Performance Impact of Multithreaded Java Server Applications

Yue Luo, Lizy K. John
Laboratory of Computer Architecture
ECE Department
University of Texas at Austin

Outline

- **Motivation**
- **VolanoMark Benchmark**
- **Methodology**
- **Results**
- **Conclusion**
- **Further Work Needed**

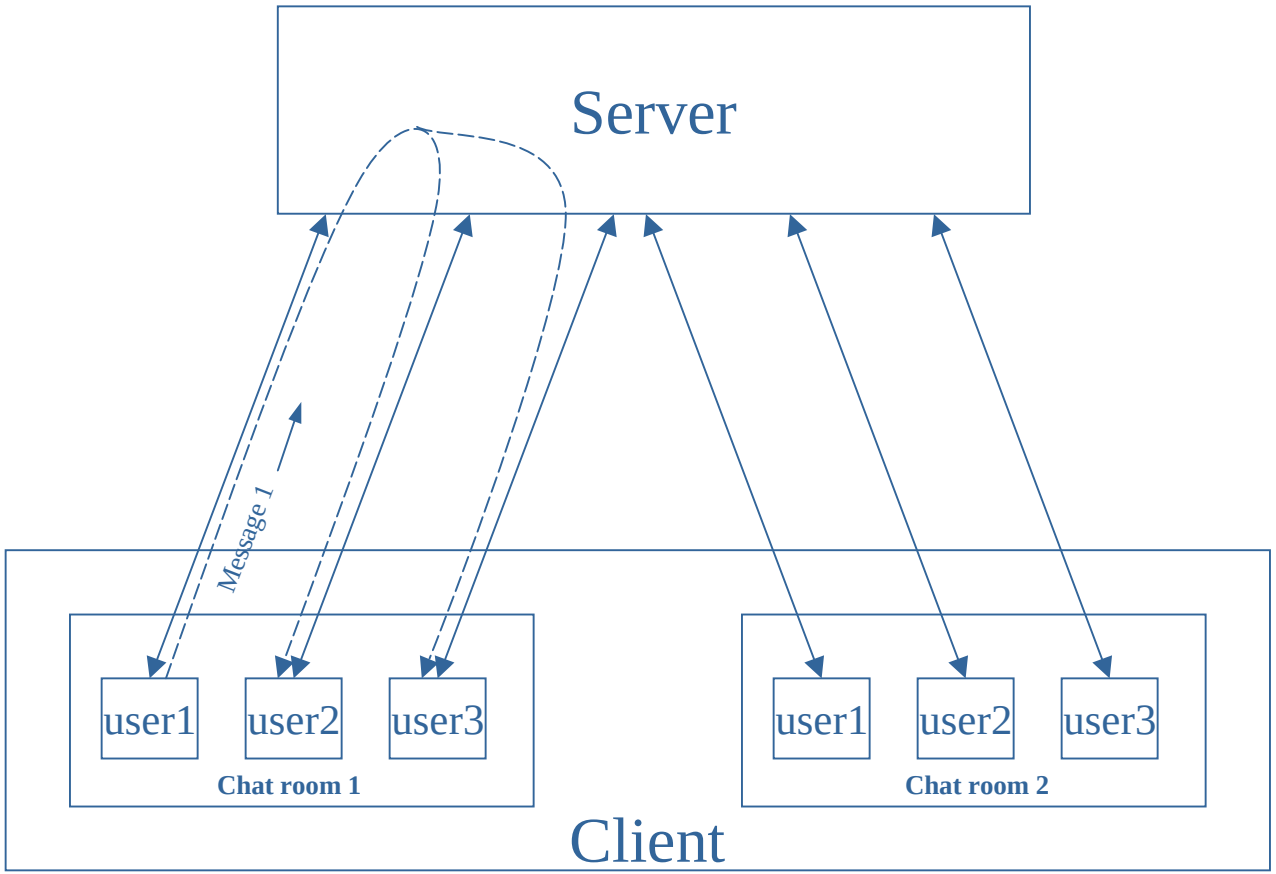
Motivation

- **Performance under the presence of a large number of threads is crucial for a commercial Java server.**
  - **Java applications are shifting from client-side to server-side.**
  - **Server needs to support multiple simultaneous client connections.**
  - **No select() or poll() or asynchronous I/O in Java**
  - **Current Java programming paradigm: one thread for one connection.**

VolanoMark Benchmark

- **VolanoMark is a 100% Pure Java server benchmark characterized by long-lasting network connections and high thread counts.**
  - **Based on real commercial software.**
  - **Server benchmark.**
  - **Long-lasting network connections and high thread counts.**
  - **Two threads for each client connection.**

VolanoMark Benchmark



Methodology

- **Performance counters used to study OS and user activity on Pentium III system.**
- **Monitoring Tools -- Pmon**
  - **Developed in our lab. Better controlled.**
  - **Device driver to read performance counters**
  - **Low overhead**

Platform Parameters

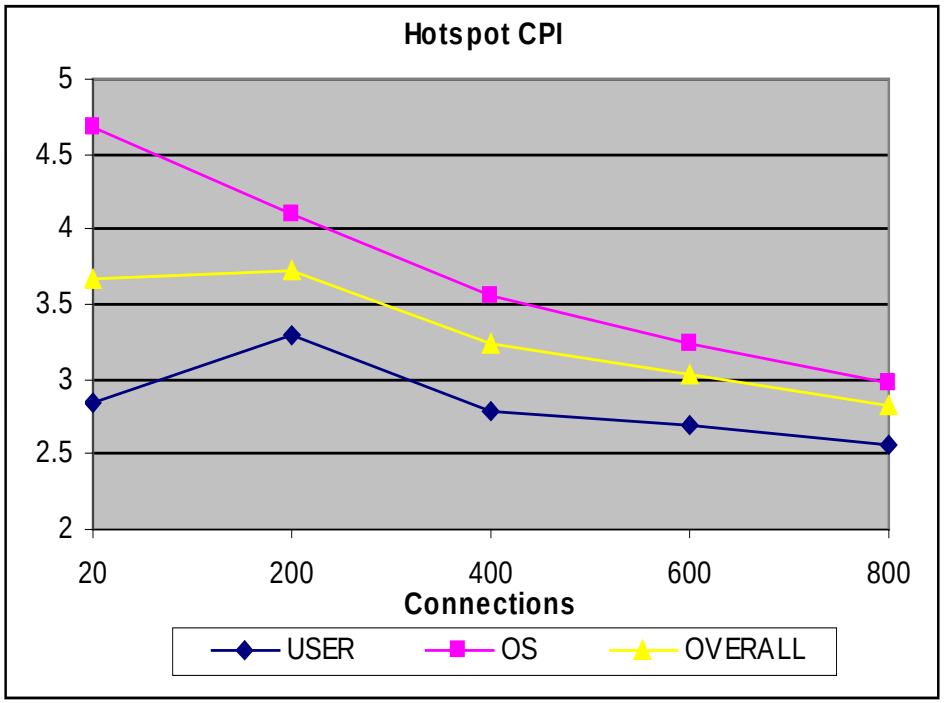
- **Hardware**
  - **Uni-processor**
  - **CPU Frequency: 500MHz**
  - **L1 I Cache: 16KB, 4-way, 32 Byte/Line, LRU**
  - **L2 Cache: 512KB, 4-way, 32 Byte/Line, Non-blocking**
  - **Main Memory: 1GB**
- **Software**
  - **Windows NT Workstation 4.0**
  - **Sun JDK1.3.0-C with HotSpot server (build 2.0fcs-E, mixed mode)**

Monitoring Issues

- **Synchronize measurements with client connections to skip starting and shutdown process**
  - **Add wrapper to the client.**
  - **The wrapper starts an extra connection immediately before starting the client to trigger measurement.**
- **Avoid counter overflow**
  - **Counting interval: 3sec**

Results

- **Decreasing CPI**

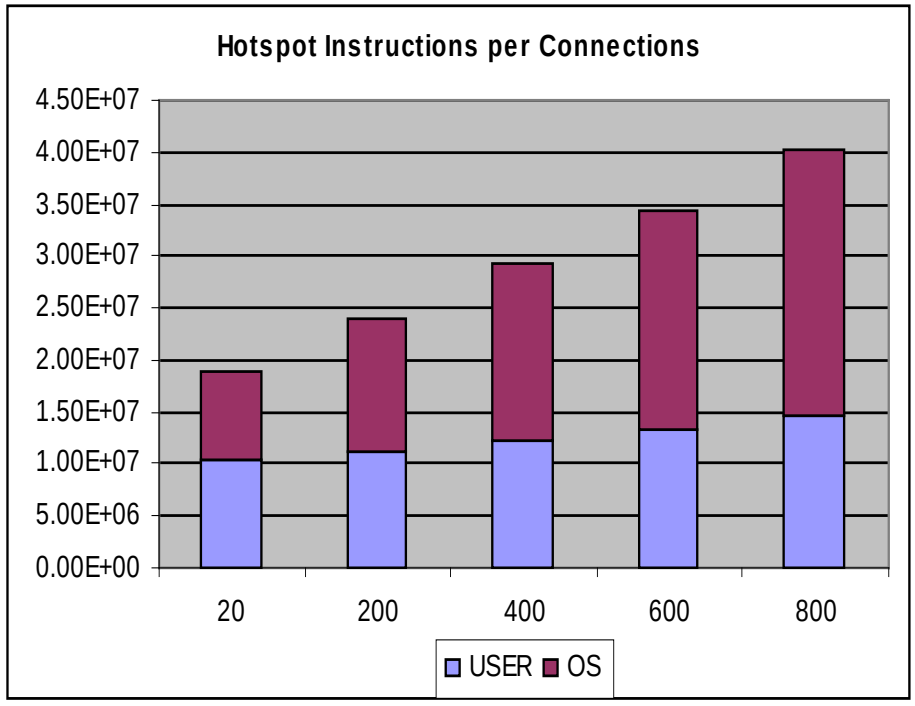


- OS has larger CPI
- OS CPI decreases significantly
- User CPI sees small fluctuation.

Results

- **More instructions executed!**

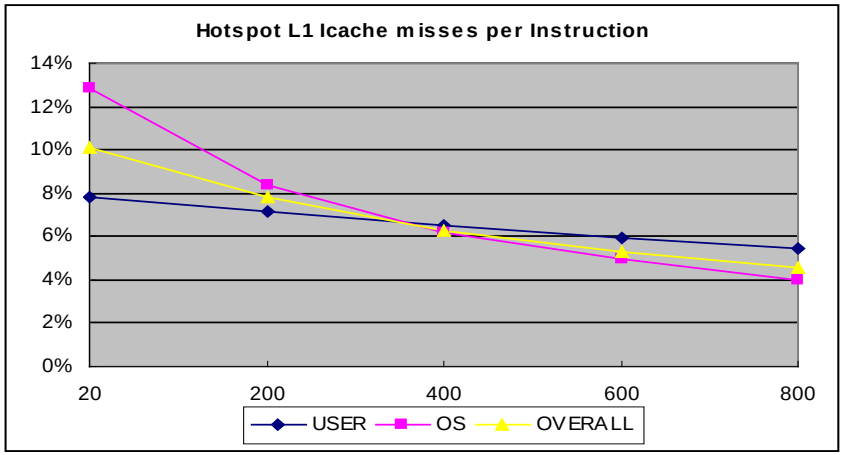
Regardless of the number of connections, each thread basically does the same thing. Therefore instructions for each connection should remain the same.

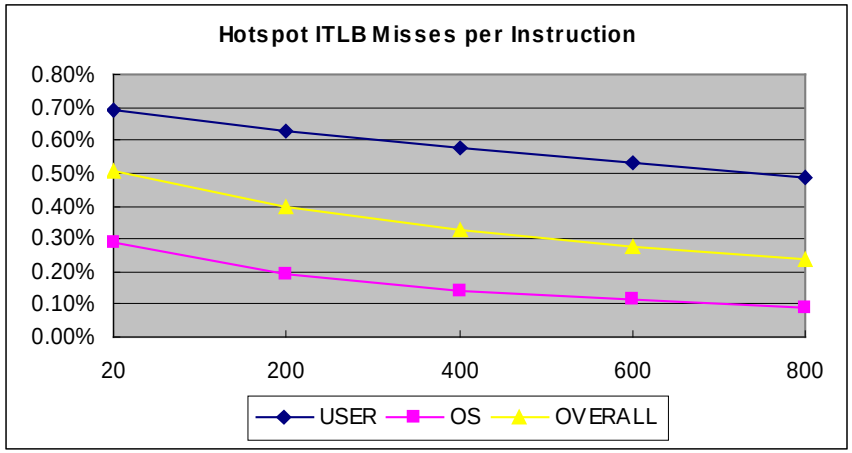


- OS part increases significantly
- User part increases slightly
- Even more execution time is in OS mode due to the larger CPI in OS.
- One guess: overhead in connection and thread management; some OS algorithm with non-linear complexity (e.g. O(N*logN))

Results

- **Decreasing L1 I-Cache miss ratio**

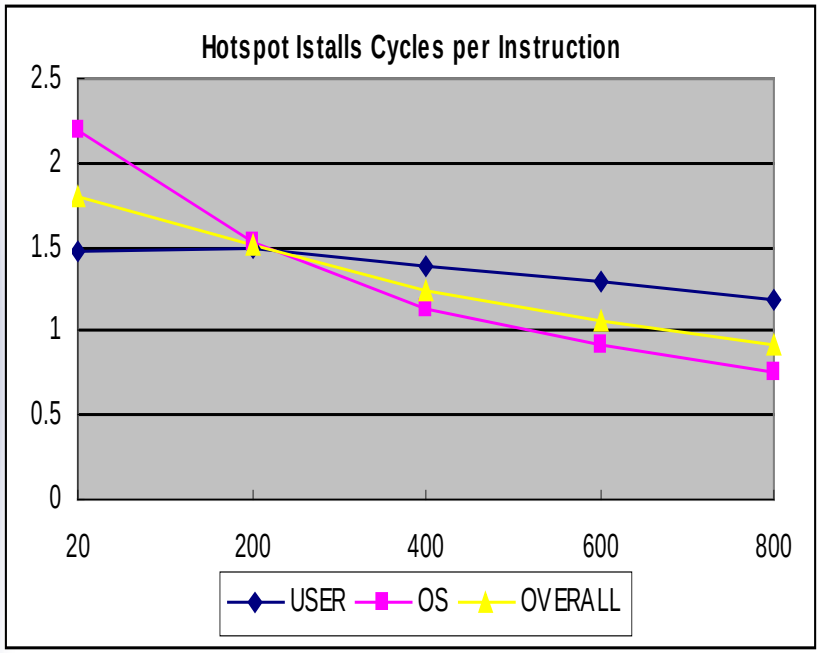




- Beneficial interference between threads: They share program codes.
- The more threads we have, the more likely we context switch to another thread that is executing the same part of the program thus codes in I cache and entries in ITLB are reused.

Results

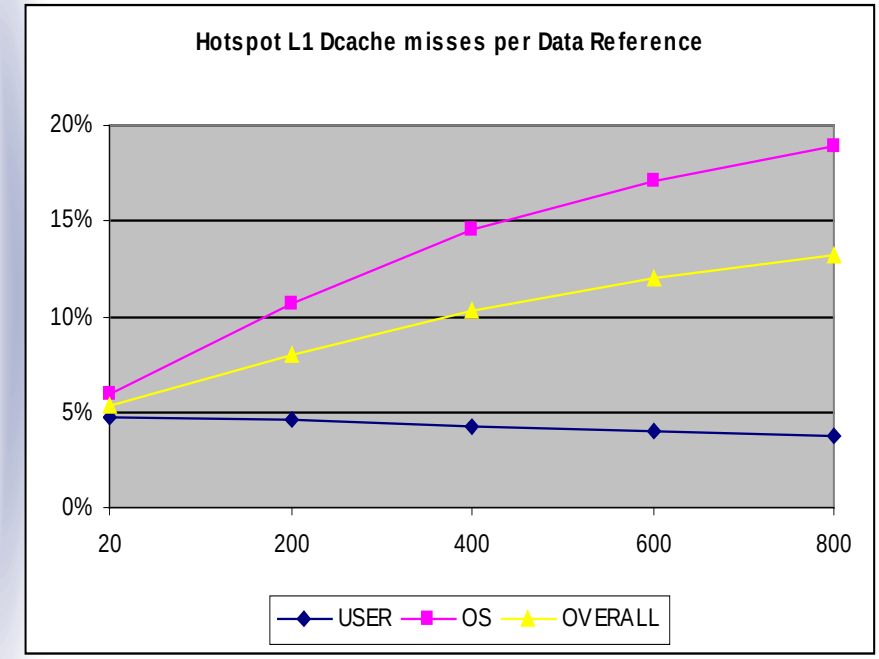
- **Decreasing I stalls per instruction**



- As the result of decreasing I-cache miss ratio and ITLB miss ratio, instruction fetching stalls are lowered for both OS part and user part.

Results

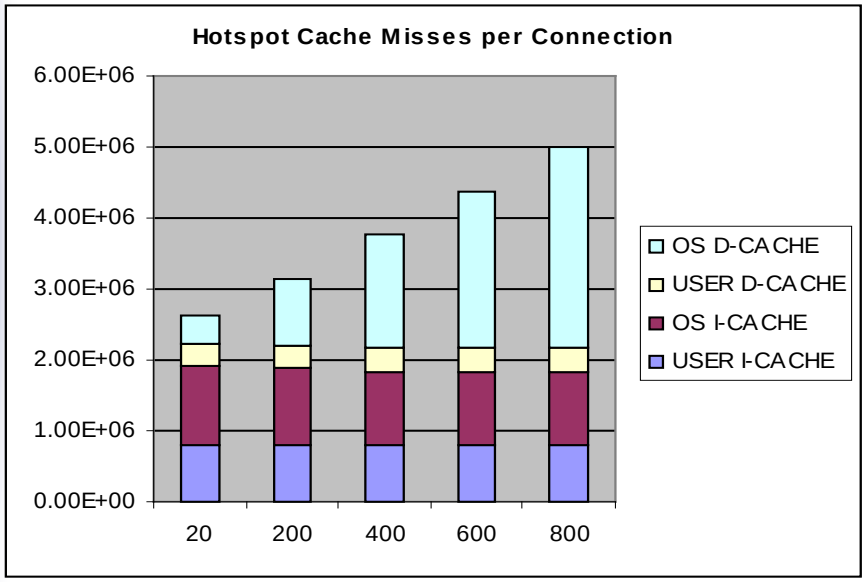
- **Increasing L1 D cache miss ratio**



- OS: Huge increase
  - –More thread data, larger data footprint
  - –More context switches
- User: Slight decrease

Results

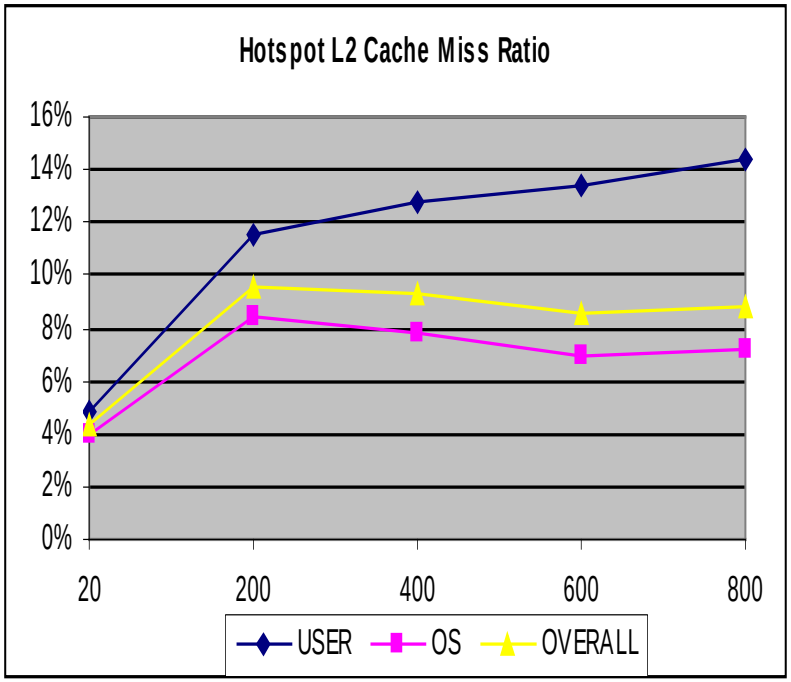
- **Significant OS L1 D-cache Misses**



- With more connections, OS are doing more and incurring more data misses
  - –Send & receive network packets
  - –Threads scheduling & synchronization

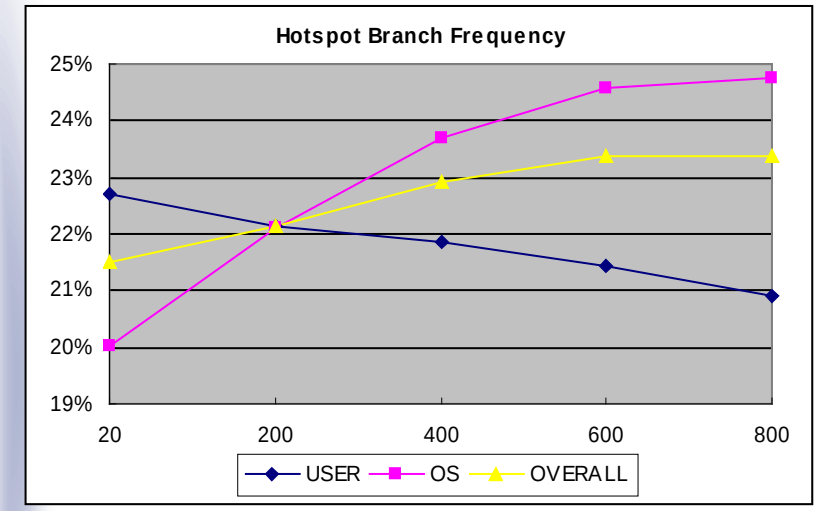
Results

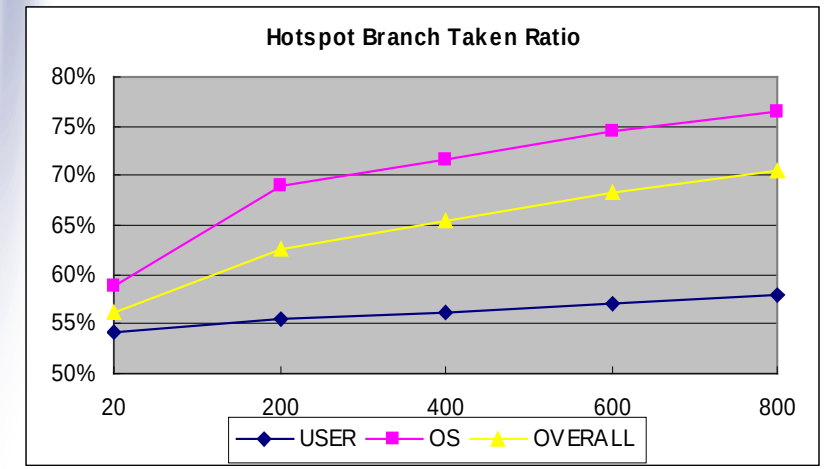
- **L2 Cache miss ratio**



Results

- **Branches**

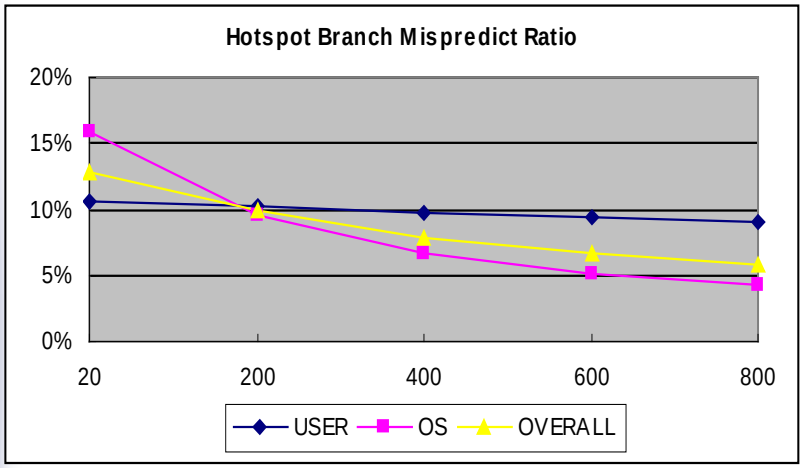




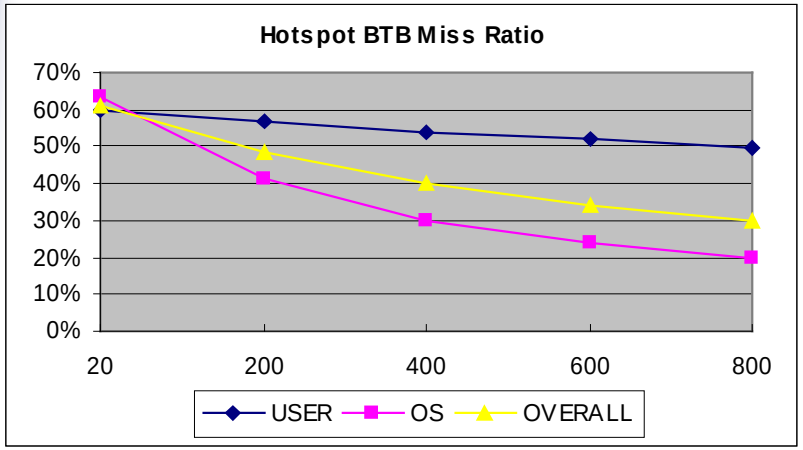
- More branches in OS code
- More branches are taken
- May be due to more loops in OS code

Results

- **More accurate branch predictions**



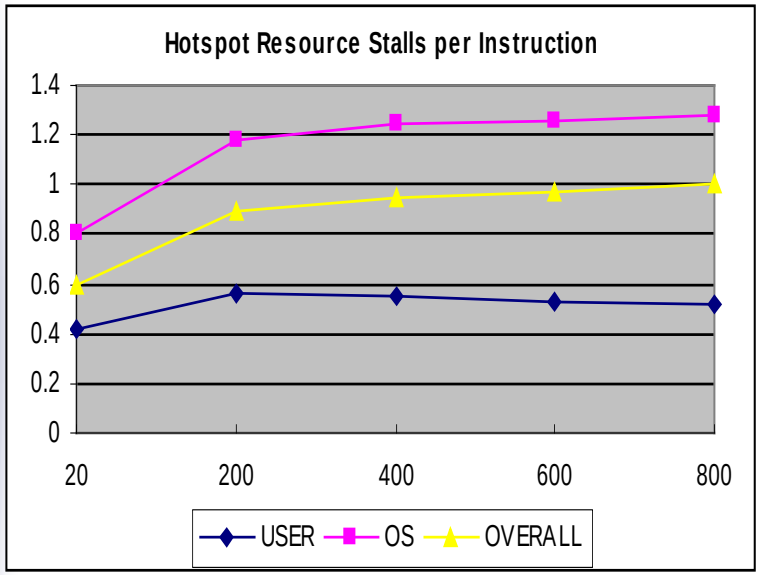
- Branches in loops are easier to predict so we have more accurate branch predictions



- Due to the beneficial code sharing among threads, BTB miss ratio decreases.

Results

- **More resource stalls**



- Lower instruction stalls and better branch prediction result in larger resource stalls
- May favor a CPU with more resources.

Conclusions

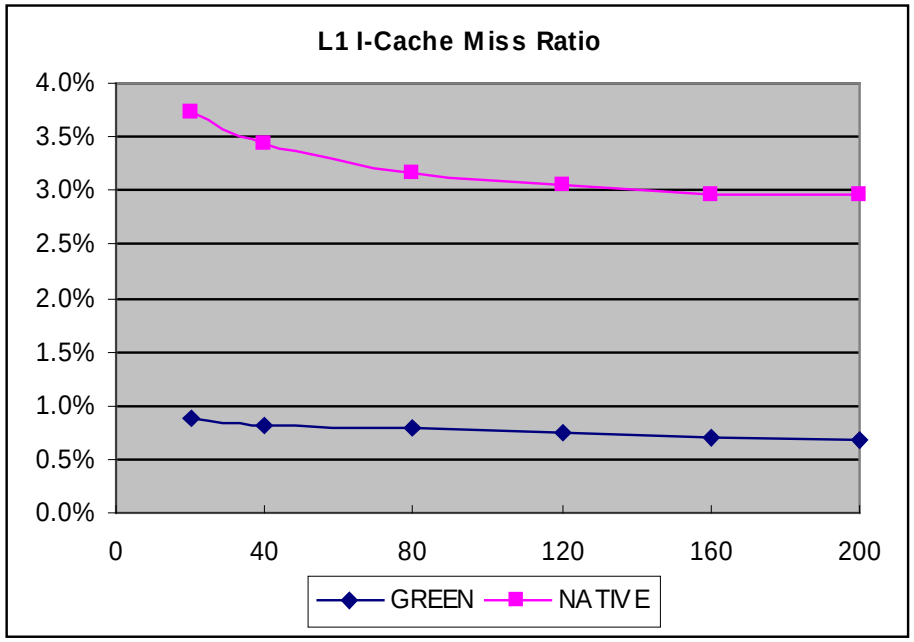
- **Multi-threading is an excellent approach to support multiple simultaneous client connections. Heavy multithreading is more crucial to Java server applications due to its lack of I/O multiplexing APIs.**
- **Thread creation and synchronization as well as network connection management are the responsibility of the operating system. With more concurrent connections, more OS activity is involved in the server execution.**
- **Threads usually share program code; thus instruction cache, ITLB and BTB will all benefit when the system context switch from one thread to another thread executing the same part of code. Multi-threading also benefits branch predictors.**
- **Each thread will incur some code and data overheads especially in operating system mode. Given enough memory resources, the nonlinearly increasing overheads are the biggest impediment to performance scalability. Further tuning of the application and operating system may alleviate this problem.**

Further work needed

- **More complex benchmark needed (eg SPEC jbb2000) to validate the results. We need to distinguish characterization of multi-threaded server applications from that of VolanoMark.**
- **Find why much more instructions are executed in OS with more connections and try to reduce them.**

Results on Sparc With Shade (backup slide)

- **Also observed decreasing L1 I-cache miss ratio**



Results on Sparc With Shade (backup slide)

- **Also observed better branch prediction**