

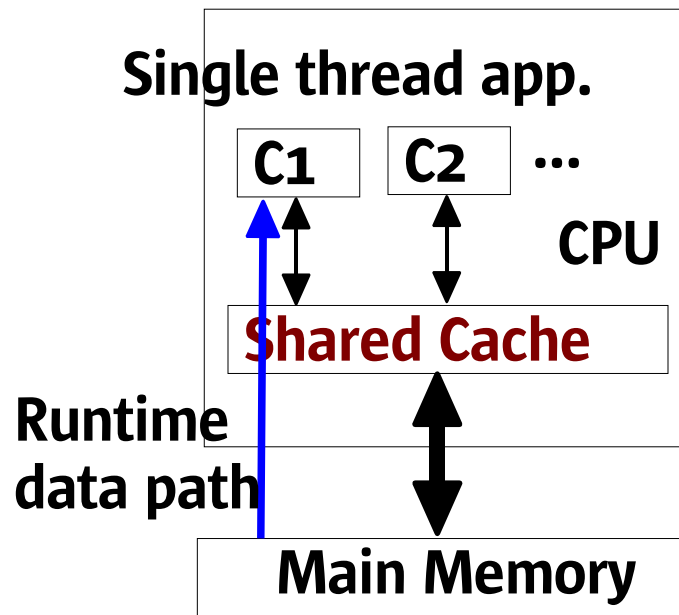
# **Design and Implementation of A Compiler Framework for Helper Threading on Multi-Core Processors**

**Yonghong Song, Spiros Kalogeropoulos  
and Partha Tirumalai**

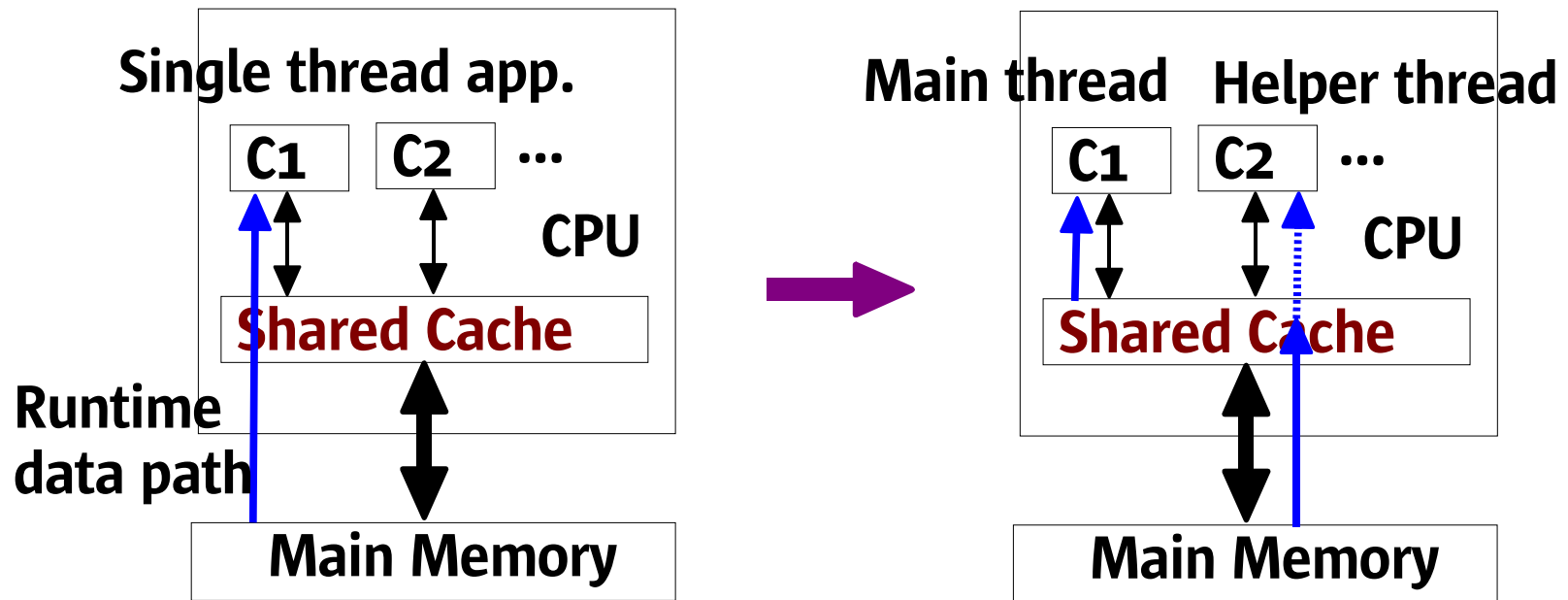
**September, 2005**

**Remember Single-Threaded Application Performance!**

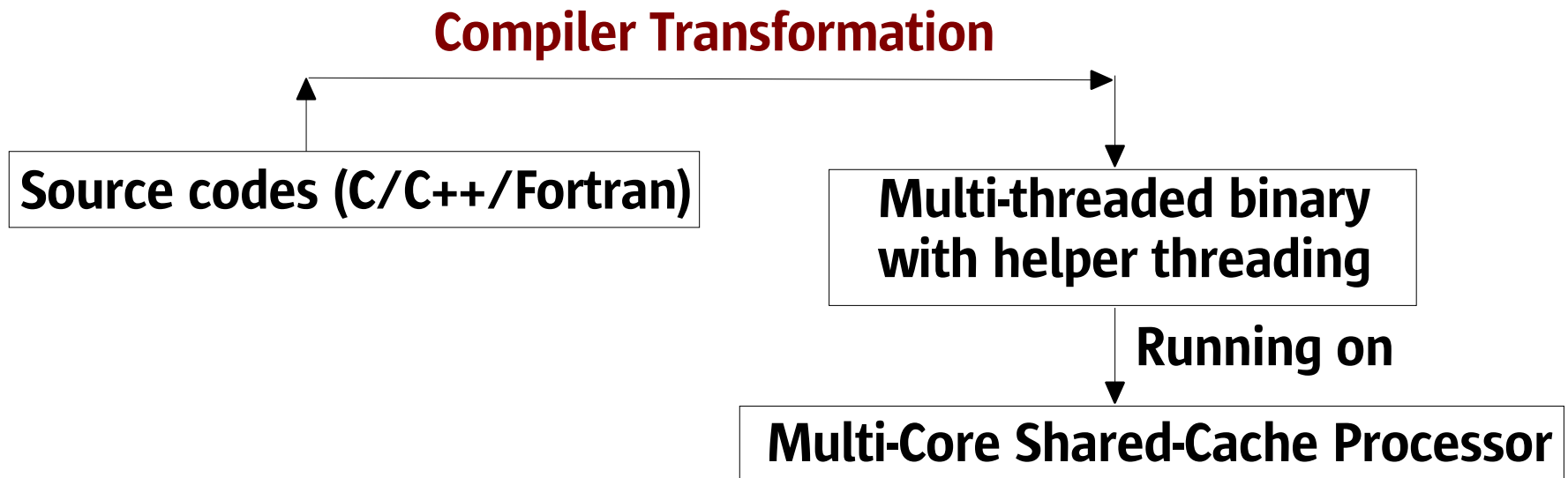
# Single Thread Application



# Helper Threading



# Compiler Framework



# Outline

- Motivation for helper threading
- Design requirements
- The compiler framework
- Performance results
- Related works
- Conclusions

# Motivation

- Interleaved prefetching in single-threaded applications
- Not effective when
  - Large overhead to compute future even-predictable addresses (e.g., `a[index[i]]`)
  - Pointer-chasing codes

# Design Requirements

- Existing hardware
- Avoid slowdown of the main thread
- Ensure the helper thread doing **useful** work

# Helper Thread

- A stripped original code
  - Helper thread **expected** to run faster than the main thread
- No user visible side effect
- Necessary synchronization code with the main thread

# Original Example Code

```
while (p->val > VAL) {  
    if (p->data != 0) {  
        q = p->data;  
        while (q) {  
            q->data = c;  
            q = q->next;  
        }  
    }  
    p = p->next;  
}
```

# Potential Benefit

- Ld/st ==> prefetch in helper thread
- **Effective** prefetch candidates
  - Lds not used to compute branch cond or address of another ld/st
  - Prefetch analysis disqualify ld/st's hit in the cache

# Effective Prefetch Candidate

```
while (p->val > VAL) {  
  if (p->data != 0) {  
    q = p->data;  
    while (q) {  
      q->data = c;  
      q = q->next;  
    }  
  }  
  p = p->next;  
}
```

Effective prefetch candidate

Prefetch (&(q->data))

# Computing Benefit

- $\sum \text{est\_cache\_miss\_rate} * \text{cache\_miss\_penalty} * \text{est\_access\_count}$ , for all effective prefetch candidates
- Estimated cache miss rate (main thread interleaved?, addr complexity?)
- Access count (compiler analysis, profile feedback, symbolic analysis, predefined)

# Profitability

- Cost: startup cost and parameter passing
- Profitable if benefit  $>$  cost
- If the access count expressed symbolically, two versioning: one for helper threading and the other for serial execution

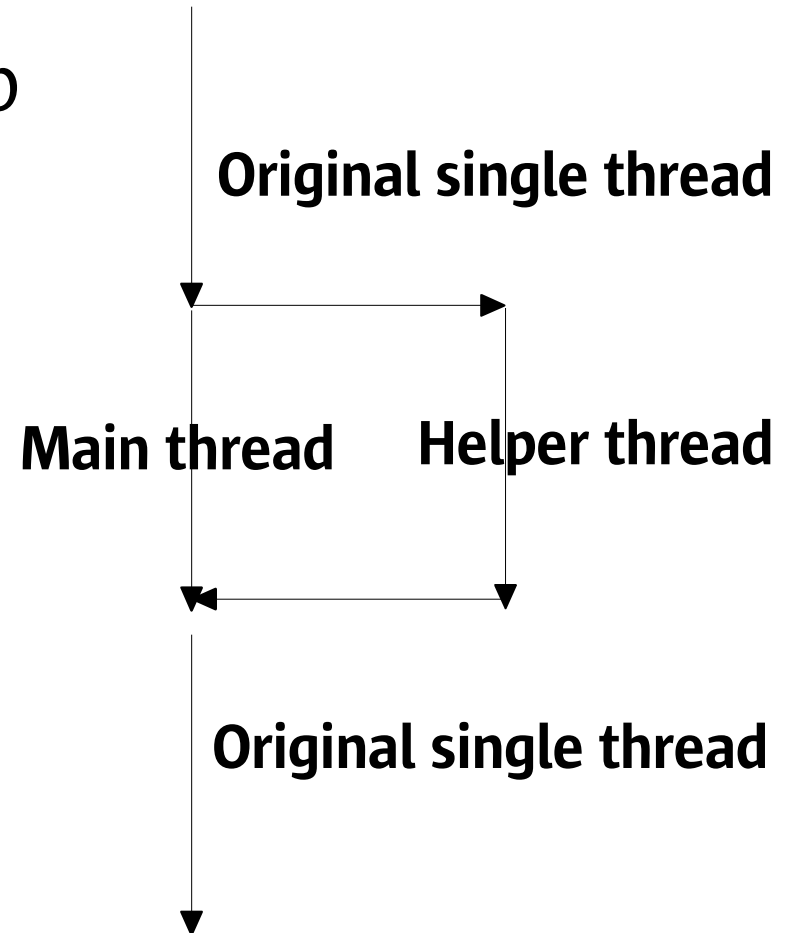
# Selecting Candidate Loops

- Loop hierarchy outside-to-inside
- Outer loop possible to amortize startup cost and maximize benefit
  - Large access counts

# Code Generation

- Two iteration parallel loop

```
for (t = 0; t <= 1; t++) {  
  if (t == 0) {  
    <original_code>  
  } else {  
    <helper_thread_code>  
  }  
}
```



# Generating Helper Thread (1)

- Data flow/control flow based slicing
- Start with condition codes and effective prefetch candidate address computation
- Effective prefetch candidates to prefetches
- Other loads to speculative loads
- Scalar renaming for **live-out** scalars

# Generating Helper Thread (2)

```
<helper_thread_code>:
/* assume p live-out, q not */
tmp_p = p;
while (tmp_p->val > VAL) {
  if (tmp_p->data != 0) {
    q = tmp_p->data;
    while (q) {
      prefetch (&q->data);
      q = q->next;
    }
  }
  tmp_p = tmp_p->next;
}
```

# Sync With Main Thread

```
is_mt_done = FALSE;
for (t = 0; t <= 1; t++) {
  if (t == 0) {
    <original_code>
    is_mt_done = TRUE;
  } else {
    <helper_thread_code2>
  }
}
```

- Helper thread may lag behind the main thread

```

<helper_thread_code2>:
if (is_mt_done == FALSE) {
    tmp_p = p;
    tmp_c = 0;
    while (tmp_p->val > VAL) {
        if (tmp_p->data != 0) {
            q = tmp_p->data;
            while (q) {
                prefetch (&q->data);
                q = q->next;
                <check_code>
            }
        }
        tmp_p = tmp_p->next;
        <check_code>
    }
}
next:

```

```

<check_code>:
if (tmp_c >= check_c) {
    if (is_mt_done == TRUE) {
        goto next;
    }
    tmp_c = 0;
}
tmp_c = tmp_c + 1;

```

- Permits inexact control flow in helper thread

# Avoiding Slowdown of Main Thread

- Helper thread for a particular instance could lag behind or be **skipped**
- Main thread should not wait for helper thread

# Problem?

- Certain **shared** data, passed from the main thread to the helper thread, **allocated** on the heap
- Helper thread could be skipped!
- Potential out-of-memory issue unless proper freeing allocated heap memory

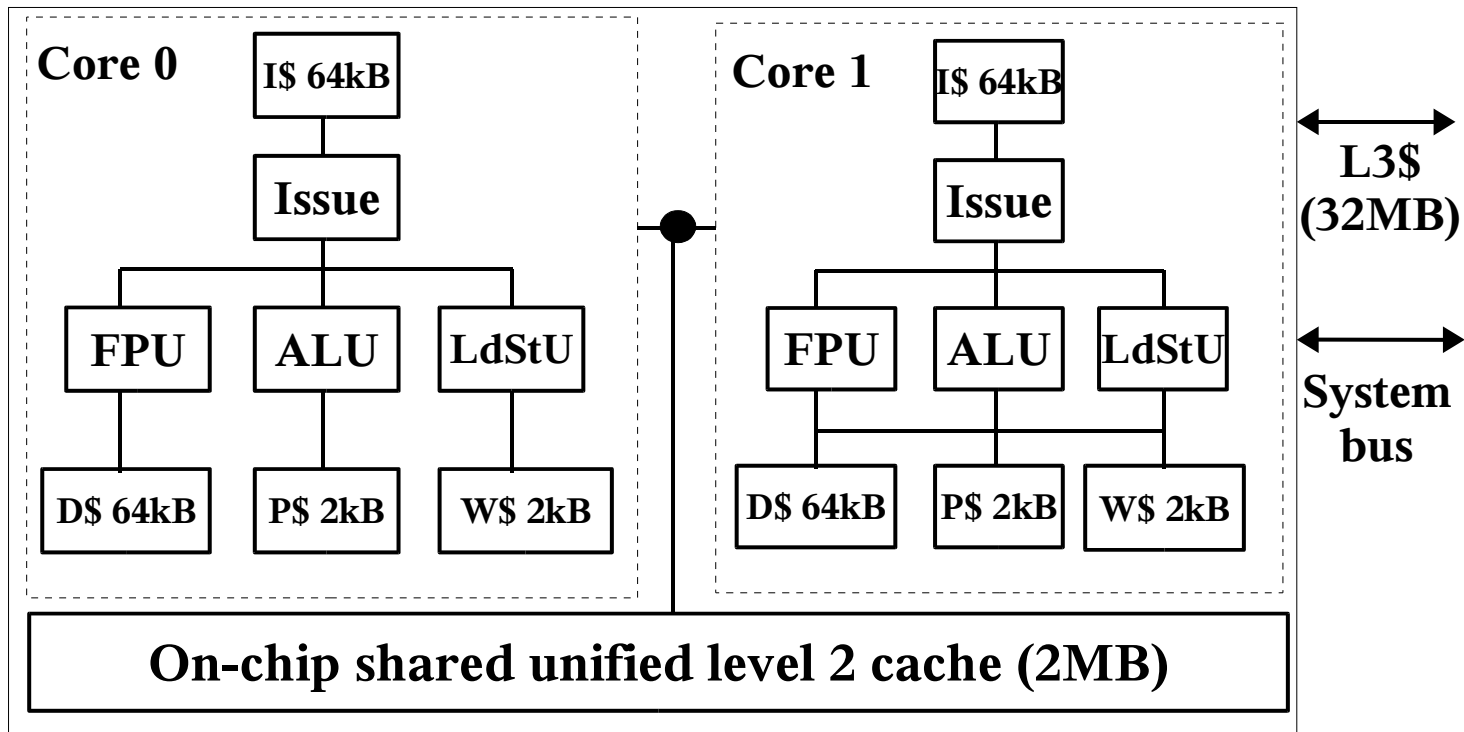
# Solution

- Free those shared data inside parallelization runtime by both main thread and helper thread
- Another communication between main thread and helper thread
- Protected by common locks

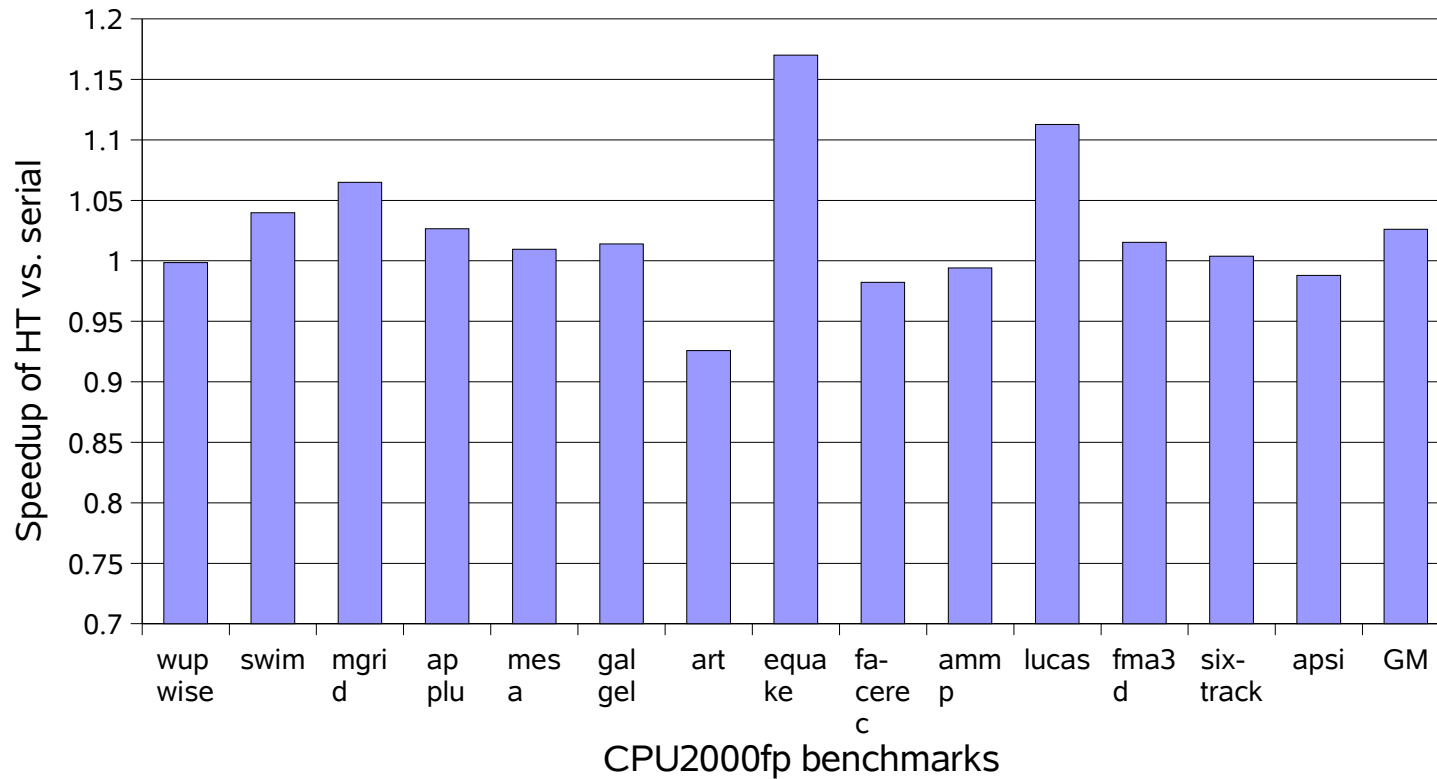
# Experiment Methodologies

- Implemented based on SUN's production compiler
- SPEC CPU2000 benchmark suite
- SUN's UltraSPARC IV+ processor
- Compared to best published serial results

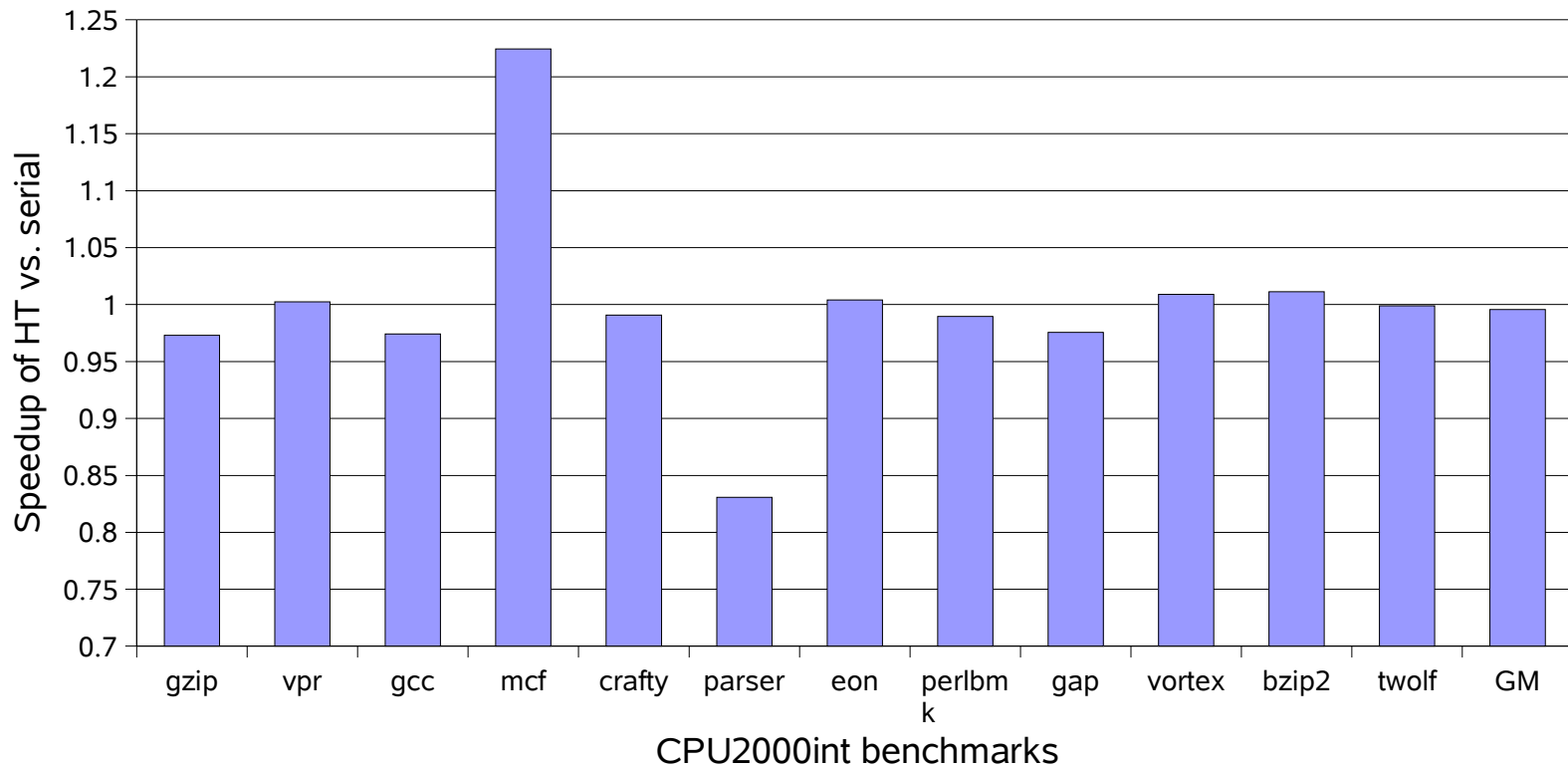
# UltraSPARC IV+ Processor



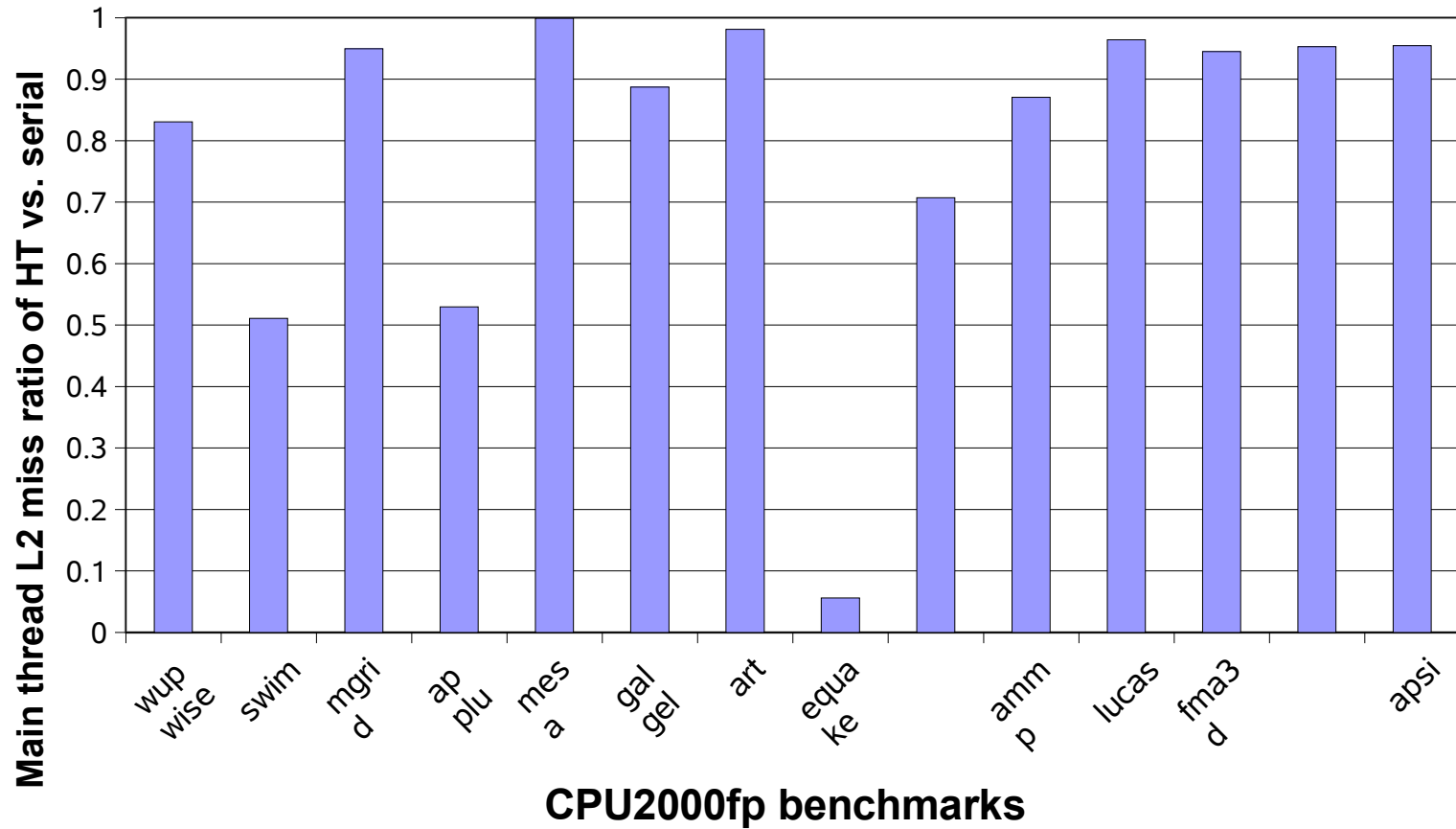
# Performance (HT vs. Serial)



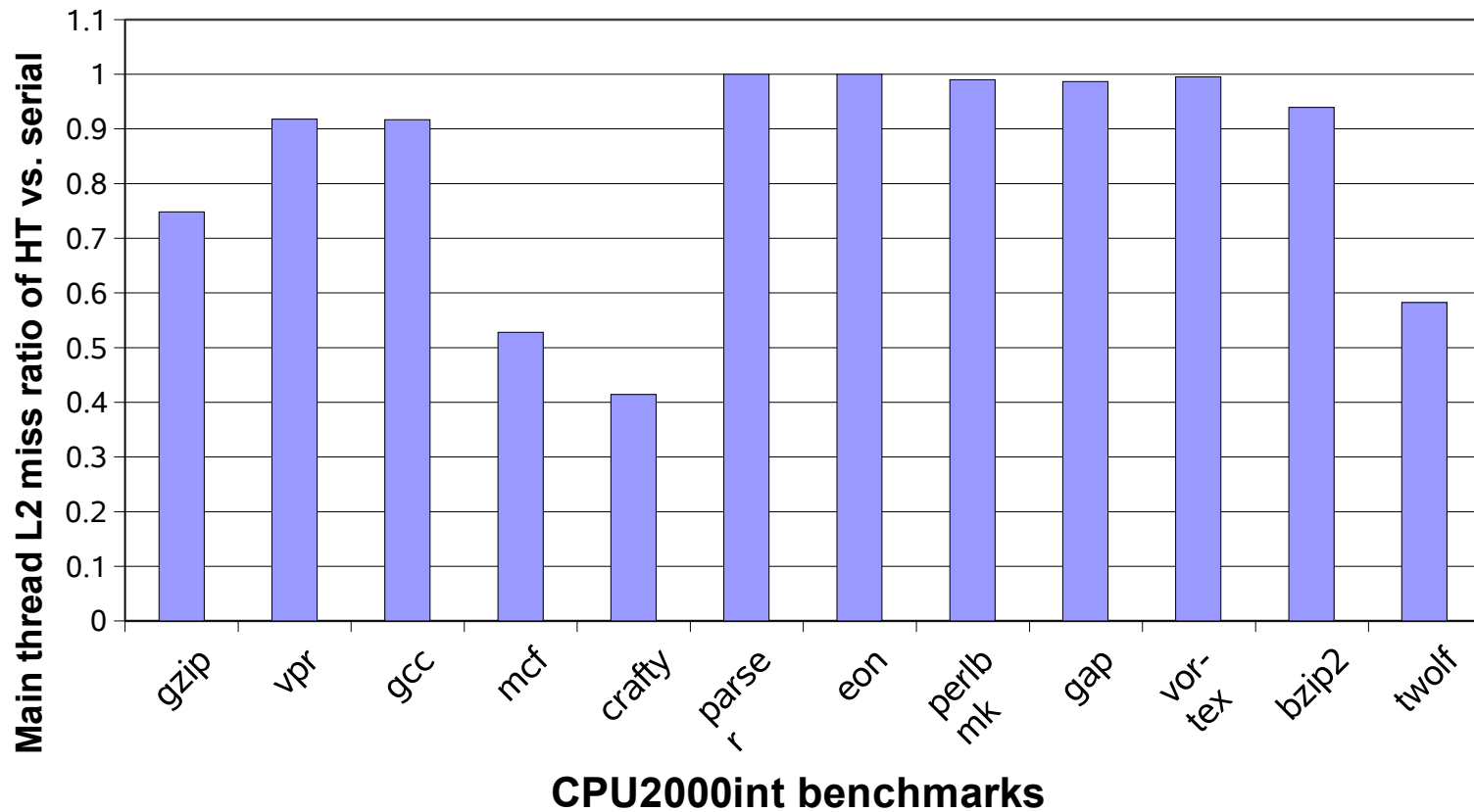
# Performance (HT vs. Serial)



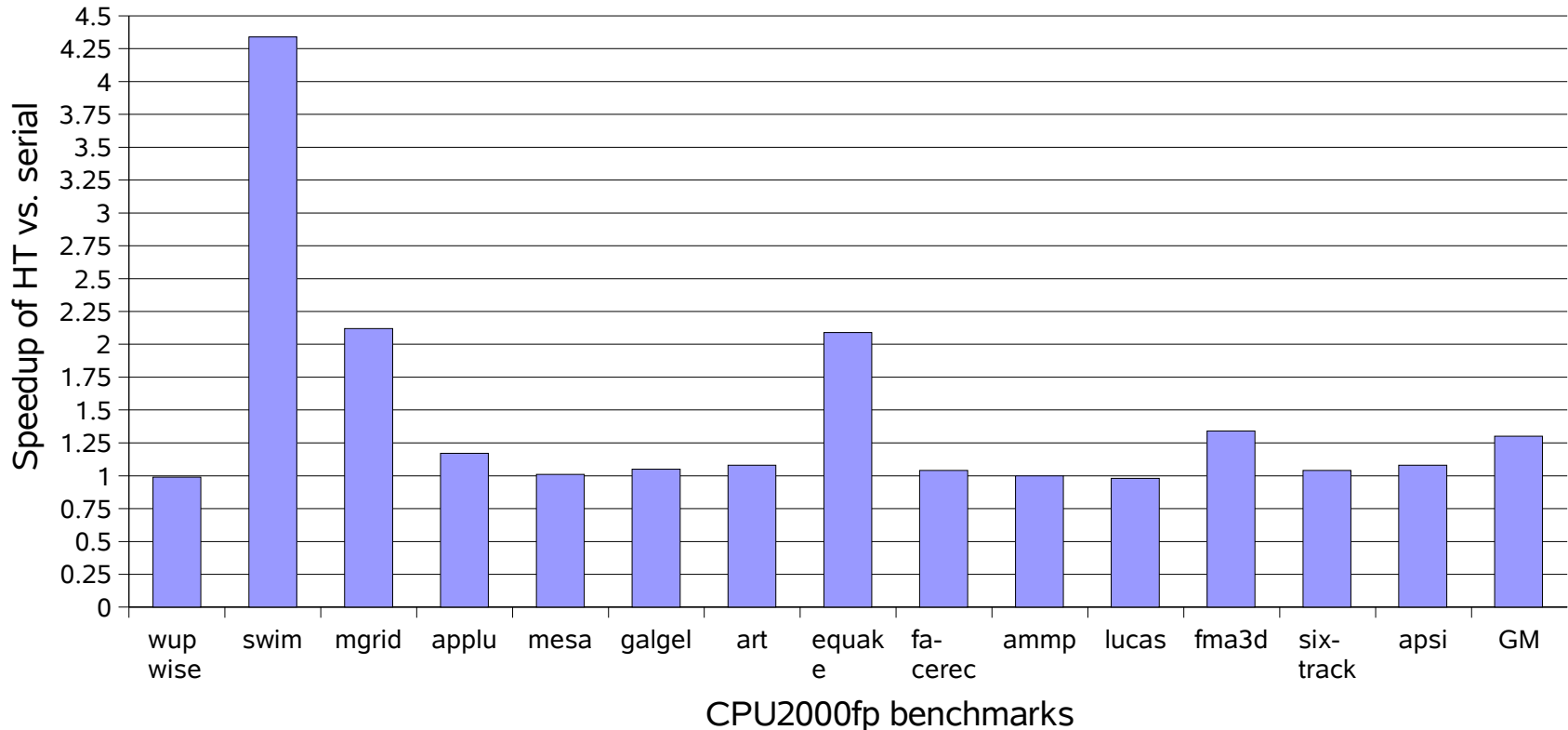
# L2 Cache Misses (HT vs. Serial)



# L2 Cache Misses (HT vs. Serial)



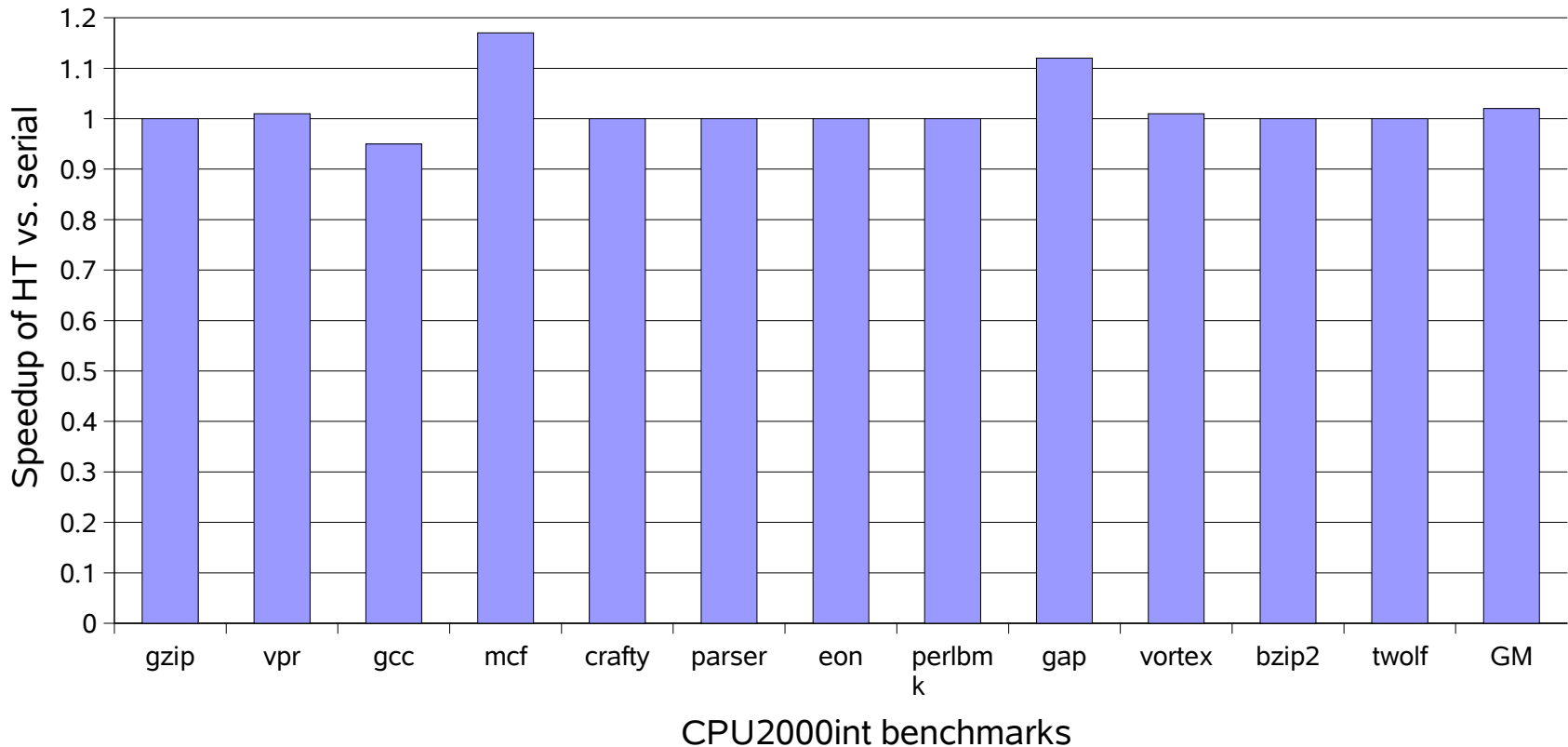
# Performance (HT vs. Serial)



**HT: no interleaved prefetching in main thread**

**Serial: no interleaved prefetching**

# Performance (HT vs. Serial)



**HT:** no interleaved prefetching in main thread  
**Serial:** no interleaved prefetching

# Related Works

- Source-level helper threading (pre-execution) code generation ([ASPLOS02, TOCS04] by Kim et al.)
- Post-binary helper threading code generation ([PLDI02] by Liao et al., [ISCA01] by Luk, ...)
- Runtime helper threading code generation ([ICS01] by Moshovos et al.)

# Conclusions

- Helper threading can deliver perf improvement
- To further improve perf: cache profiling, better runtime, hardware support, various heuristic improvement, ...