

# Boiling the Ocean Is NOT the Only Option

**Title Inspired by David Patterson**

David I. August

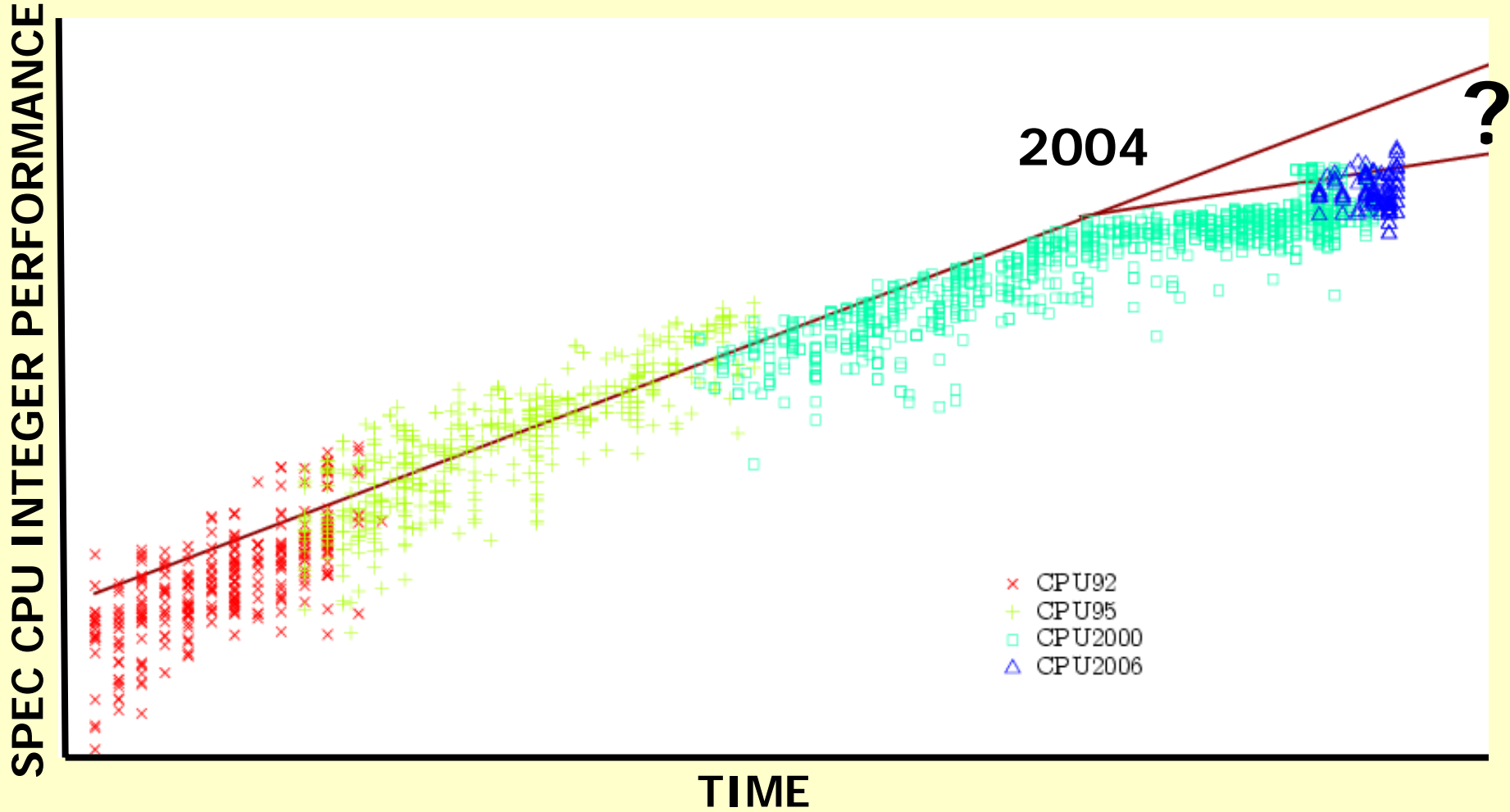
Department of Computer Science  
Princeton University

**LIBERTY RESEARCH GROUP**

**"I don't intend to offend anyone." – Ian Watson**



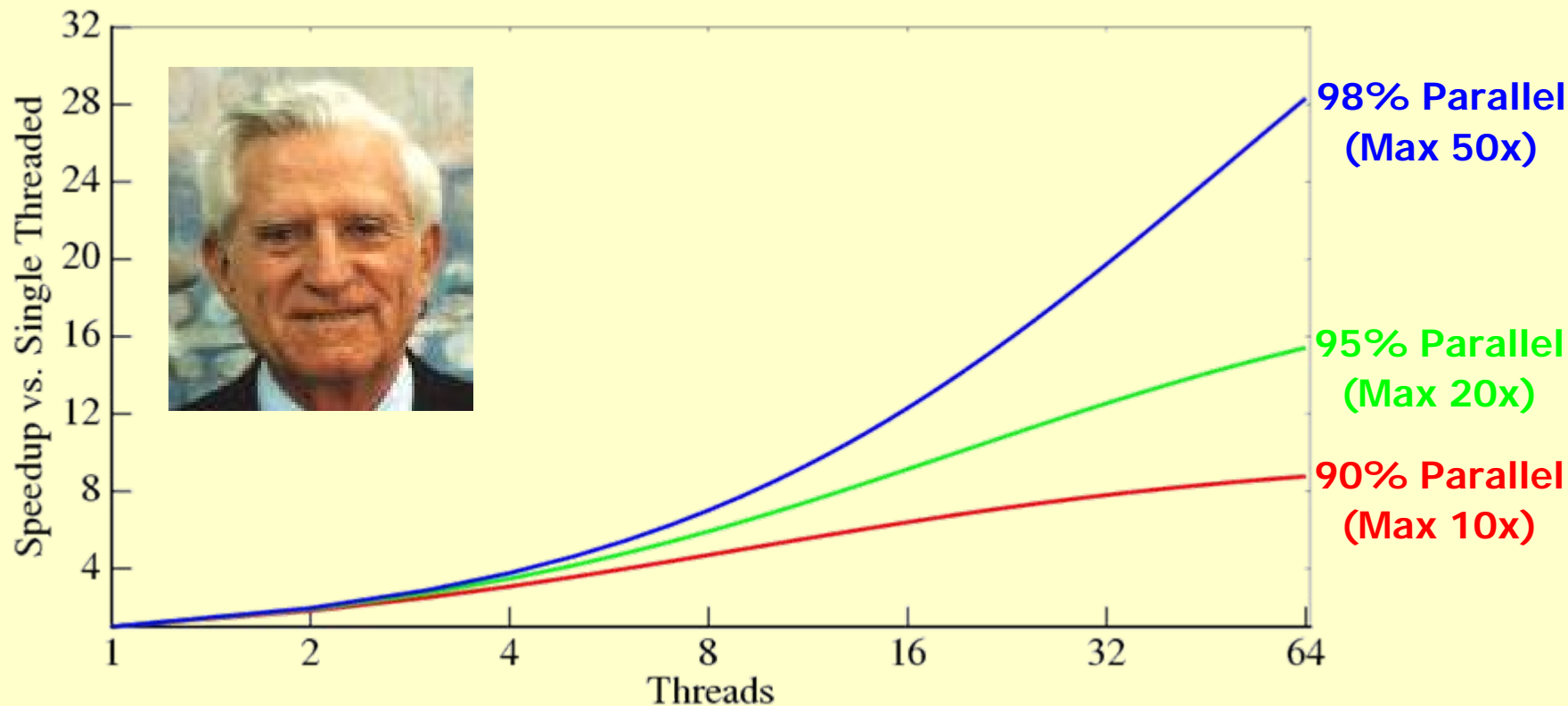
# THIS is the Problem!



# Why New Programming Models Will Fail

1. Money is earned by relieving customer pain
2. Programmers adopt new programming models
3. Legacy code
4. The Market: \$2 Trillion in SW + IT vs. \$200 Billion in HW
5. Parallel programming is more difficult
6. Parallel programming models have longevity issues

# It's ALL About Sequential Code!



“Sequential Portion” – More about limitations of languages and programmer tolerance for pain.

Ordinary sequential codes with irregular memory accesses and complex control flow. “All that icky pointer chasing code...” – Tim Mattson

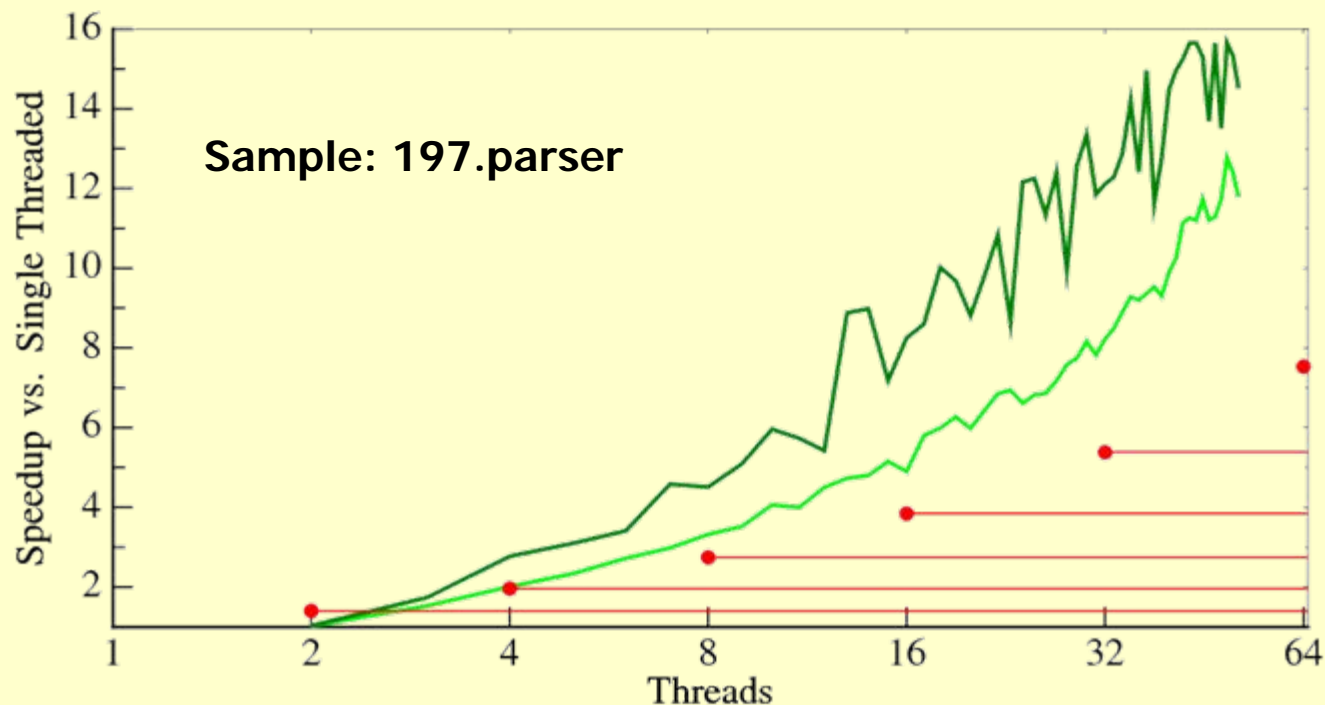
# Automatic Parallelization

“Before we design another programming model,  
we have to pause and understand...” – **Tim Mattson**

On SPEC CINT 2000 on 32 core Itanium 2 (simulated):

GEOMEAN Speedup: **5.54x** (vs. historic trend 5.38x)

Total Changes: **56 LOCs**



# THE SOLUTION

“It’s a mistake to design hardware first and let the compiler writers handle it.” – **Ian Watson**

- Automatic Parallelization
  - Whole-Program Optimization [PLDI 2006]
  - Compiler Controlled Speculation
  - DSWP (similar to DOPIPE) [MICRO 2005]
  - Memory analysis and profiling (such as in David Penry’s dynamic reparallelization)
- Hardware
  - Support for Compiler Controlled Speculation
  - Support for inter-thread queues [MICRO 2006]
- Minimal Changes to Software
  - Legacy sequential languages
  - Limitation: Alternate valid outcomes not expressible
  - Add 2 keywords (23 of the 56 LOCs).

“The best parallel algorithm is the best parallel implementation of the best sequential algorithm.” – **Michael Heroux**

# But Automatic Parallelization of Arbitrary code is Impossible!

“That isn't to say we are parallelizing arbitrary C code, that's a fool's errand!” – **Richard Lethin**

“Compiler can't determine a tree from a graph” –  
**Burton Smith**

[PLDI 2007]

“Compiler can't determine dependences without type information. Even then...” – **Burton Smith**

## Reality:

- ILP research more relevant than traditional automatic parallelization
  - Compiler Controlled Speculation!
  - TLP extraction easier than ILP extraction!
- Automatic parallelization needs hardware support, respect, and...

“Funding is the issue.” – **Burton Smith**



# Conclusions

---

What we know:

- Single thread programming is complicated
- Parallel programming is worse (less worse with new developments)
- Augmented sequential programming models have nice properties
- Automatic parallelism is underrated

What we should do:

- Bring state-of-the-art *Region-Based Whole Program, Analysis, Optimization* techniques together in one compiler/RTO
- Provide hardware support for code generated (*TLS Memory, Queues*)
- Make incremental changes to existing sequential programming models to enable alternate outcomes

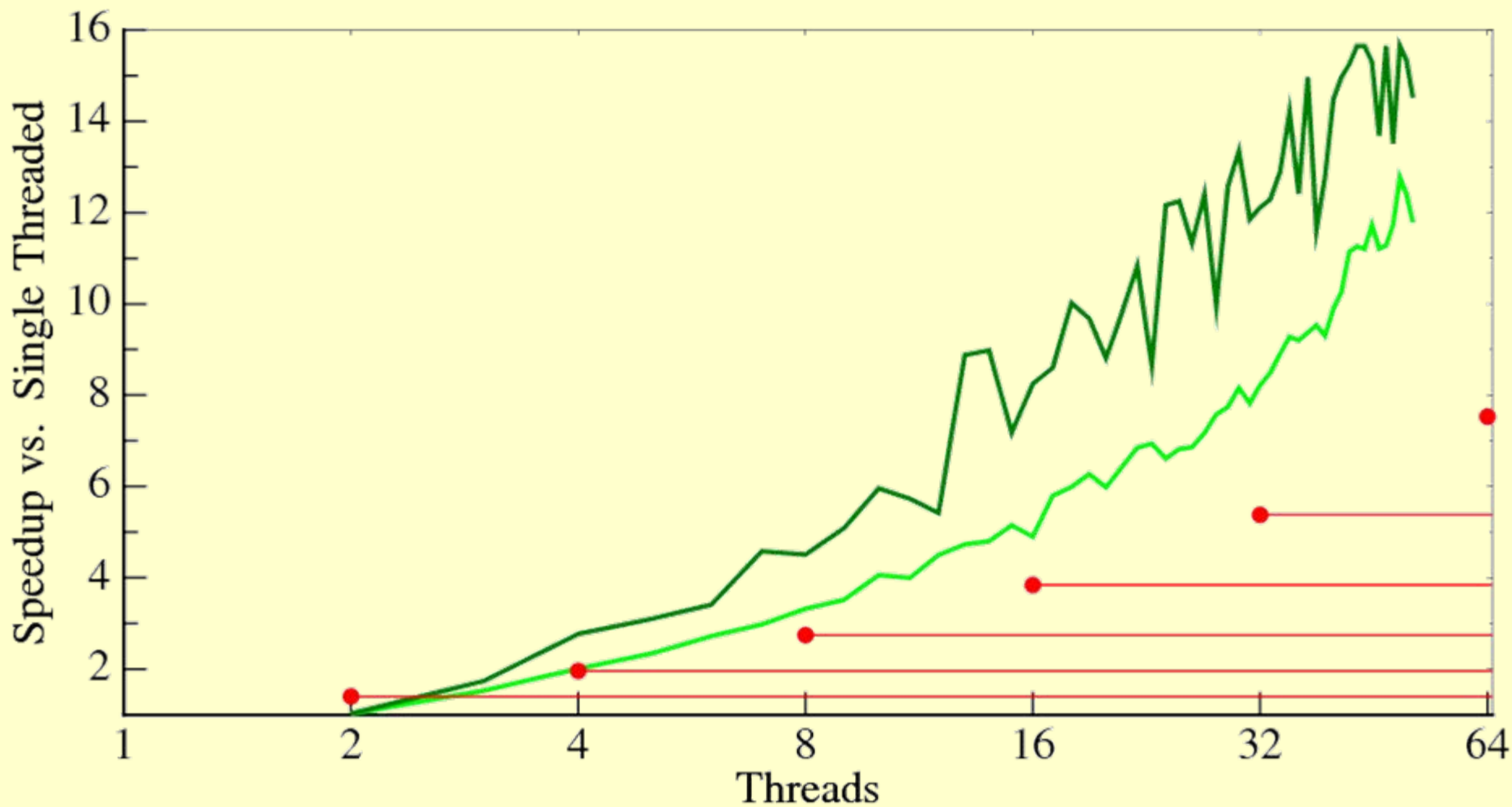


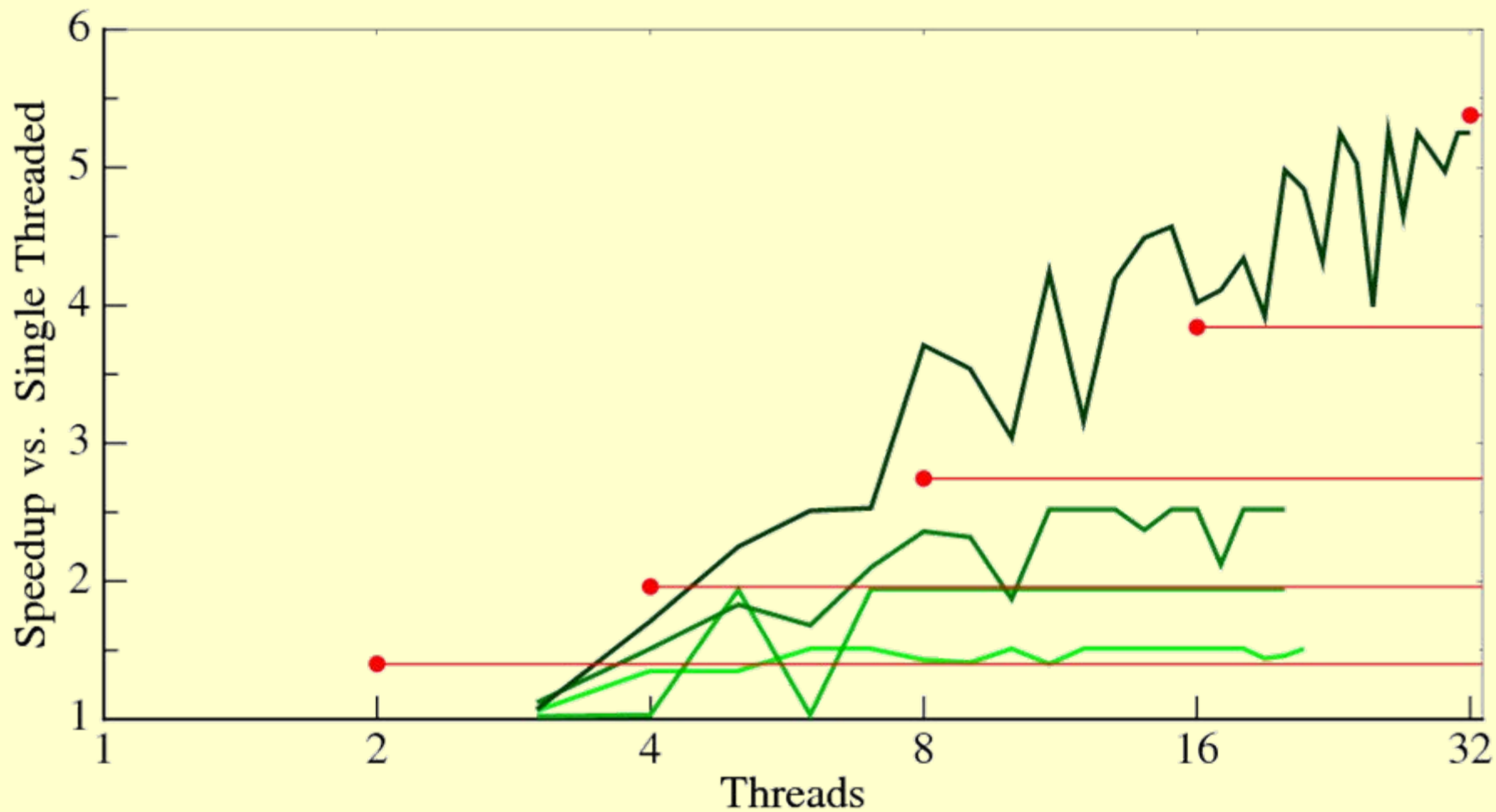
# Full Results: Performance

Benchmark	Threads at Peak	Speedup	LOCs Changed
164.gzip	32+	29.91	26
175.vpr	15	3.59	1
176.gcc	16	5.06	17
181.mcf	32+	2.84	0
186.crafty	32+	25.18	9
197.parser	32+	24.50	2
253.perlbmk	5	1.21	0
254.gap	10	1.94	1
255.vortex	32+	4.92	0
256.bzip2	12	6.72	0
300.twolf	8	2.06	1
<b>GEOMEAN</b>	<b>17</b>	<b>5.54</b>	
<b>ARITHMEAN</b>	<b>20</b>	<b>9.81</b>	

# Full Results: How Code Was Transformed

<b>Benchmark</b>	<b>LOC (All)</b>	<b>LOC (Model)</b>	<b>Model Techniques</b>	<b>Compiler Techniques Applied</b>
164.gzip	26	2	Y-Branch	TLS Memory, DSWP
175.vpr	1	1	PURE	Alias, Value, & Control Spec, TLS Mem, DSWP
176.gcc	17	7	PURE	Alias & Control Spec, TLS MEM, DSWP
181.mcf	0	0		Alias, Silent Store, & Control Spec, TLS Mem, DSWP, Nested
186.crafty	9	9	PURE	TLS Mem, DSWP, Nested
197.parser	2	2	PURE	TLS Mem, DSWP
253.perlbnk	0	0		Alias, Control, & Value Spec, DSWP
254.gap	1	1	PURE	TLS Memory, DSWP, Alias Spec
255.vortex	0	0		Alias & Value Spec, TLS Mem, DSWP
256.bzip2	0	0		TLS Memory, DSWP
300.twolf	1	1	PURE	Alias & Control Spec, TLS Mem, DSWP





```
void **table;

@PURE
void *lookup_or_insert(void *item) {
    void *result = get(table, item);
    if (result == NULL) {
        put(table, item);
        result = item;
    }
    return result;
}
```

(a) Cache Example

```
char memory[MAX_MEMORY];
void * free_list;

@PURE
void * my_malloc(int size) {
    void *memory = find_memory(free_list, size);
    update_free_list_pointers(free_list, memory);
    return memory;
}

@PURE
void my_free(void *ptr) {
    update_free_list_pointers(free_list, ptr);
}
```

(b) Memory Manager Example

**Figure 2.** Motivating Examples for the *PURE* Addition to the Sequential Programming Model

# Y-Branch

```
start_dictionary();
while ((char = read(1)) != EOF) {
    profitable = compressChar(char)

    @YBRANCH(probability=.000001)
    if (!profitable)
        finish_and_restart_dictionary();
}
finish_dictionary();
```

(a) Y-branch

```
#define CUTOFF 100000
start_dictionary();
int count = 0;
while ((char = read(1)) != EOF) {
    profitable = compressChar(char)

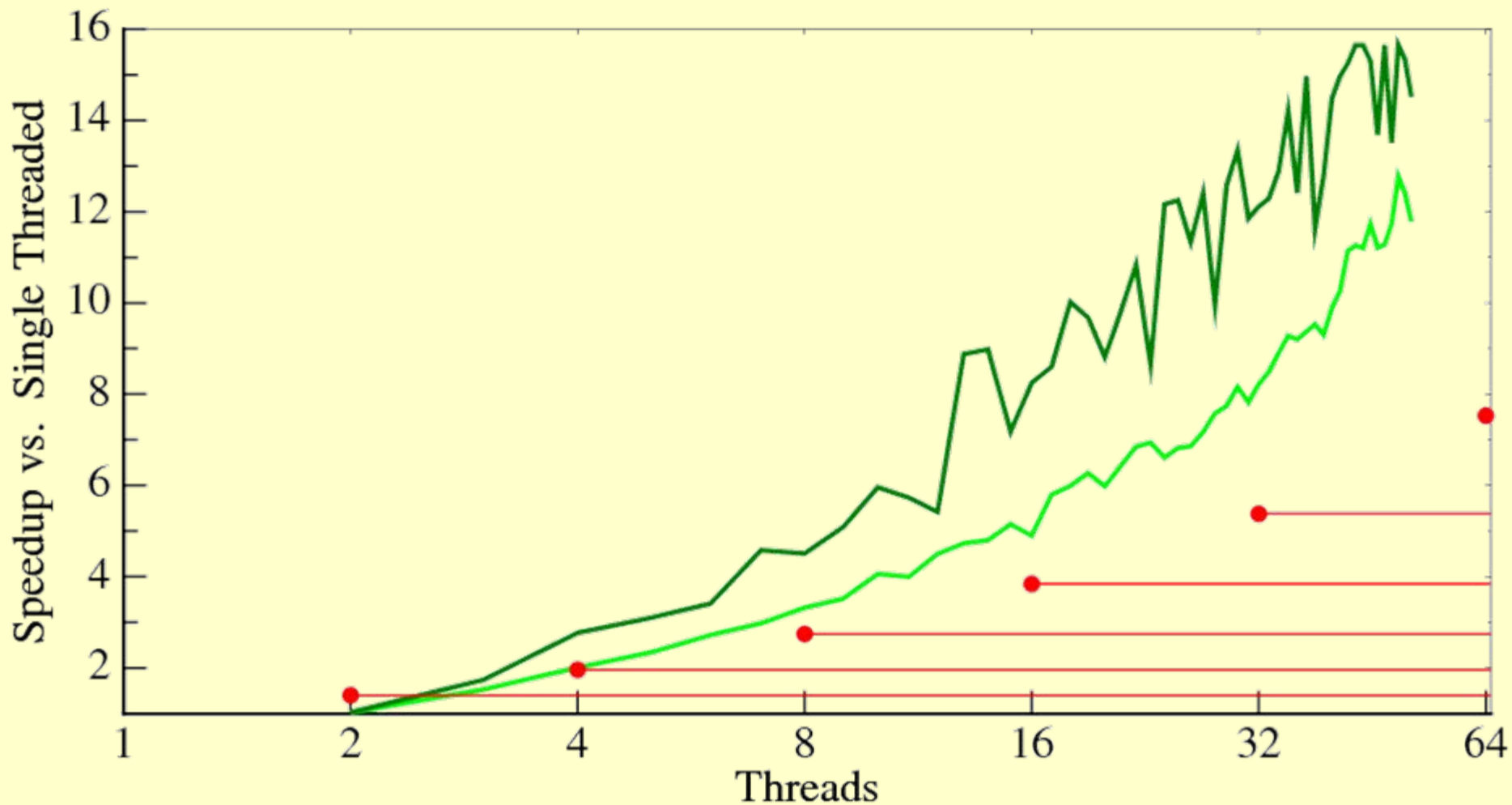
    if (!profitable || count == CUTOFF) {
        finish_and_restart_dictionary();
        count = 0;
    } else {
        count++;
    }
}
finish_dictionary();
```

(b) Manual Choice of Parallelism

**Figure 1.** Motivating Examples for the Y-branch Addition to the Sequential Programming Model

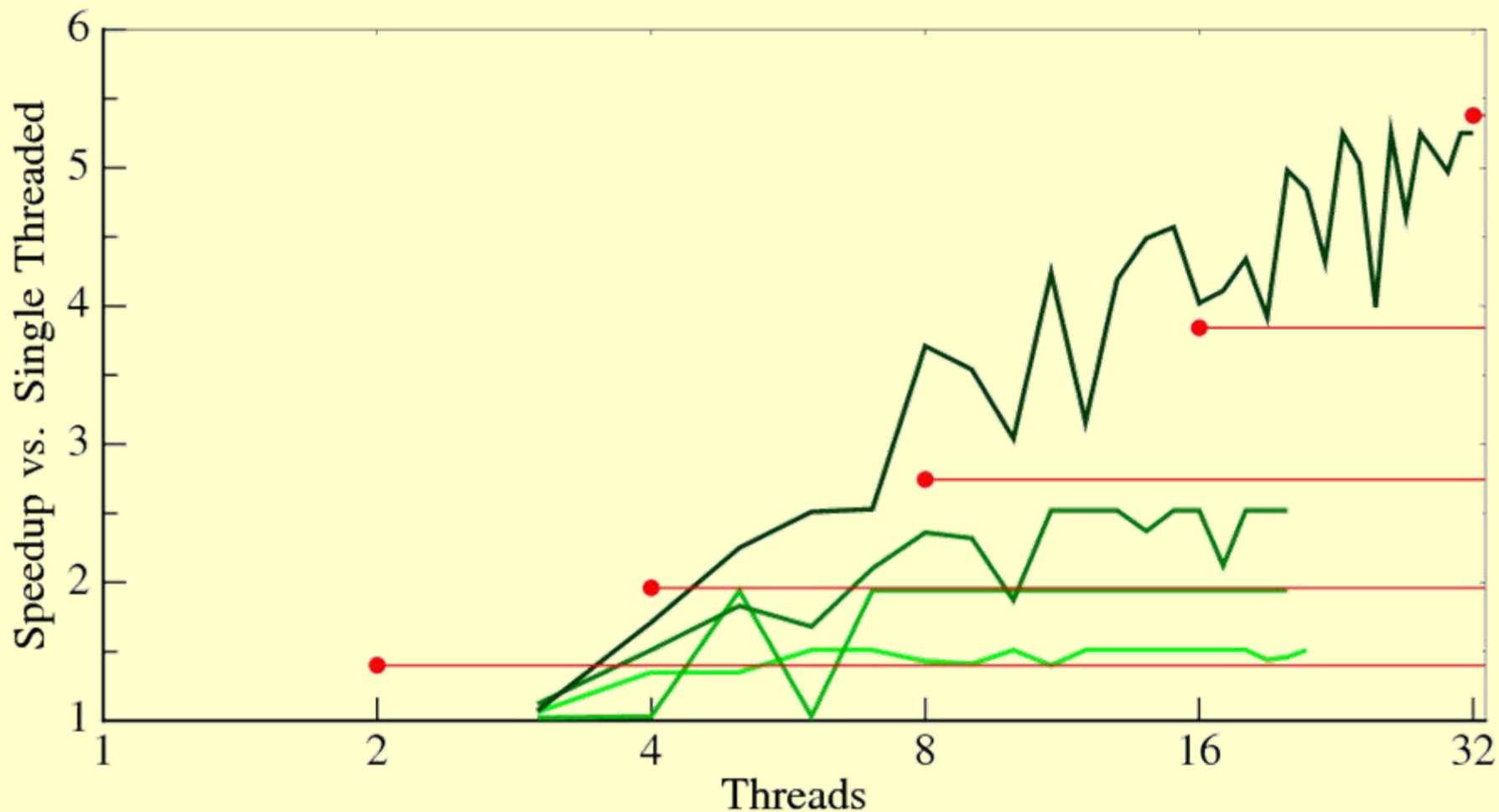
# SPEC 2000: 197.parser

## Automatic Parallelization



Threads run on multicore model with Itanium 2 cores.

# Automatic Parallelization



Threads run on multicore model with Itanium 2 cores.

