



Turnaround or Throughput ? Measuring Application Performance

Sanjay Goil and Mukund Buddhikot
Scalable Systems Group
Sun Microsystems Inc.



Outline

- Performance definition
 - Turnaround
 - Throughput
- Throughput modes
- Application results
- Future Performance Implications
- Summary

Performance Questions

- What do we ask ?
 - Turnaround time for my program
 - Runtime alone ?
 - Queuing + runtime ?
 - Runtime = wall time or cpu time ?
 - How parallel did I go, i.e. what scalability did I see ?
 - Implications on turnaround time
 - Queue time ?
 - Runtime ?
 - Efficiency of execution, i.e. utilization
 - Is the machine dedicated to my run ?
 - Queue time and runtime implications

“Turnaround” Performance

- Runtime execution (Time units)
 - Single program on a single cpu
 - No processing contention
 - However, memory and interconnect may be shared
 - Single program on a single thread of execution
 - Multiple threads can execute on a “cpu”
 - No processing contention on single hardware thread
 - Can share memory hierarchy upto the L2 caches
 - Hard to isolate resources
 - Single program, **now multi-threaded**, running on
 - Traditional SMP – Many single cpu + custom interconnect
 - Clusters – Many single CPU + COTS interconnect
 - H/W multi-threaded chips + some interconnect

Focus on Turnaround

- Single CPU performance is continually improving
- Large applications routinely use more than one cpu
 - Memory and bandwidth do not keep up
- Multi-threading is a solution
 - OpenMP, MPI, Pthreads etc.
 - Sharing of memory and network bandwidth
 - Still need dedicated resources per program run for best performance
 - Negative effects of over subscription ?
 - Capacity on demand for parallel resources ?
 - Do I really need to know how many cpu's are free ?

What about Throughput ?

- As opposed to turnaround
 - Number of users submit single cpu programs
 - A single user submits many single cpu programs
 - Number of users submit multi-threaded programs
 - A single user submits many multi-threaded programs
 - Effects of sharing CPU/caches, memory and network bandwidth
- **In addition to turnaround**
 - Minimize effects of sharing
 - Balanced design
 - Minimize effects of over subscription
 - Better thread scheduling and synchronization
 - Example: Do idle threads spin or sleep ?

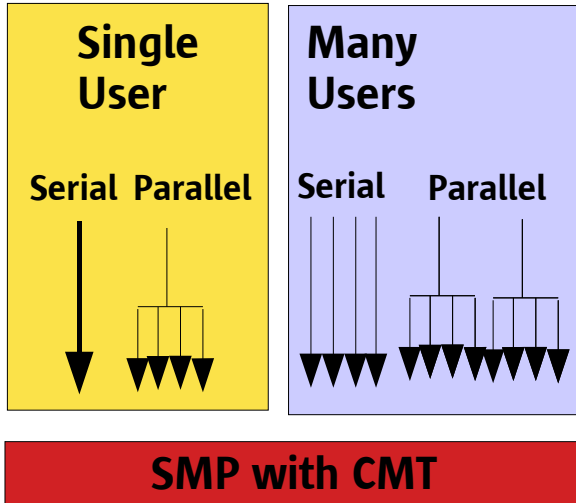
Throughput Modes

- Single user submitting many single cpu jobs
 - Design of Experiments exploring combinations
- Many users submitting single cpu jobs
 - Best single cpu turnaround for all users
- Single user submitting parallel job
 - Best turnaround time for this single job
- Many users submitting parallel jobs
 - Best turnaround for all users
 - CMT is a boon

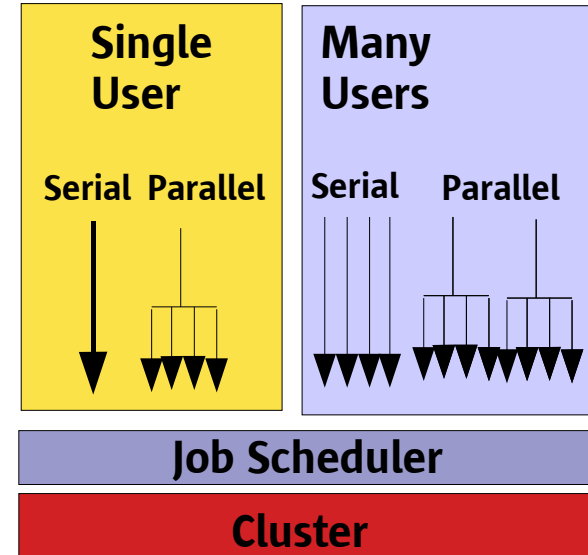
Chip multi-threading futures

- Multi-core
 - Minimal on-chip sharing, share memory bandwidth
- Vertical, Simultaneous Multithreading
 - Mechanisms to share chip resources
 - Registers, state, context switch, synchronization, atomic
 - Grow bandwidth to memory and network
 - Reduced footprint, costs; Increased RAS
- Architected for Throughput
 - **In addition to turnaround ?**
 - For full subscription and oversubscription
 - Extract TLP and ILP within that
 - Exploiting MLP by overlapping memory accesses
 - Focus on utilization

Throughput Computing



**Capability
(SMP => CMT)
and Capacity**



**Capacity
(COTS cpus + interconnect)**



Results

- Some results showing the effects of full and oversubscription on current systems
 - What do we need to architect to improve these ?
- Two applications
 - Eclipse (Energy Markets)
 - BLAST (Life Sciences Markets)
- Two benchmarks
 - SPECCPU
 - SPECOMP
- Two architectures
 - Vertical scaling: SF6800/E6900
 - Horizontal scaling: V20z cluster

Oversubscription modes

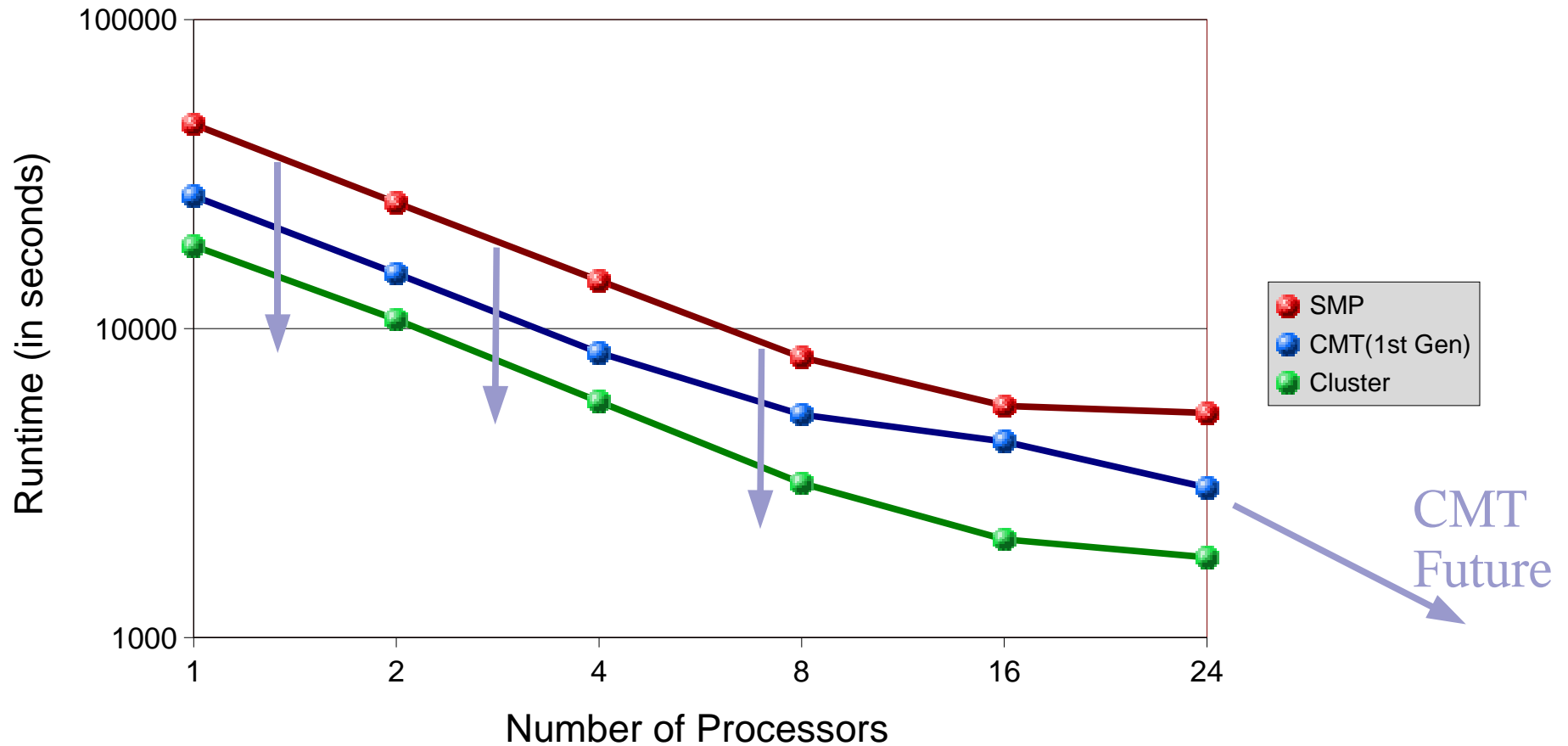
Used in the results that follow

- 1:1 Number of software threads *equals* Number of execution units (cores, threads, thread-execution engines)
- 2:1 Number of software threads is *twice* the Number of execution units
- 4:1 Number of software threads is *four times* the Number of execution units

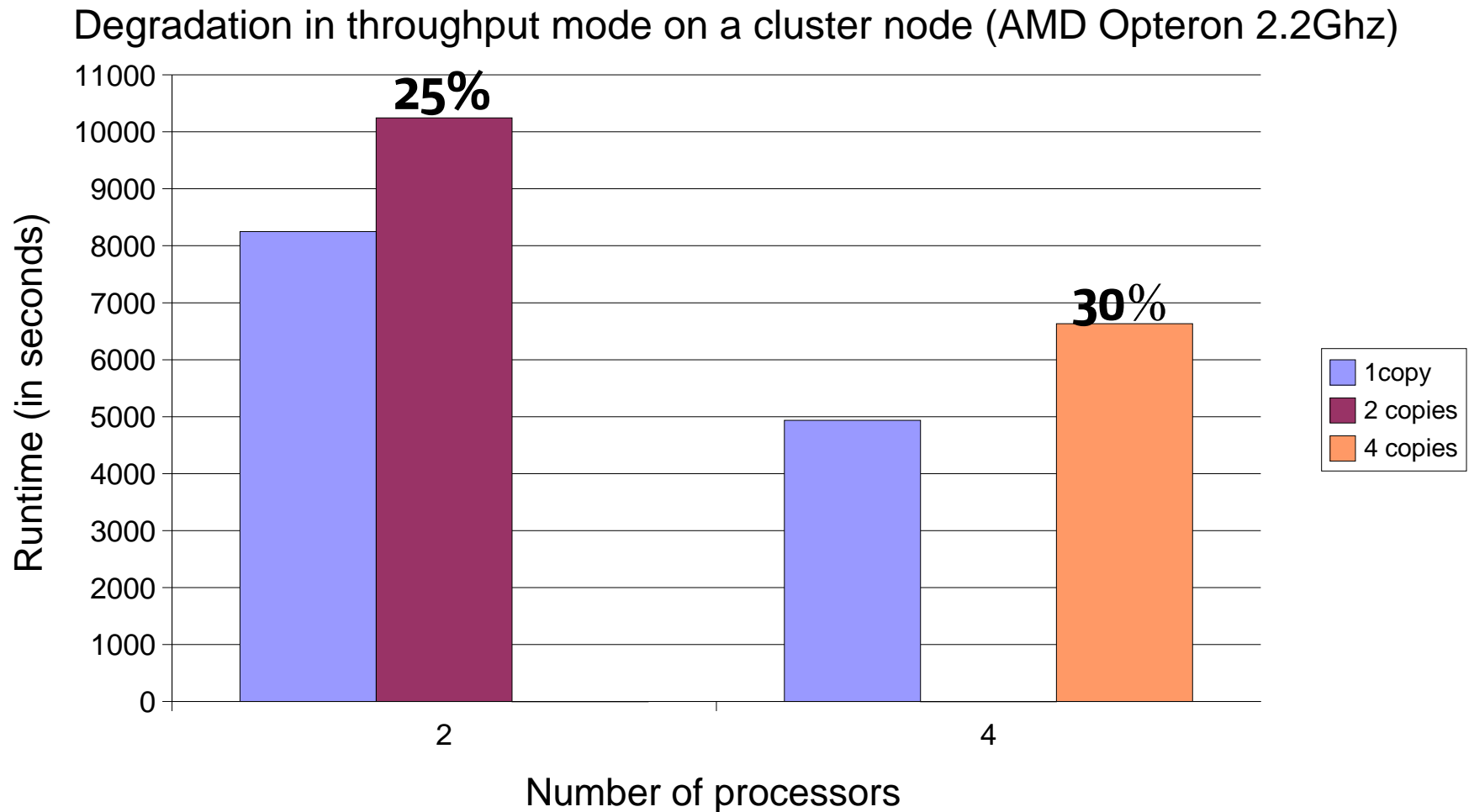
ECLIPSE results

Single Parallel Job

Turnaround on SMP, SMP with 1st gen CMT, and a cluster



Eclipse – Throughput experiments

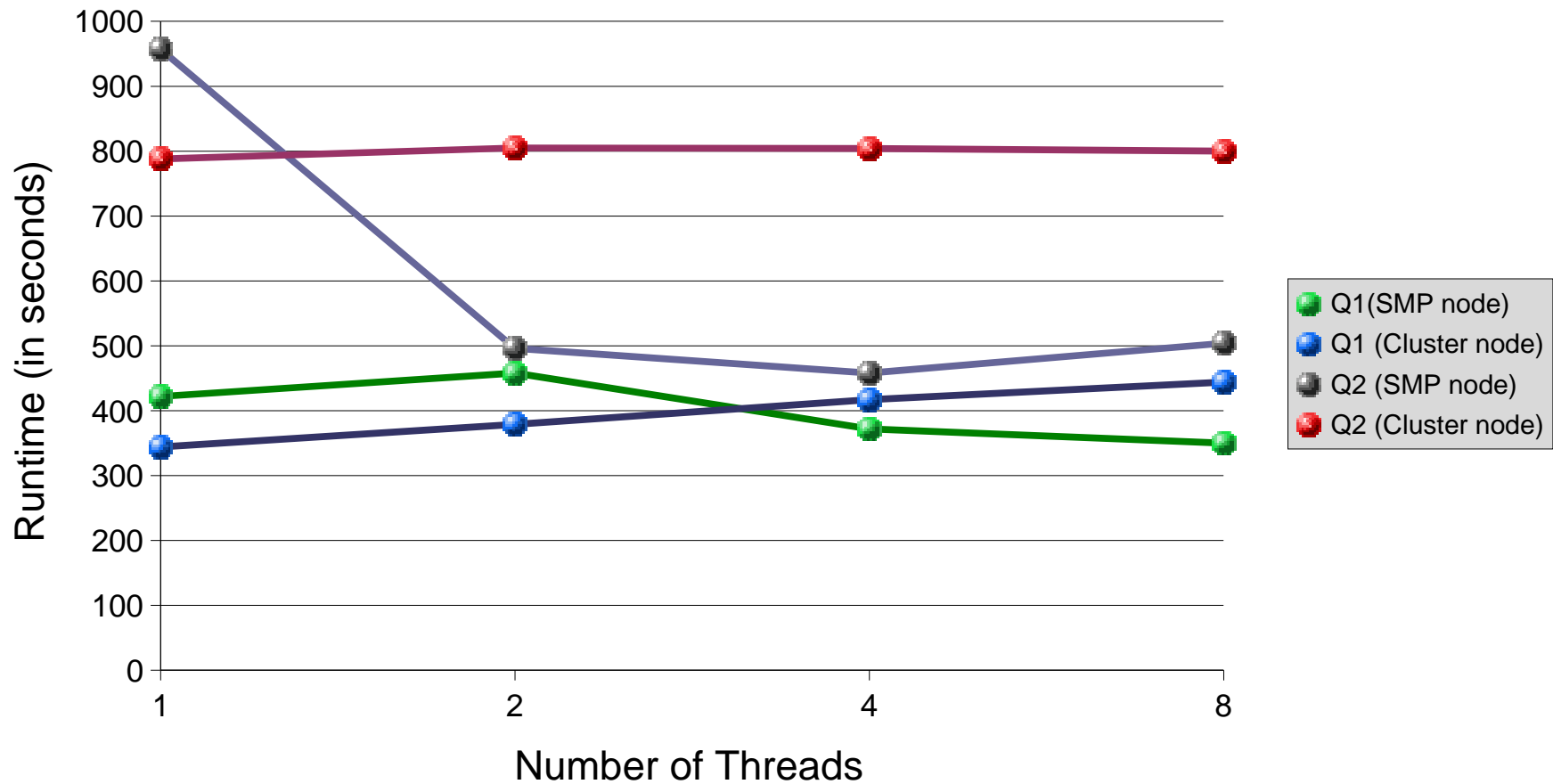


Surprisingly low degradation in turnaround; Big increase in throughput

BLAST results

Oversubscribed mode

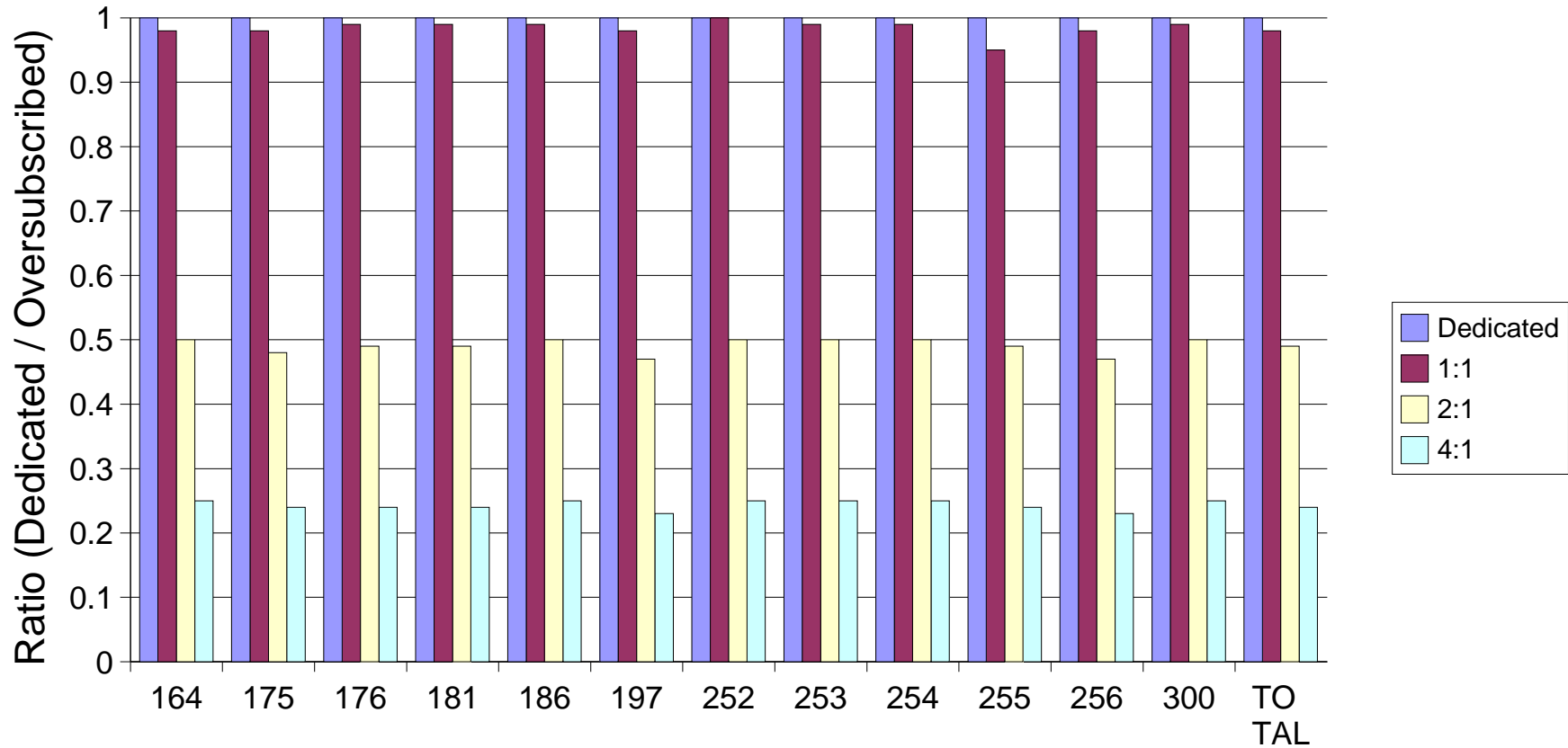
BLAST results on 2 cpu SMP node, and cluster node



Throughput Experiments

Preliminary results – Many single cpu jobs

Turnaround performance on 2 CPU machine in various oversubscribed modes



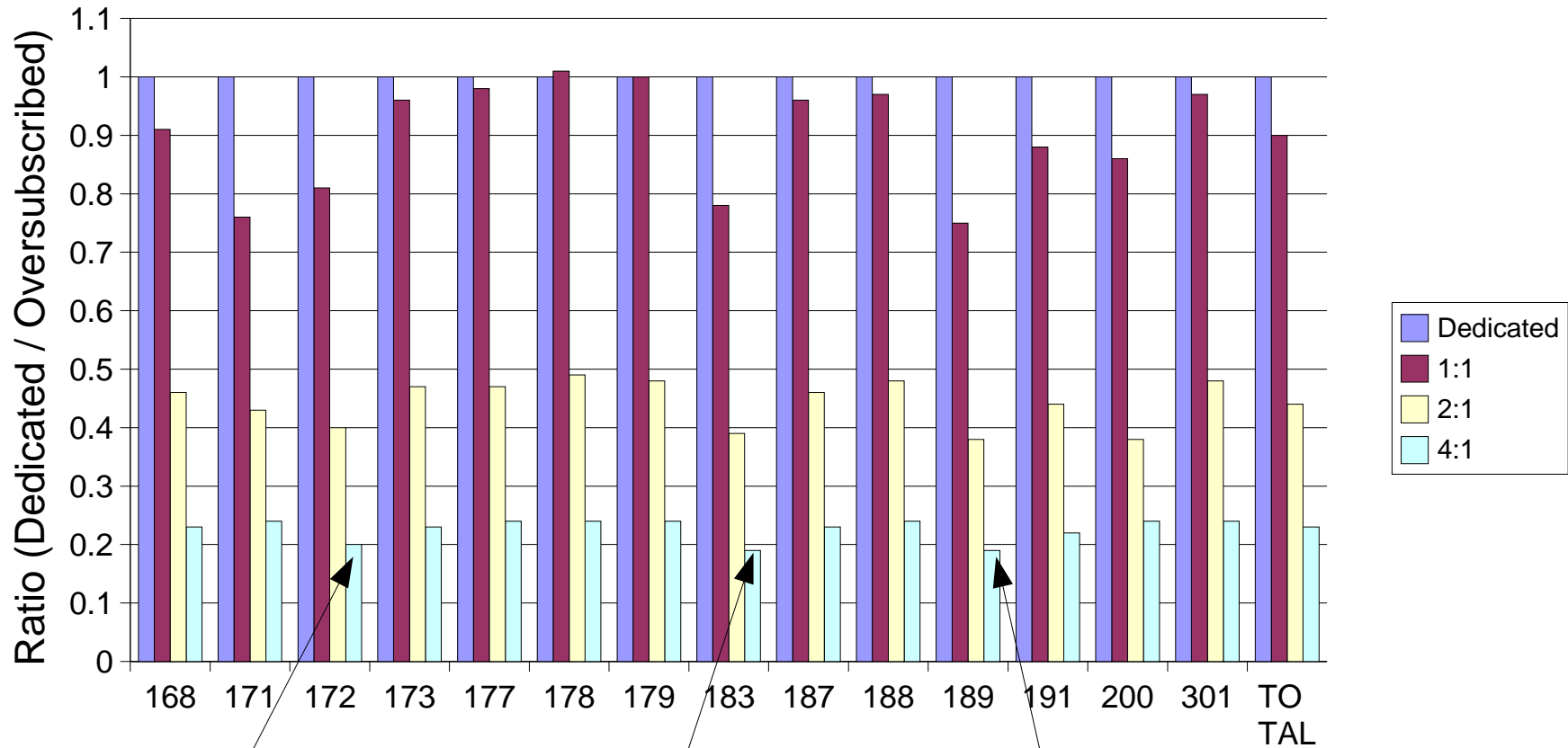
SPECCPU2000 INTEGER benchmarks

Throughput does not degrade in oversubscribed mode

Throughput Experiments

Preliminary results – Many single cpu jobs

Turnaround performance on 2 CPU machine in various oversubscribed modes



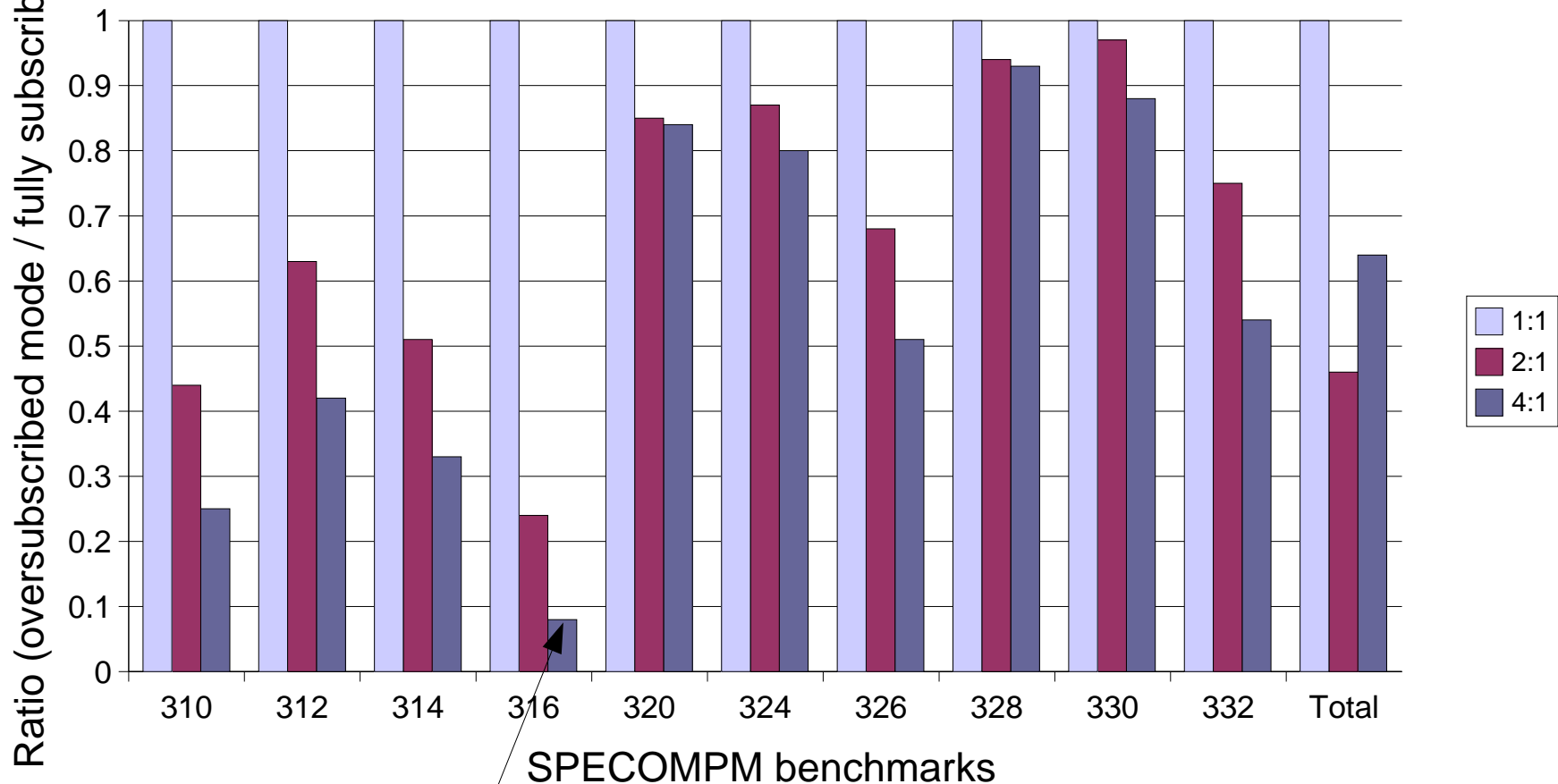
SPECCPU2000 FLOATING POINT benchmarks

Several benchmarks here put pressure on shared resources: cpu, caches memory bandwidth

Throughput Experiments

Preliminary results – Single parallel job, more threads

Degradation due to oversubscription by 2:1 and 4:1, run on 4 cpus of SMP

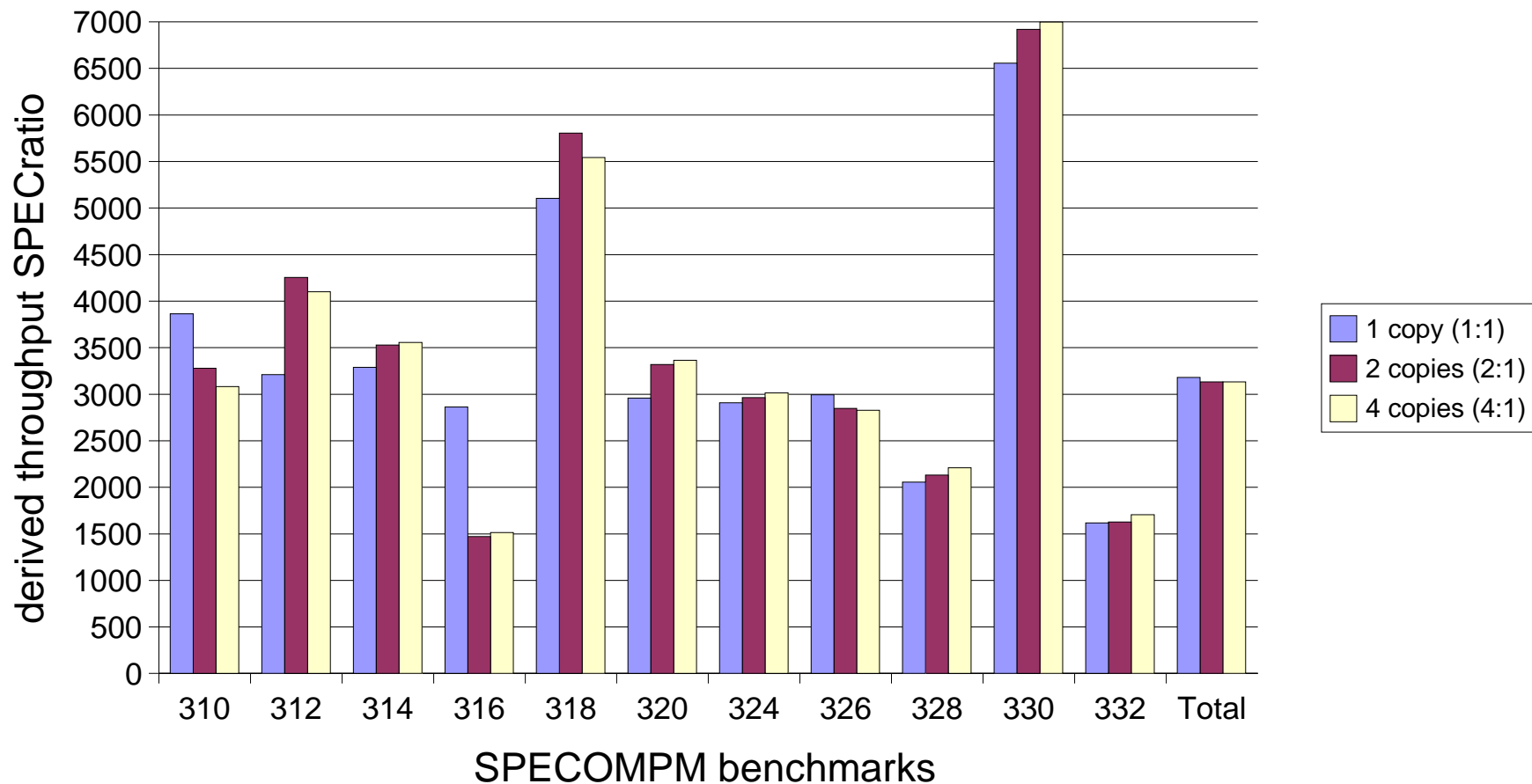


Execution with co-located threads should improve with CMT

Throughput Experiments

Preliminary results – Multiple parallel jobs

Throughput performance in 1:1, 2:1 and 4:1 mode on 4 CPU SMP



Throughput stays up; CMT will help in big way

Summary

- Chip multi-threading allows throughput computing to address turnaround concurrently for multiple users
- Oversubscription hurts on current systems
- Full subscription of machine reflects most production use
 - Ability to support concurrent users with parallel jobs
 - Reduce turnaround time for many jobs
- Throughput computing is the present and it is the future
 - Present: SMPs, Clusters Future: CMT, Clusters
 - Gain cost advantage, efficiency and reliability with CMT
- More analysis and experiments are needed in determining
 - Shared resource use; overheads ; OS scheduling ; hardware trade-offs
- Exploit TLP using CMT to reduce Processor-Memory gap