# Characterizing and Improving the Performance of Intel Threading Building Blocks

**Gilberto Contreras, Margaret Martonosi**
**Princeton University**

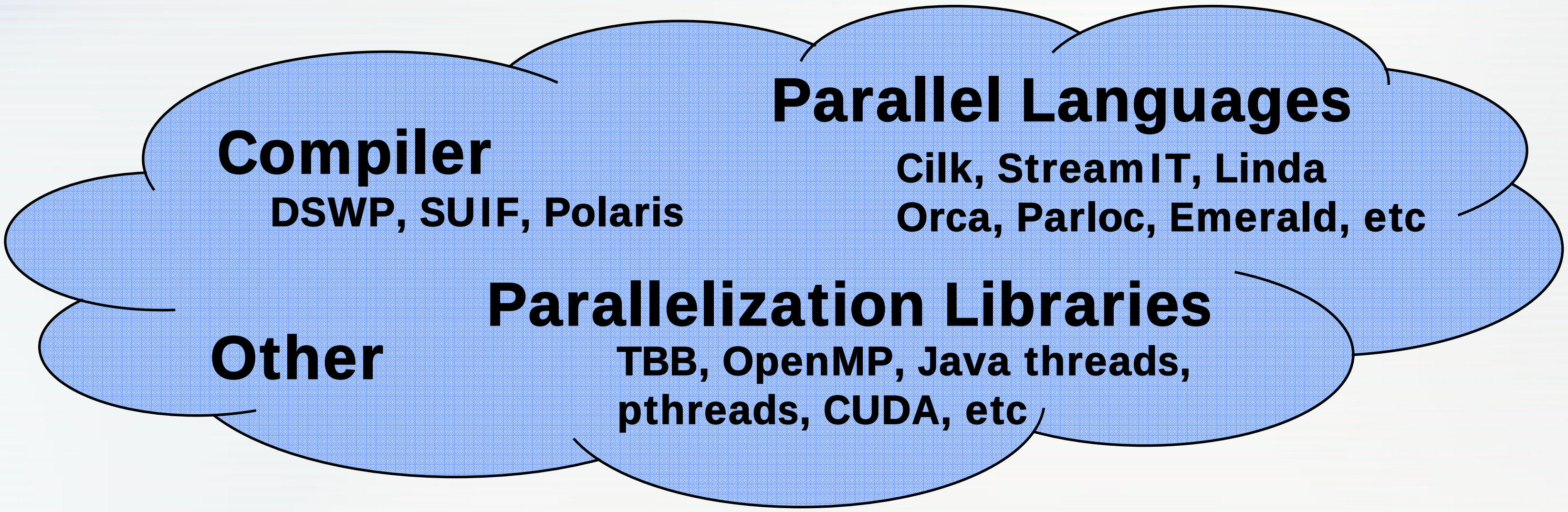
IISWC'08

# Motivation

- Chip Multiprocessors are the new computing platform.
  - 2 cores, 4 cores, 8 cores… Are we ready?
- Why is parallelism so challenging?

> - Identify parallelism
> - Annotation/extraction of parallelism
> - Mapping to cores
>
> - Respond to:
>   - OS effects
>   - Thermal emergencies
>   - Variability trends
>   - Reliability issues

# How is Parallelism Annotated/Extracted




This work answers the following questions:

- **What are some of the major sources of overheads?**
- **How do they impact overall parallelism performance?**
- **How can we improve parallelism performance?**

# Our work focus

- **This talk will focus on the Intel Threading Building Blocks (TBB)**
  - **Task-based parallelization library for C++ applications**
  - **Support a wide range of parallelism types**
  - **Utilizes task stealing for load balancing**

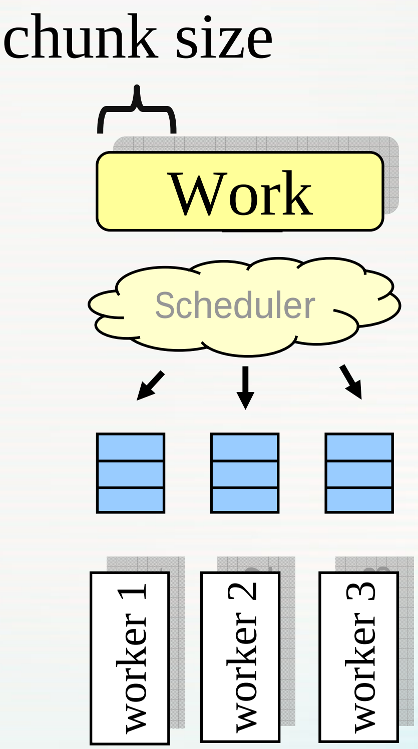
Methodology is applicable to other parallelism management approaches

# Presentation Outline

- **Description of TBB**
  - **Programming example**
  - **Task management in TBB**
- **Characterization Methodology**
  - **Measuring basic operations using simulation and real-system measurements**
  - **TBB overheads in PARSEC benchmarks**
  - **Performance of Task Stealing**
- **Improving TBB**
  - **Occupancy-based task stealing**
- **Summary and Conclusions**

# Annotation and Management

```
for (i=k+1; i<size; i++) {
  L[i][k] = M[i][k] / M[k][k];
  for(j=i+1; j<size; j++)
    M[i][j] = M[i][j] -
                 L[i][k]*M[k][j];
}
```




Runtime procedure:

spawn()
acquire_queue()
get_task()
spawn()
spawn()
steal()
acquire_queue()
get_task()
spawn()
steal()
acquire_queue()
get_task()

# Reducing TBB Library Overhead?

- **Understand Overheads**
  - **Creating tasks**
    - `spawn()`
  - **Assigning tasks to worker threads**
    - `get_task()`
    - `queue_acquire()`
    - `wait_for_all()`
  - **Stealing or rebalancing parallelism**
    - `steal()`

- **Improve parallelism reorganization policies**
  - **Employ smart redistribution policies**
  - **Make this as fast and as efficient as possible**

# Methodology

**Benchmarks**
- PARSEC
- Microbenchmarks

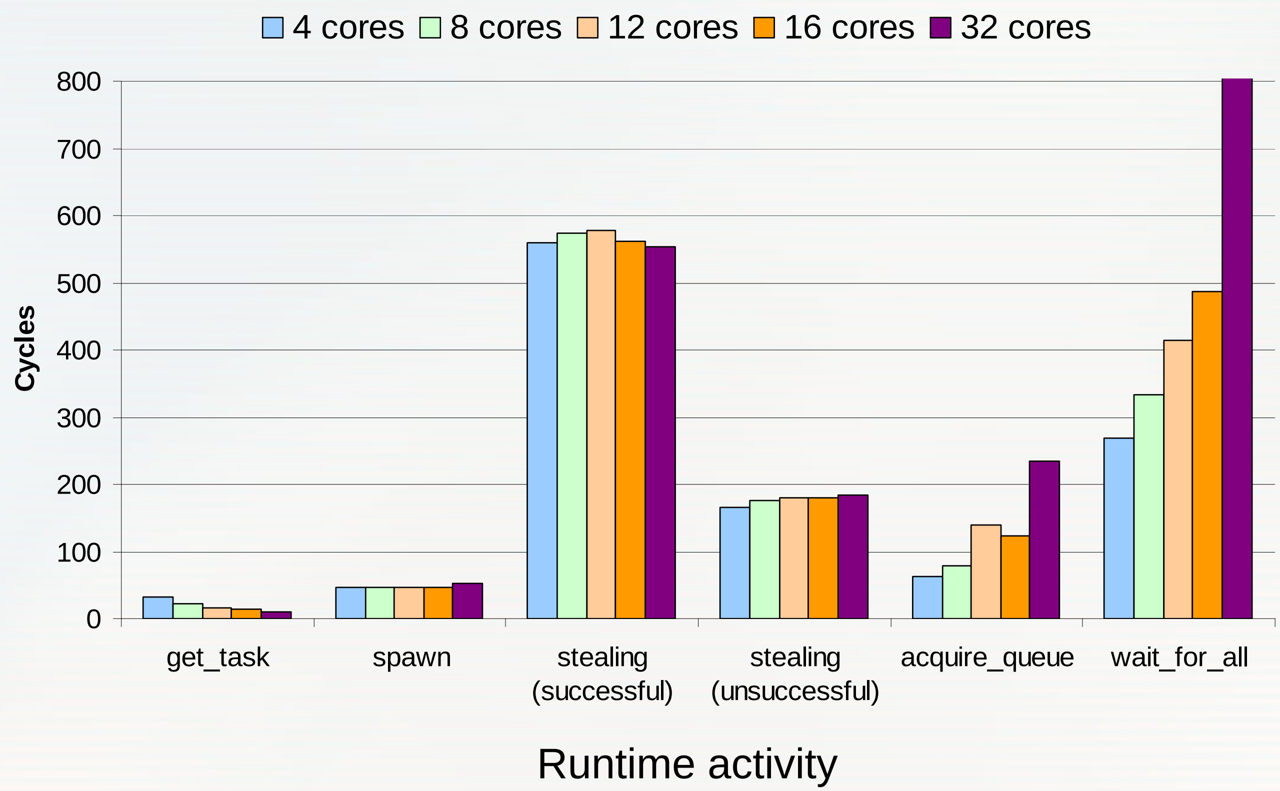
**Intel Threading Building Blocks (TBB)**
- Open source 2.0 version

**Real CMP System**
- 4-core AMD system (2 processors)
- 4GB RAM
- Linux 2.6
- *Oprofile* is used for performance counter measurements

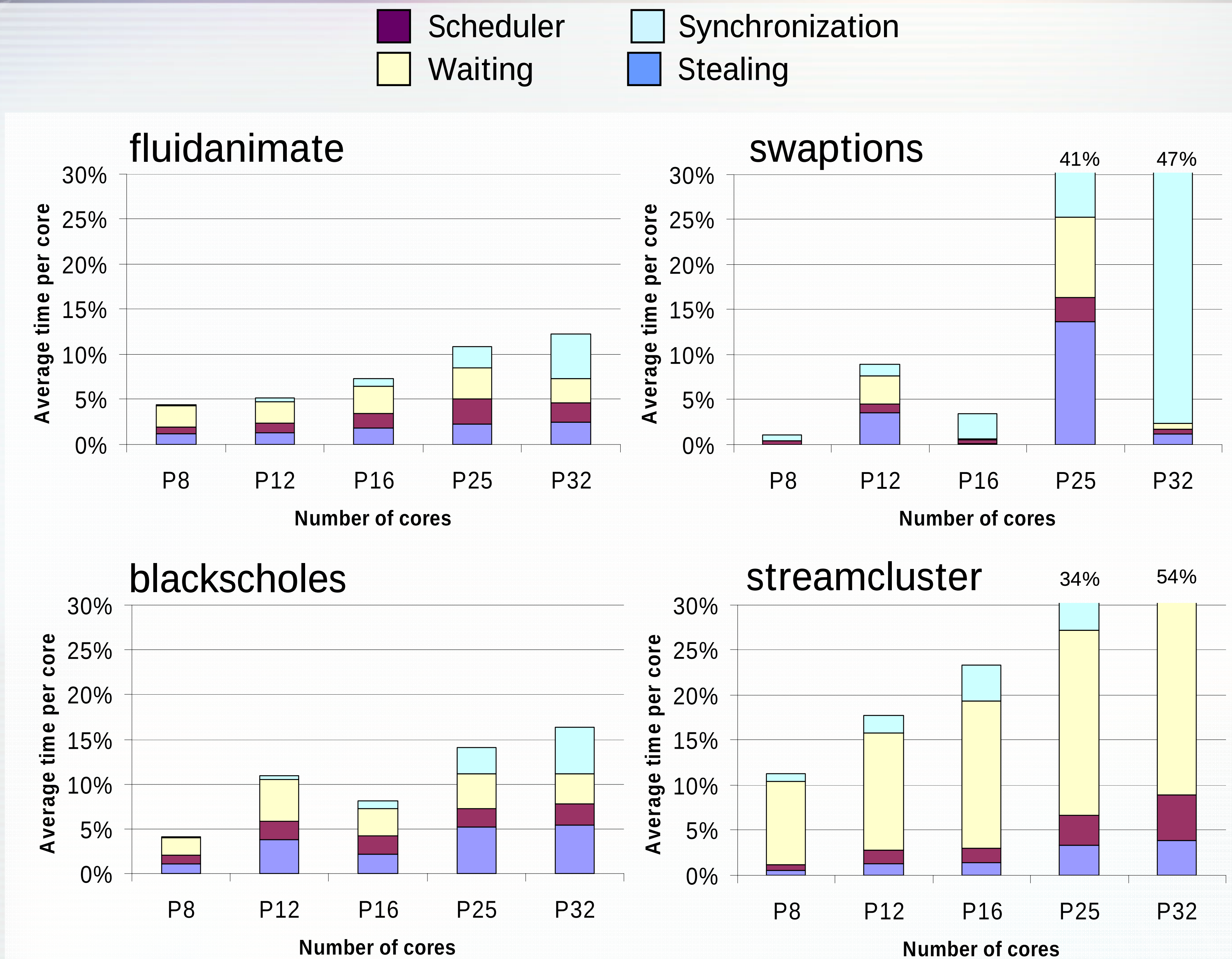
**Cycle-accurate CMP simulator**
- 2-issue, in-order cores
- 32KB D$ (coherent), 32K I$
- 8MB shared L2 cache
- MSI directory-based coherence protocol
- Mesh network, 32b BW/port/cycle

# Cost of Parallelism Management

Simulation Results (4-32 cores)

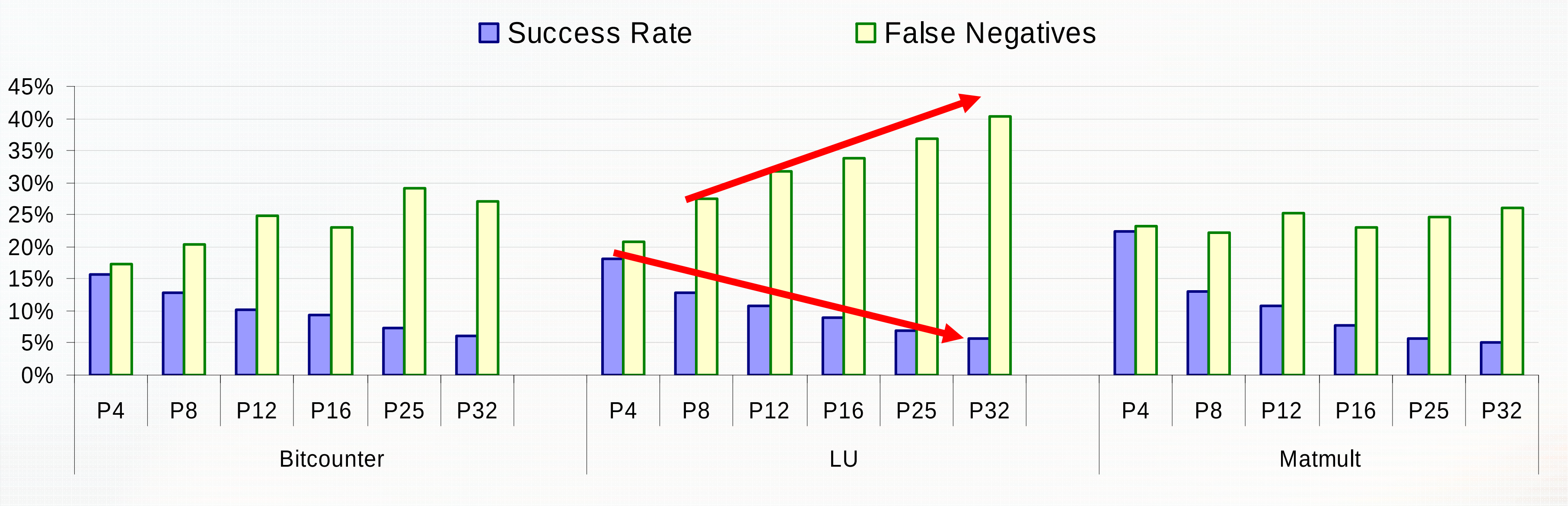


# TBB Overheads: PARSEC



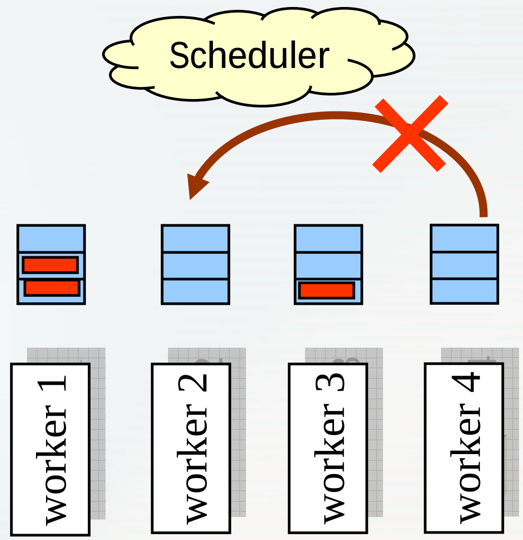
# Improving Stealing

- **TBB utilizes random stealing as its victim selection policy**

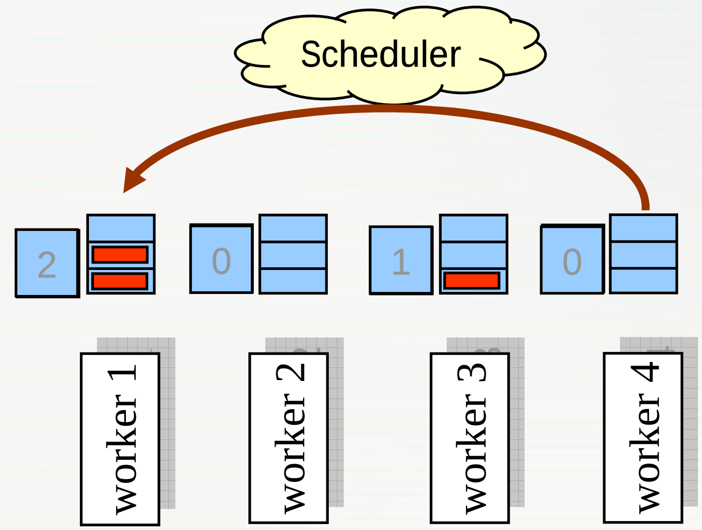


# Occupancy-based Stealing

Random Stealing




Occupancy-based stealing




2

- Random stealing:
  - Random number
  - Stealing

- Occupancy stealing:
  - Scanning
  - Stealing

# Performance of Occupancy-based Stealing

| | Occupancy-Based | | | 1-cycle scan Normal stealing | | | 1-cycle scan 1-cycle stealing | | |
|---|---|---|---|---|---|---|---|---|---|
| | P16 | P25 | P32 | P16 | P25 | P32 | P16 | P25 | P32 |
| **Bitcounter** | **2.5%** | **2.5%** | **2.7%** | **2.4%** | **2.8%** | **3.7%** | **4.7%** | **6.9%** | **7.8%** |
| **LU** | **10%** | **4.1%** | **9.7%** | **10.2%** | **4.6%** | **8.0%** | **16%** | **10.4%** | **20.6%** |
| **Matmult** | **9.5%** | **6%** | **19%** | **9.8%** | **7.0%** | **21.1%** | **10.8%** | **9.8%** | **28.7%** |

- Smarter selection policies are desired
- High potential in overhead reduction

# Conclusions

- Increasing usage of TBB makes it a prime candidate for in-depth characterization
- Parallelization libraries help, but tend to exhibit high (dynamic) overheads (>40% at 32 cores)
- Understanding software overheads is the first step in creating high-performance parallel systems
- We have presented a detailed characterization of the Intel Threading building Blocks and implemented *occupancy-based stealing* (19% performance over random stealing).

# Thanks!

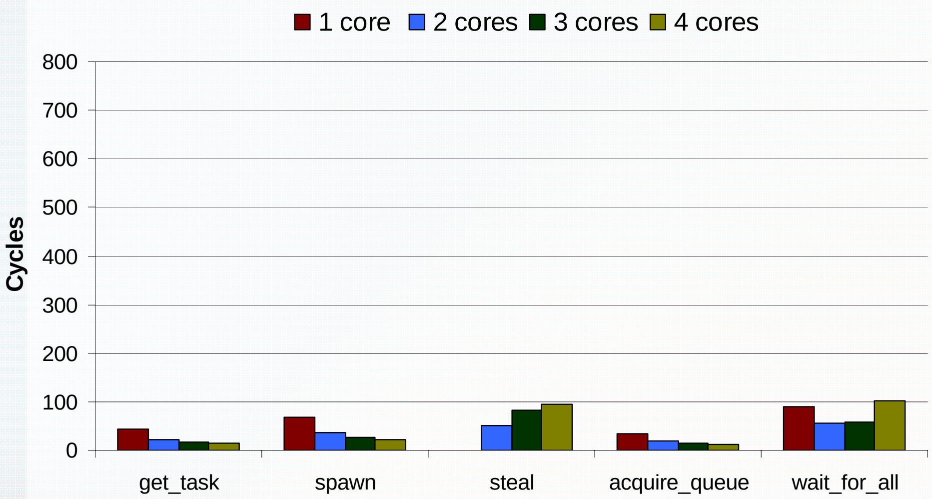
# Summary

- **Programmers require tools that allows them to take (fast) advantage of increasing core counts.**
- **Parallelization libraries help, but tend to exhibit high (dynamic) overheads (>40% at 32 cores)**
- **Understanding software overheads is the first step in creating high-performance parallel systems**
- **We have presented a detailed characterization of the Intel Threading building Blocks and implemented *occupancy-based stealing* (19% performance over random stealing)*.***

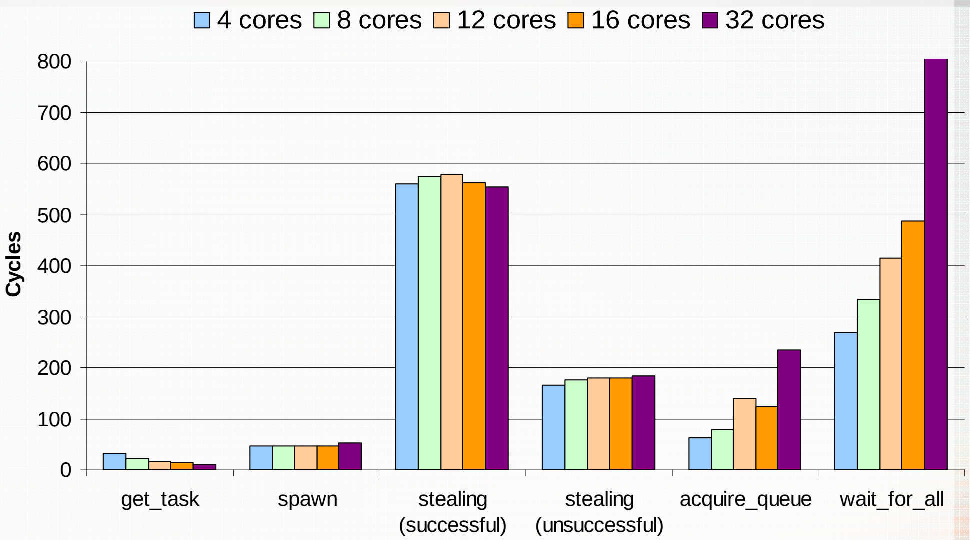
# Cost of Parallelism Management

- 4 core, 1.8GHz AMD system
- *Oprofile* configured to measure CPU_CLK_UNHALTED

- 1 to 32 core CMP simulator
- 2-issue, in-order cores
- Shared L2




Runtime activity




Runtime activity

Our goals:

1) Reduce per-event overheads
2) Improve rebalancing

# Static versus Dynamic Management