CS315A/EE382B: Lecture 7

Irregular Scientific and Commercial Applications

Kunle Olukotun
Stanford University

http://eeclass.stanford.edu/cs315a

Announcements

- PS1
  - Due today
- PA2
  - Due May 10
- New information sheet
  - Change in one of readings

Today's Outline: Irregular Applications

- Irregular applications: Characteristics
  - Variation in workload
  - Variation in dataset
- Some example applications:
  - Sparse matrix multiply
  - Barnes-Hut
- Commercial applications:
  - OLTP
  - Decision support
- Memory System Characterization of Commercial Workloads

Characteristics of Irregularity

- What are "irregular" applications?

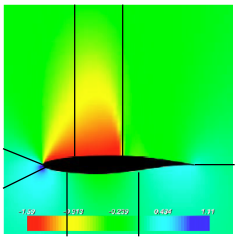
- Ones with significantly varying work/datapoint
  - Hard to estimate how time will be spent
  - Different processors running different code (database)
  - Need task queuing to balance loads

- Ones with datasets not easily divisible by processor
  - Changing data layout over time
  - Sparse data layout
  - Trees of dependent work (sorts, searches)

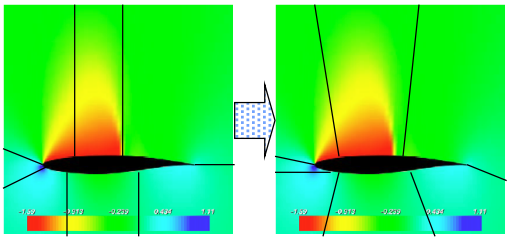
Changing Datasets

- Take a static, grid-based simulation algorithm
  - Can divide up a priori, in an optimized way
- Now adjust the problem structure on a regular basis
  - To focus in on areas of particular interest
  - Requires dynamic re-division of the whole grid

Static = Easy!



Adjusting Grids = Hard!



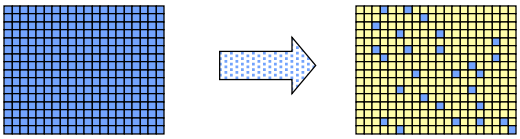
How Do We Rebalance?

- **Task stealing** is a quick and simple solution
  - But leads to poor memory locality
  - Can only work if we are using task queues
  - Should only be used for temporary shifting of a few points

- Need to **migrate** points to reflect long-term trends
  - Must keep track of approximate work/point
  - Quick heuristics to determine # points/processor
  - Like census & congressional representatives
  - Move points across processor borders:
    - Underutilized processors eat up neigboring points
    - Overutilized processors release points
    - All may have to shift to keep all processors' points together

Sparse Matrices I

- We have discussed "dense" matrix calculations
  - Most numbers in matrix are non-zero
  - Best solution performs all *potentially* necessary calculations
- But many matrix calculations are "sparse"
  - Only a few % of numbers are non-zero
  - Zeros make most answers easy-to-determine
  - Best solution is to *skip* unnecessary calculation . . .
  - . . . But now our calculation densities are hard-to-predict



Sparse Matrices II

- Communication patterns are also hard-to-predict
  - Must find active points that are multiplied with each other
    - May be fairly far away in the matrix, so localization doesn't help
  - Must deal with **hot spots**: nodes used many times
    - May need to be replicated or divided
  - Often need to map nodes to processors *quickly*
    - May only use for one or two multiplies
- There is **no** "standard" algorithm for sparse matrix multiply
  - Some basic division patterns are a start (rows, blocks, etc.)
  - But algorithms are tuned for:
    - Most frequent arrangement of non-zero elements

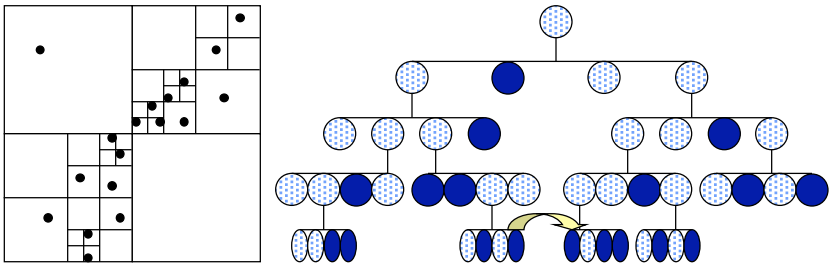
Barnes-Hut I

- A common N-body gravitational problem solver
  - Best described by an analogy:
    - Dense matrix multiply is to sparse matrix multiply, as
    - Dense N-body code is to Barnes-Hut
  - Deals with scattered objects in space:
    - A planet here, a star there, a comet over there
    - And LOTS of empty space in between

- Combines many problems into one application:
  - Sparse data representation
  - Unbalanced load
  - Need to find local neighbors across a complex data structure
  - Data points gradually move around in space

Barnes-Hut II

- Major problem is the data structure: an “octtree”
  - Divides 3-D space into 8 equal-size cubes
  - Subdivides any cubes containing more than one body
  - Continue recursively dividing up data
- Here is a 2-D “quadtree” equivalent:
  - Note that nearest neighbors are not always close in tree



Barnes-Hut III

- This application is difficult to parallelize
  - Must communicate in all-to-all fashion
    - Complicated by combining of remote elements (“superbodies”)
    - Dense parts of tree take longer
  - Must allocate tree branches dynamically
    - Depth of the tree varies greatly
  - Nearest-neighbors may be far apart in the tree
  - Must move nodes around in the tree at every timestep
- Communication is *unstructured*
  - Lots of local references
  - But also a fair number of remote references

OLTP: On-Line Transaction Processing

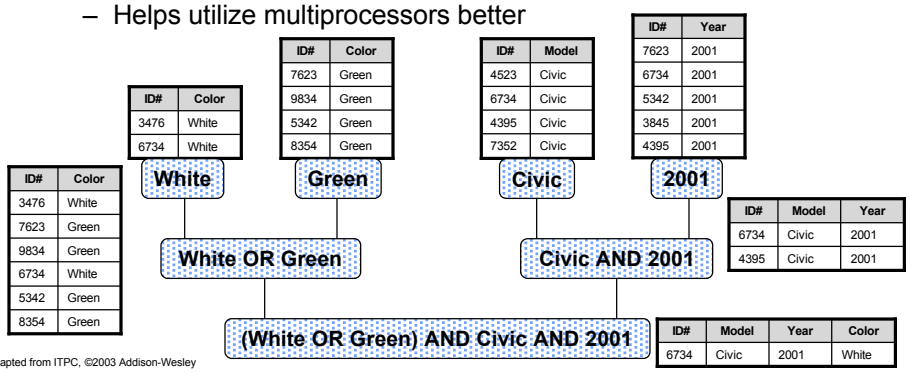
- Database application with many uses
  - Bank/credit card account management systems
  - Warehouse inventory management
- Lots of small “transaction” tasks
  - Each transaction just updates a small number of records
    - *Ex.:* Buyer’s & Seller’s bank accounts
  - Each task is typically read-write
    - Make a quick search to find affected records
    - Update all records
    - Save results in a *reliable* way
- Multiprocessor interaction requires *locks on records*

DSS: Decision Support Systems I

- Businesses often want to learn about customers
  - Look up records of a certain type (*Ex.:* Customers in N. Cal)
  - Look for relationships (*Ex.:* In N. Cal *and* own a home)
- Must support a few tasks on *tables of records*
  - Extract records of one type from a database (search)
  - Intersection/union of table pairs to form composite
  - Sort
- Different requirements from OLTP
  - Much larger "transactions"
    - Each one is a complex sequence of basic tasks
  - Lower I-O/computation ratio, so kernel has little effect
  - Typically read-only: R/W locks useful
    - Although OLTP may be occurring simultaneously

DSS: Decision Support Systems II

- Larger transactions make *intra*-transaction parallelism important
  - Can parallelize within/**among** each basic table operation
  - Helps hide disk latency with more threads
  - Helps utilize multiprocessors better



Adapted from ITPC, ©2003 Addison-Wesley

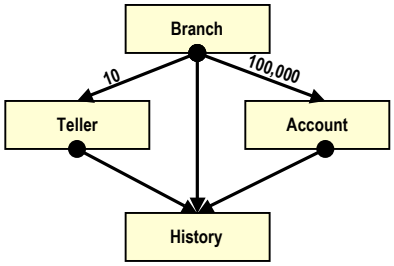
Memory System Characterization of Commercial Workloads

- Market shift for high-performance systems
  - yesterday: technical/numerical applications ($2B today)
  - today: databases, web servers, e-mail services, etc ($48B today).
- Bottleneck shift in commercial application
  - yesterday: I/O
  - today: memory system
- Lack of data on behavior of commercial workloads
- Re-evaluate memory system design trade-offs

TPC history of benchmarks

- TPC-A
  - First OLTP benchmark
  - Based on Jim Gray's Debit-Credit benchmark
- TPC-B
  - Simpler version of TPC-A
  - Meant as a stress test of the server only
- TPC-C
  - Current TPC OLTP benchmark
  - Much more complex than TPC-A/B
- TPC-D
  - Current TPC DSS benchmark
- TPC-W
  - New Web-based e-commerce benchmark

The TPC-B benchmark



- Models a bank with many branches
  - 1 transaction type: account update

```
Begin transaction
    Update account balance
    Write entry in history table
    Update teller balance
    Update branch balance
Commit
```

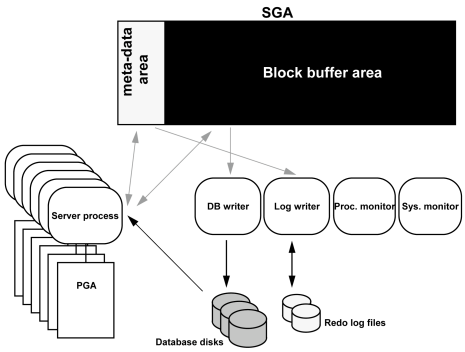
- Metrics:
  - tpsB (transactions/second)
  - $/tpsB
- Scale requirement:
  - 1 tpsB needs 100,000 accounts

Workloads

- OLTP (on-line transaction processing)
  - modeled after TPC-B, using Oracle7 DB engine
  - short transactions, intense process communication & context switching
  - multiple transactions in-transit
- DSS (decision support systems)
  - modeled after TPC-D, using Oracle7
  - long running transactions, low process communication
  - parallelized queries
- AltaVista
  - Web index search application using custom threads package
  - medium sized transactions, low process communication
  - multiple transactions in-transit

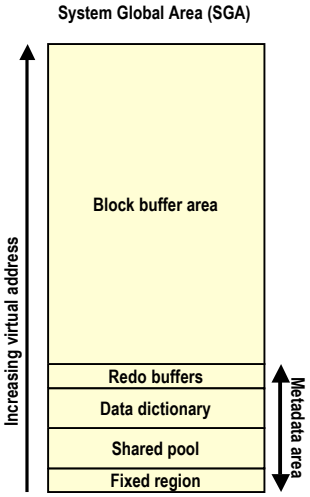
Oracle: software structure

- Server processes
  - actual execution of transactions
- DB writer
  - flush dirty blocks to disk
- Log writer
  - writes redo logs to disk at commit time
- Process and system monitors
  - misc. activity monitoring and recovery
- Processes communicate through SGA and IPC



Oracle: software structure(2)

- SGA:
  - shared memory segment mapped by all processes
- Block buffer area
  - cache of database blocks
  - larger portion of physical memory
- Metadata area
  - synchronization structures
  - shared procedures
  - directory information
  - How does metadata differ from block buffer area?
- Hiding I/O latency:



Methodology: Platform

- AlphaServer4100 5/300
  - 4x 300 MHz processors (8KB/8KB I/D caches, 96KB L2 cache)
  - 2MB board-level cache
  - 2GB main memory
  - latencies: 1:7:21:80/125 cycles
    - Why two memory latencies
  - 3-channel HSZ disk array controller
- Digital Unix 4.0B

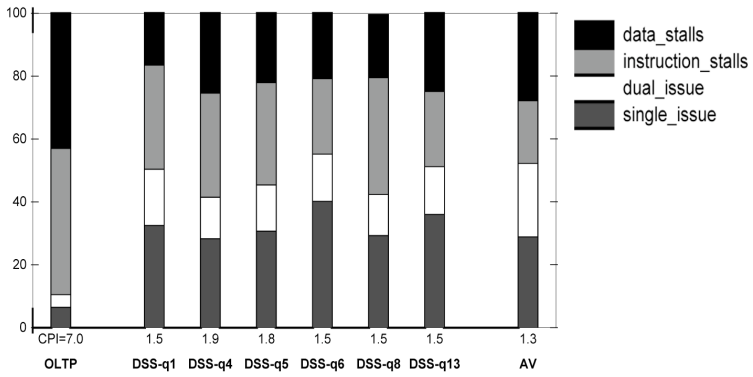
Methodology: Tools

- Monitoring tools:
  - IPROBE
  - DCPI
  - ATOM
- Simulation tools:
  - tracing: preliminary user-level studies
  - SimOS-Alpha: full system simulation, including OS
- Why is combination good?

Scaling

- Why do we need to scale benchmark?
- Scaling the problem size is critical
- Validation criteria: ?
- Requires good understanding of workload
  - make sure system is well tuned
  - ?
  - ?
- Does scaling work?

CPU Cycle Breakdown

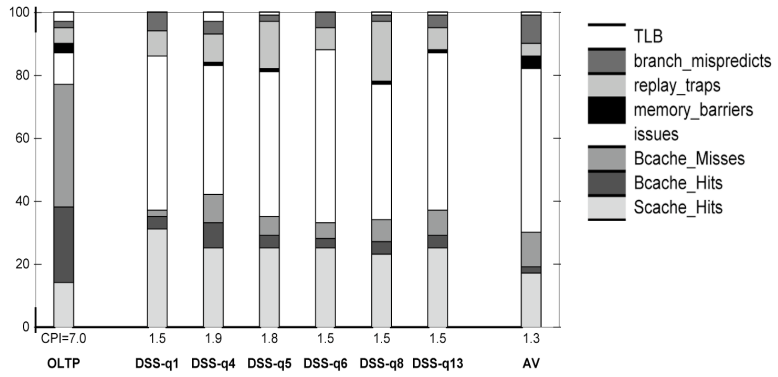


- **Very high CPI for OLTP**
- **Instruction and data related stalls are equally important**
  - **Why?**

Cache behavior

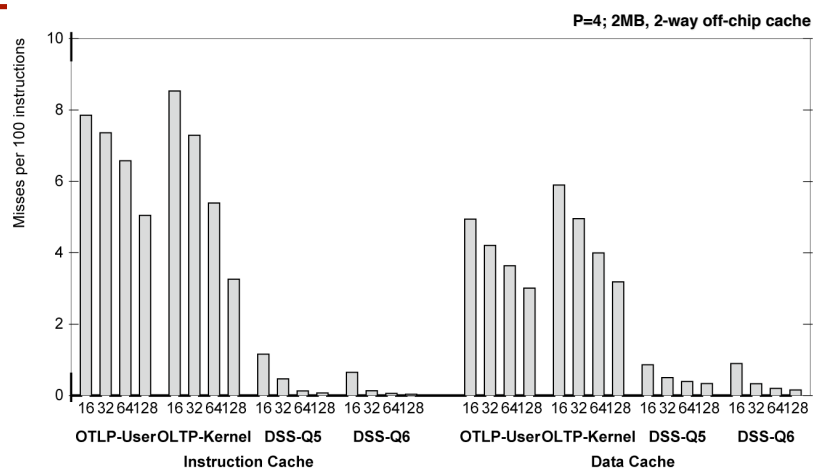
	OLTP	DSS-Q1	DSS-Q4	DSS-Q5	DSS-Q6	DSS-Q8	DSS-Q13	AltaVista
Icache (global)	19.9%	9.7%	8.5%	4.6%	5.9%	3.7%	6.7%	1.8%
Dcache (global)	42.5%	6.9%	22.9%	11.9%	11.3%	11.0%	12.4%	7.6%
Scache (global)	13.9%	0.8%	2.3%	1.0%	0.6%	1.0%	1.0%	0.7%
Bcache (global)	2.7%	0.1%	0.5%	0.2%	0.2%	0.3%	0.3%	0.3%
Scache (local)	40.8%	3.6%	10.7%	5.7%	3.9%	6.0%	6.1%	7.6%
Bcache (local)	19.1%	13.0%	21.3%	23.9%	30.7%	27.9%	31.3%	41.2%
Dirty miss fraction	15.5%	2.3%	2.2%	10.6%	1.7%	8.4%	3.3%	15.8%

Stall Cycle Breakdown



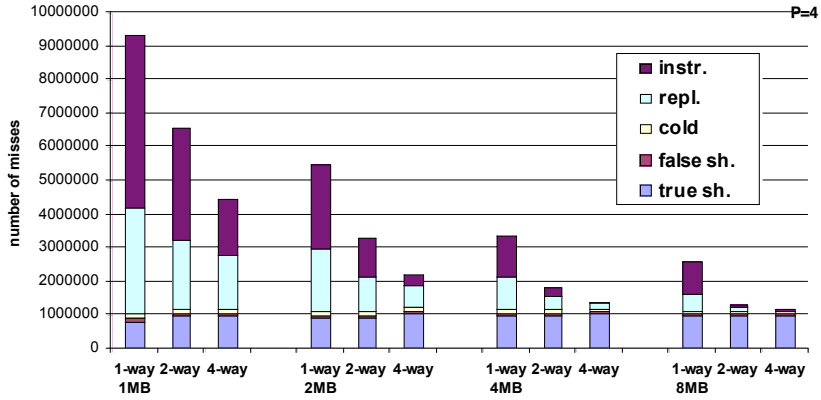
- **OLTP dominated by non-primary cache and memory stalls**
- **DSS and AltaVista stalls are mostly Scache hits**

Impact of On-Chip Cache Size



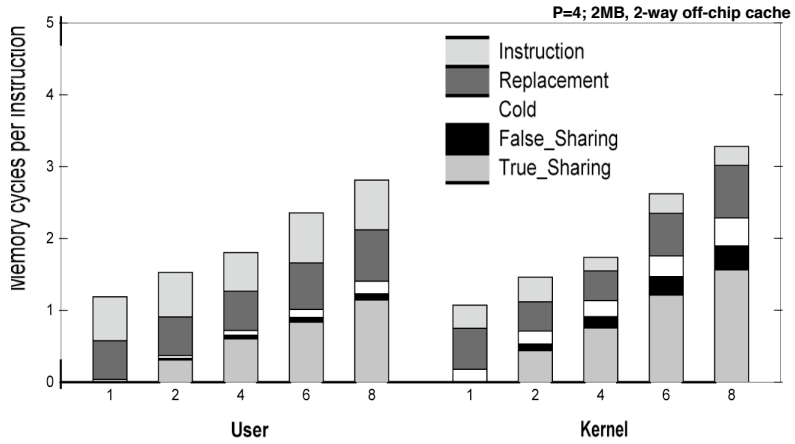
- **What can you conclude about cache size for DSS?**

OLTP: Effect of Off-Chip Cache Organization



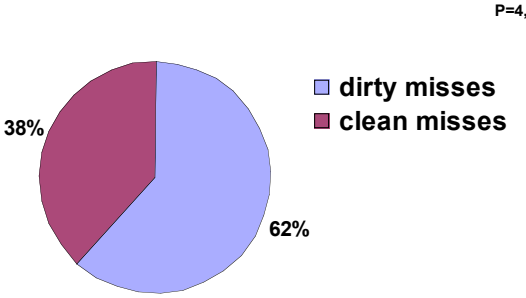
- **Significant benefits from large off-chip caches (up to 8MB)**
- **What is the impact of large caches besides low miss rate?**
- **What is impact of set-associativity?**

OLTP: Impact of system size



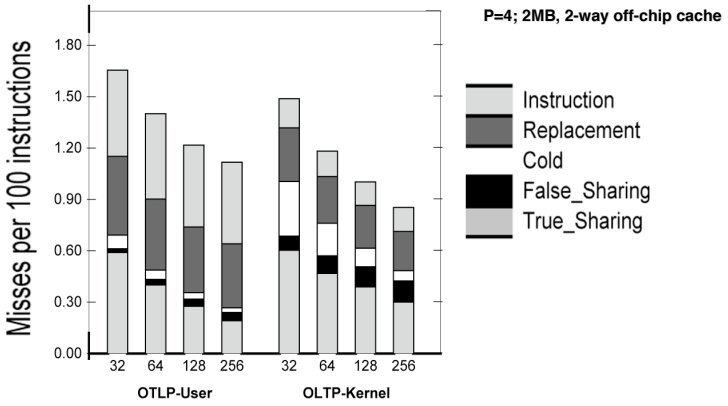
- Communication misses become dominant for larger systems
- Why does ratio of false sharing / true sharing not increase?

OLTP: Contribution of Dirty Misses



- Shared metadata is the important region
  - 80% of off-chip misses
  - 95% of dirty misses
- What happens to dirty misses with increases in cache and system size?

OLTP: Impact of Off-Chip Cache Line Size



- **Good spatial locality on communication for OLTP**
- **Very little false sharing in Oracle itself**

Summary of Results

- On-chip cache
  - 64KB I/D sufficient for DSS & AltaVista
- Off-chip cache
  - OLTP benefits from larger caches (up to 8MB)
- Dirty misses
  - Can become dominant for OLTP

SPLASH vs. Online Transaction Processing (OLTP)

A typical SPLASH app. has

> 3x the issue rate,
>
> ~26x less cycles spent in memory barriers,
>
> 1/4 of the TLB miss ratios,
>
> < 1/2 the fraction of cache-to-cache transfers,
>
> ~22x smaller instruction cache miss ratio,
>
> ~1/2 L2$ miss ratio

...of an OLTP app.

Conclusions

- Parallelizing irregular applications offers a challenge
  - Get good speedup . . .
    - . . . While balancing uneven-sized work
    - . . . And difficult-to-split datasets
    - Requires planning of task stealing & adjustment, for SM
- Commercial applications
  - Memory system is the current challenge in DB performance
  - Careful scaling enables detailed studies
  - Combination of monitoring and simulation is very powerful
  - Architect needs deep understanding of the workload
  - Diverging memory system designs
    - OLTP benefits from large off-chip caches, fast communication
    - DSS & AltaVista may perform better without an off-chip cache