1 pass to sort each page (2N pages)

1 pass to merge into 2 sorted pages (2N pages)

1 pass to merge into 4 sorted pages (2N pages)

1 pass to merge into 8 sorted pages (2N pages)

$2N(\log_2(N)+1)$
take into account the whole memory hierarchy

measure cost in bytes moved across the memory hierarchy

build model step by step

(for model) focus on the weak link
data size: $N$ pages
memory size: $M$ pages

how much memory $M$ do we need to sort $N$ data in $p$ passes only?

or

how much data can we sort in $p$ passes if we have $M$ memory?

$$\log_{M-1}(N/M)+1 \leq p$$
I spend a lot of time debugging. Am I doing something wrong? Maybe, but probably not.

1. Learn to use gdb & valgrind
2. After spending X time debugging, ask for help
3. Enjoy it :)

Test extensively every few lines of code. Isolate problems - one change at a time.
Martin Kersten
SIGMOD Systems Award
ACM SIGMOD Edgar F. Codd Innovations Award

Extra: **A Database System with Amnesia**
Conference on Innovative Data Systems Research, 2017
fast scans

hardware, data and query based optimizations
(project M2)

apply to all algo/data structures
CPU

memory

Vs.

registers

on chip cache

on board cache

memory

disk
always want to minimize data movement - computation

& utilize all resources!
1. can do >1 tasks at the same time

2. can predict data accesses & execute instructions out of order

3. can do the same task on >1 data items
from single core to multi-core

```
cpu
  core
  registers
  L1
  L2
  L3
  memory

cpu
  core1
  core2
  core3
  core4
  regis.
  regis.
  regis.
  regis.
  L1
  L2
  L3
  memory
```
from multi-core to NUMA

data placement becomes crucial

Extra: **ATraPos: Adaptive transaction processing on hardware Islands**
Danica Porobic, Erietta Liarou, Pinar Tözün, Anastasia Ailamaki
International Conference on Data Engineering (ICDE), 2014
many “small” cores (1000s)
subsets of cores work on same task
so branches are again problematic

data transfer may be expensive
challenge

how do we keep all CPUs/cores at 100%

Extra: **Dynamic fine-grained scheduling for energy-efficient main-memory queries**
Iraklis Psaroudakis et al
International Workshop on Data Management on New Hardware (*DAMON*), 2014
L2

L1

L3

memory

can work in parallel

cpu

core1  core2  core3  core4

regis.  regis.  regis.  regis.

can work in parallel

whatever we bring in L1 we can break into L1/cores problems assign to core threads

data transfer problems remain the same
operator: average

memory: 56, 34, 12, 1, 87, 22, 98, 49, 7, 12, ...

data size = L1 size x 10
divide problem & run in parallel

read it only once in L1

read it only once in memory

Registers

On-chip cache

On board cache

Memory

Disk

CPU

A

A

A

A
loop fusion

watch out for *data locality*

```c
for(i=0;i<n;i++)
    min = a[i]<min ? a[i] : min

for(i=0;i<n;i++)
    max = a[i]>max ? a[i] : max
```

Which one is best
for(i=0;i<n;i++){ 
    min = a[i]<min ? a[i] : min 
    max = a[i]>max ? a[i] : max 
} 

vs.

for(j=0;j<n;j+=vectorsize){
    for(i=j;i<j+vectorsize;i++){
        min = a[i]<min ? a[i] : min 
    }
    for(i=j;i<j+vectorsize;i++){
        max = a[i]>max ? a[i] : max 
    }
}
loop fission

watch out for data locality

Which one is best

for(i=0;i<n;i++)
    min = a[i]<min ? a[i] : min
max = b[i]>max ? b[i] : max
...

for(i=0;i<n;i++)
    min = a[i]<min ? a[i] : min
max = b[i]>max ? b[i] : max
...

**full scan**: for every tuple check if the value satisfies the predicate and if it does remember the position of the tuple

```c
select(input, low, high) //inclusiveLow, inclusiveHigh)  
1: int *output = new(sizeOf(input.data[0]) * input.count)  
2: for (i = 0; i < input.count; i++)  
3: if input.data[i] > low && input.data[i] < high  
4: *output++ = i  
5: return output
```

sequential access pattern = good for CPU + memory hierarchy  
(next class more about why this is true)
what if we have >>1 queries arriving in parallel?

Project M2 = how can we keep all CPUs busy to 100% & minimize data movement?
N queries (SPA) in parallel on the same column
1) cost (L1 misses) for plain scan
2) devise shared scan approach
3) cost (L1 misses) for shared scan

interesting cases:
what if queries do not arrive at the same time?
what if some queries are faster than others?
is there a limit to the number of queries in a shared scan?

(assume simplified memory hierarchy)
Column > L1, Column < L2, L1 block = L2 block = block bytes, Column = C blocks
CPU can read directly from Level 1 only
1) opportunistic
2) utilize last chunk of data read from previous scan
3) start scanning backwards for scan 2
   …then forward again for scan 3 and so on

---

zigzag scans

---

```
1. opportunistic
2. utilize last chunk of data read from previous scan
3. start scanning backwards for scan 2
   ...then forward again for scan 3 and so on
```
2) schedule queries on same data to run in parallel

3) data is read only once

4) each query gets a thread/core from thread pool

**Data moves once**

# of cores queries run in parallel
data is read once and >1 queries scan in parallel

but this also means that >1 queries need to write output in parallel
data is read once and >1 queries scan in parallel

but this also means that >1 queries need to write output in parallel
data is read once and >1 queries scan in parallel

but this also means that >1 queries need to write output in parallel

scan->write
scan->write
scan->write
...

MEMORY
data is read once and >1 queries scan in parallel

but this also means that >1 queries need to write output in parallel

**CACHES**

```
scan->write
scan->write
scan->write
...
```

**MEMORY**
data is read once and >1 queries scan in parallel

but this also means that >1 queries need to write output in parallel
data is read once and >1 queries scan in parallel

but this also means that >1 queries need to write output in parallel

When we reach TLB limit 256/512 we need to resolve memory location thrashing TLB…TLB is the limit
attach queries arriving asynchronously
elevate queries that are slow

q1, q2, q3

q4

q4
fast scans

what to do for milestone 2?

assume you have all queries and treat them as a batch
minimize data movement by sharing
utilize multi-core CPUs via threading

goal: ideally (within reason) shared scan scales with # of queries and # of CPUs
demonstrate performance improvement with and without sharing
testing infrastructure contains performance tests

bonus: queries arriving asynchronously
Extra: **A Case for Staged Database Systems**
Stavros Harizopoulos, Anastassia Ailamaki
Conference on Innovative Data Systems Research, 2003

Extra: **The data cyclotron query processing scheme**
Romulo Goncalves, Martin L. Kersten
extra: **Navigating big data with high-throughput, energy-efficient data partitioning.**
L. Wu, R. J. Barker, M. A. Kim, K. A. Ross

extra: **Meet the walkers: Accelerating index traversals for in-memory databases.**
O. Koçberber, B. Grot, J. Picorel, B. Falsafi, K. T. Lim, P. Ranganathan
International Symposium on Microarchitecture, 2013

extra: **Beyond the Wall: Near-Data Processing for Databases**
S. Xi, O. Babarinsa, M. Athanassoulis, S. Idreos.
International Workshop on Data Management on New Hardware, 2015
Read: **Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS**  
Marcin Zukowski, Sándor Héman, Niels Nes, Peter A. Boncz  
Very large Databases Conference (**VLDB**), 2007

Browse: **Morsel-driven parallelism:**  
a NUMA-aware query evaluation framework for the many-core age  
Viktor Leis, Peter A. Boncz, Alfons Kemper, and Thomas Neumann  
ACM **SIGMOD** International Conference on Management of Data, 2014

Read: **textbook:** Chapters 9
fast scans 1.0

DATA SYSTEMS

prof. Stratos Idreos