace	water
SI-RT	0.31	0.18	0.39	2.05	0.26	0.76	1.64	0.20
SI-SC	12.02	6.47	0.19	3.51	0.15	13.02	1.64	1.45
SP-SU	8.11	6.48	0.39	1.08	0.30	13.02	1.64	0.55
SP-CS	0.31	6.48	0.39	2.57	0.30	**15.29**	1.64	0.22
MP-SU	12.01	6.48	0.39	0.30	0.30	1.27	1.27	1.38
MP-CS	12.02	6.48	0.39	0.30	0.30	1.64	1.27	1.38
MP-CSNN	12.02	6.48	0.39	2.57	0.30	13.02	1.27	1.38

Maximum speculative store buffer size: 16KB

Related Work

- Loop-level parallelism
- Method-level parallelism
  - Warg and Stenstrom
    - ICPAC'01: Limit study
    - IPDPS'03: Heuristic based on runtime
    - CF'05: Misspeculation prediction
- Compilers
  - Multiscalar: Vijaykumar and Sohi, JPDC'99
  - SpMT: Bhowmik & Chen, SPAA'02

Conclusions

- Evaluated 7 heuristics for method-level speculation
- Take-home points:
  - Method-level speculation has complex interactions, very hard to predict
  - Single-pass heuristics do a good job: 80% of a perfect oracle
  - Most important issue is the balance between over- and under-speculating