Accelerating Database Operators Using a Network Processor

Brian Gold

Anastassia Ailamaki

Larry Huston

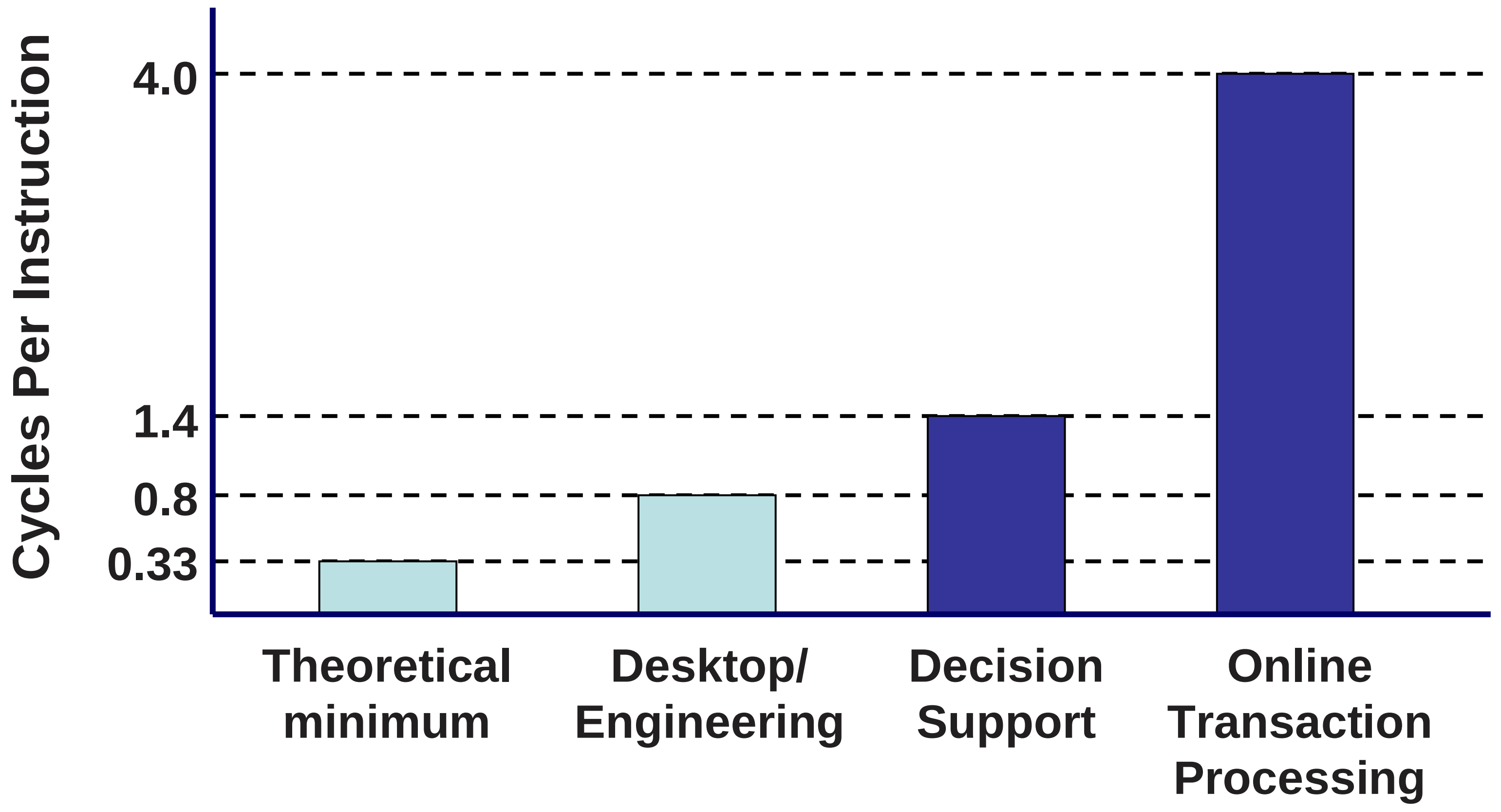
Babak Falsafi

Databases @Carnegie Mellon

intel

CALCM

DBMS on Modern Hardware



- Why so poor? Memory stalls!

Modern Architecture & DBMS

- Instruction-Level Parallelism (ILP)
  - Out-of-order (OoO) execution window
  - Cache hierarchies - spatial / temporal locality
- DBMS' memory system characteristics
  - Limited locality (e.g., sequential scan)
  - Random access patterns (e.g., hash join)
  - Pointer-chasing (e.g., index scan, hash join)
- DBMS needs Memory-Level Parallelism (MLP)

Prior Work

- Cache layout [Ailamaki 01] [Chilimbi 99] etc.
  - Increase utilization, reduce conflicts
  - Cannot hide miss latency
- Prefetching [Gracia Pérez 04] [Chen 03] etc.
  - Hide memory latency
  - Difficulties: pointer-chasing / random accesses
- SMT [Lo 98] [Garcia 05] [Zhou 05] etc.
  - Hide memory latency, expose MLP
  - Limited number of threads

Our Contributions

- DB operators on thread-parallel architectures
  - ❑ Expose parallel misses to memory
  - ❑ Leverage intra-operator parallelism

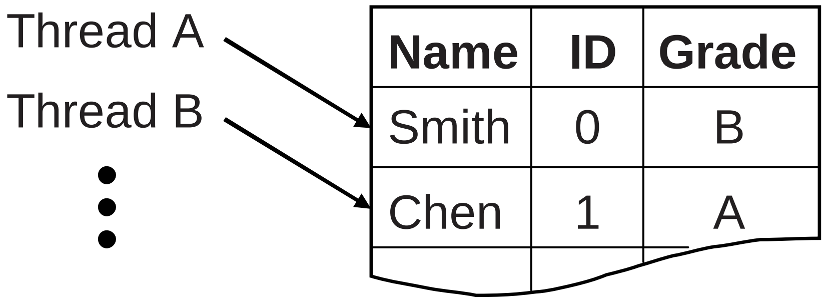
- Evaluation using network processors
  - ❑ Designed for packet processing
  - ❑ Abundant thread-level parallelism (64+)
  - ❑ Speedups of 2.0X-2.5X on common operators

Early insight on multi-core, multi-threaded architectures and DBMS execution

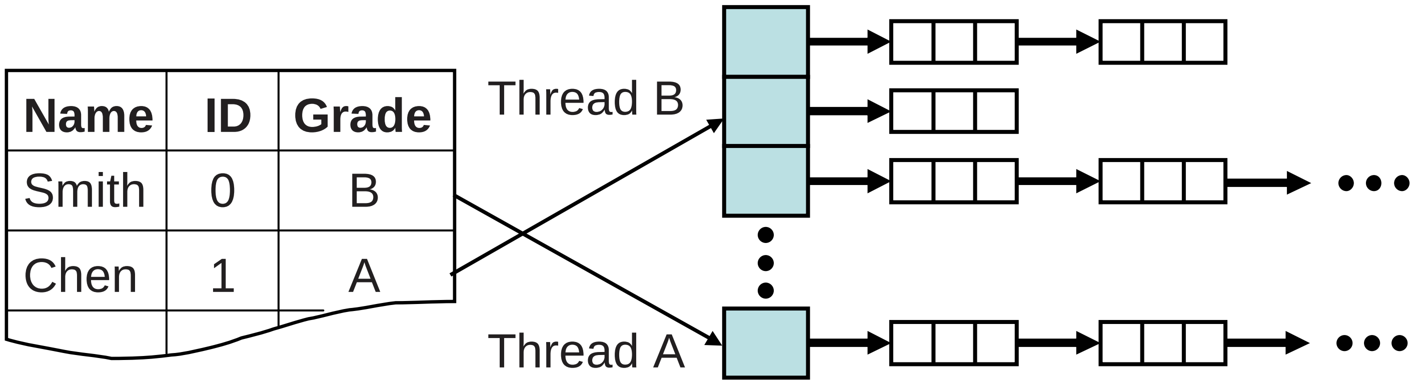
Outline

TLP in database operators

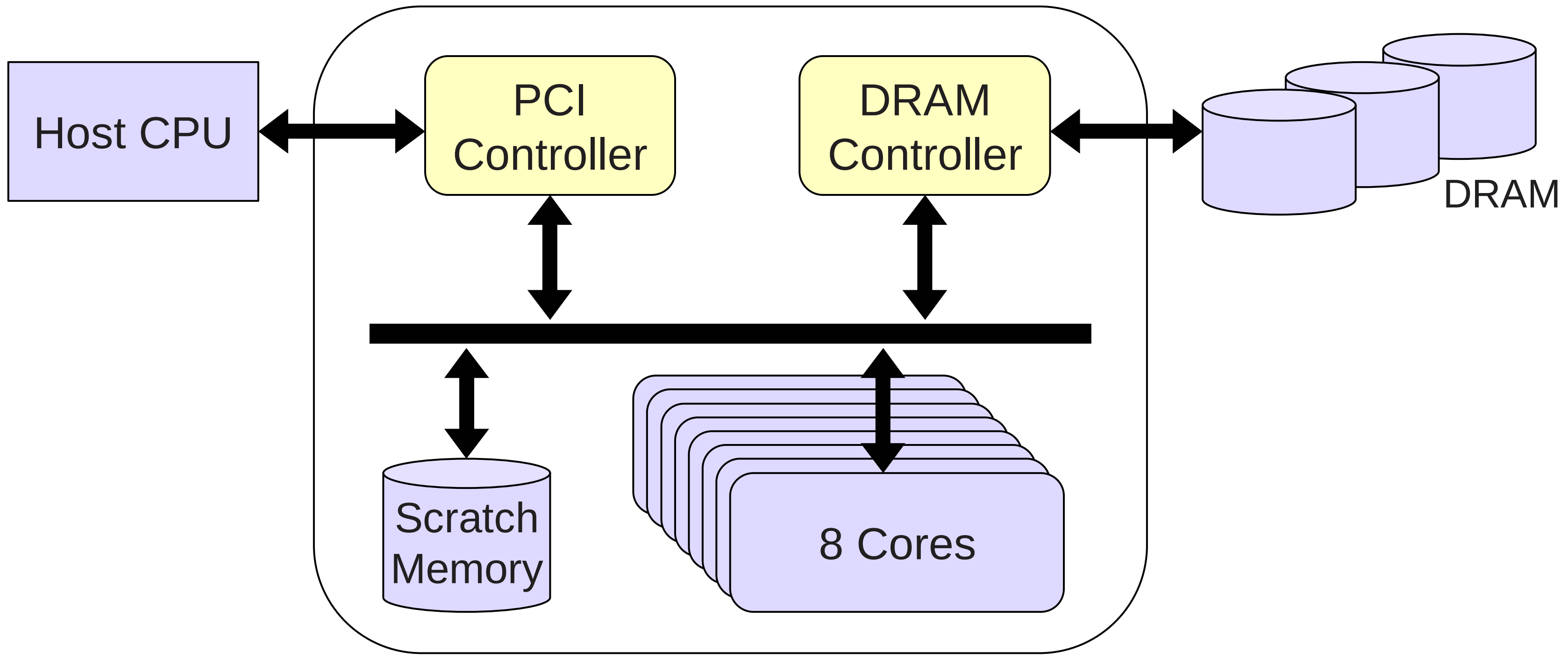
- Sequential or index scan
  - Fetch attributes in parallel



- Hash join
  - Probe tuples in parallel

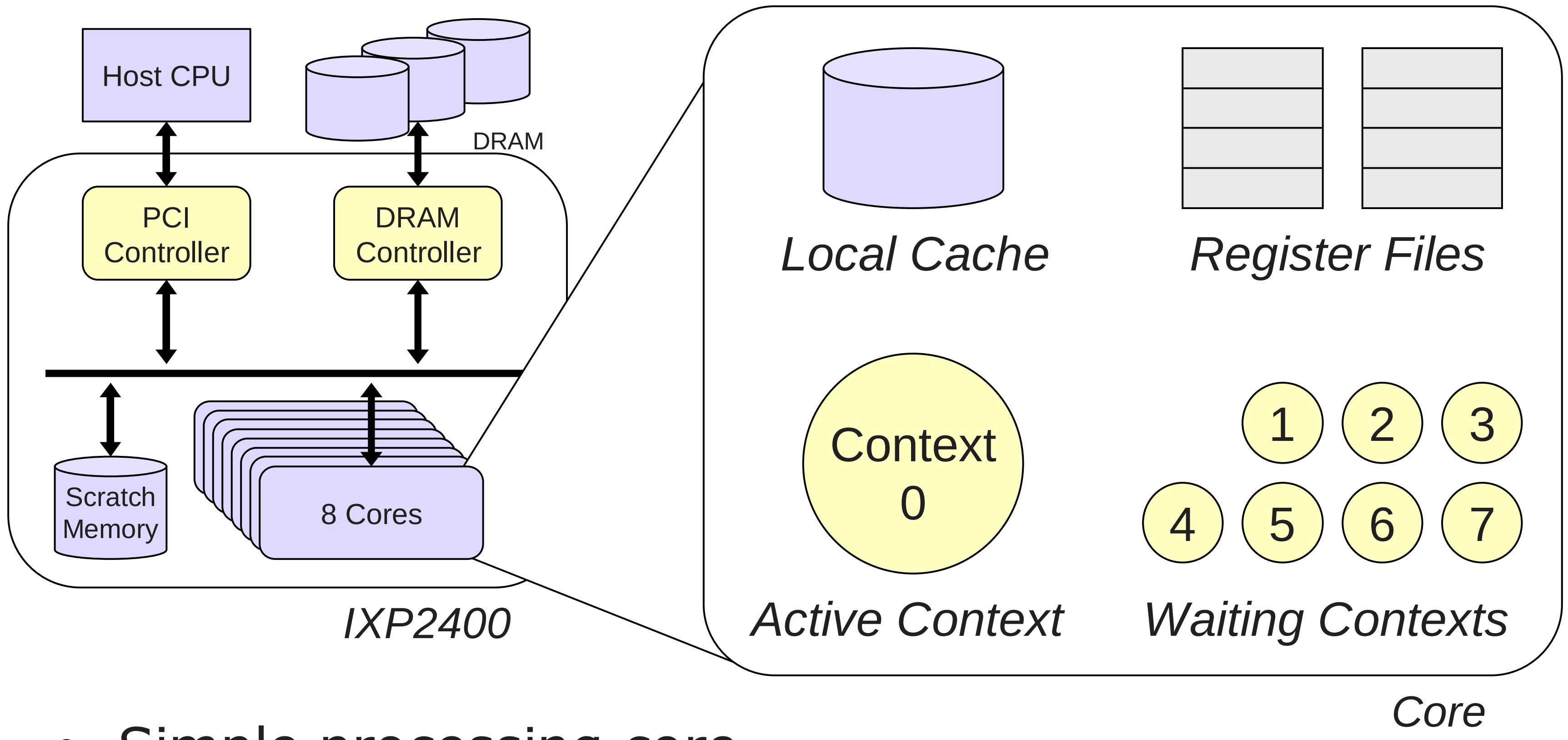


Network Processors



- Intel IXP2400
  - 8 cores, each with 8 thread contexts
  - Dedicated DDR SDRAM (up to 1GB)
  - < 20W power dissipation

Multi-threaded Core



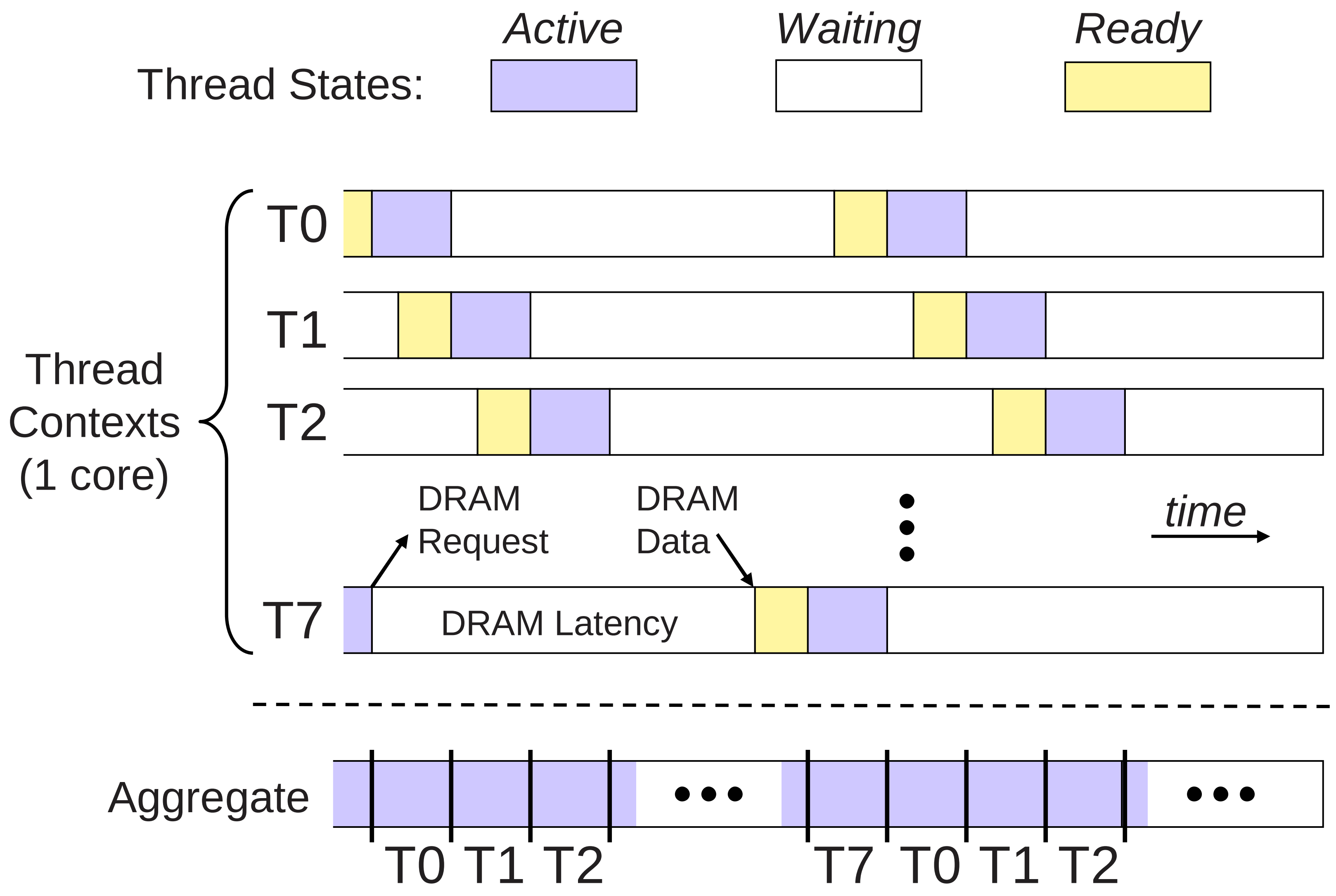
- Simple processing core
  - 5-stage, single-issue pipeline @ 600MHz
  - 2.5KB local cache
  - Switch contexts at programmer's discretion

Programming Model

- ISA supports thread switching
  - Wait for specific hardware ‘signal’
  - 4 cycle context switch (non-preemptive)
- Software-managed memory access
  - Instructions for DRAM, local, scratch memories
  - Programmer controls data access
- No OS or virtual memory

Sensible for simple, long-running code

Multithreading in Action

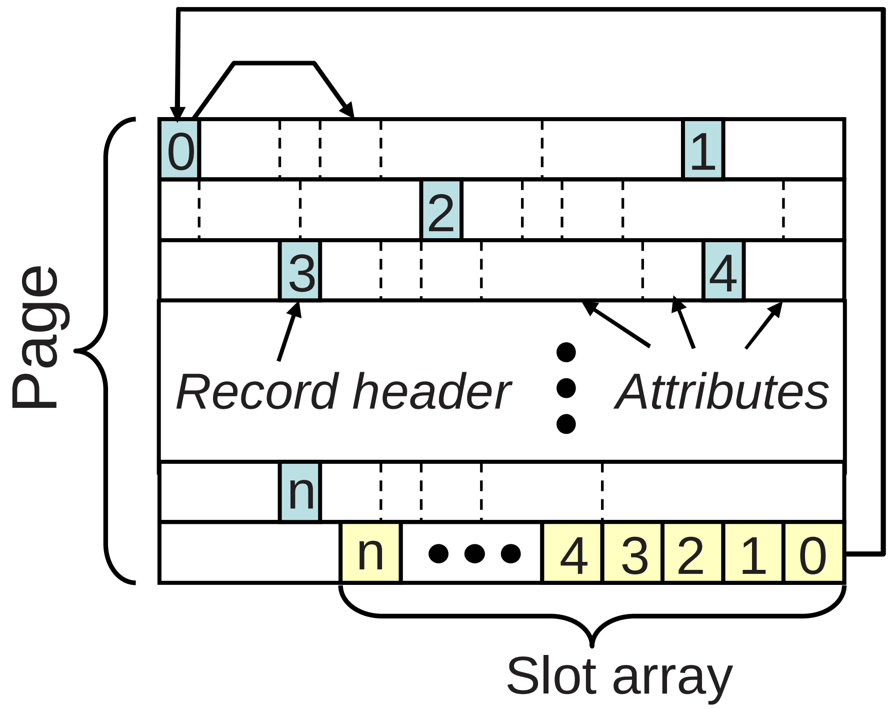


Methodology

- IXP2400 on prototype PCI card
  - 256MB PC2100 SDRAM
  - Separated from host CPU
- Pentium 4 Xeon 2.8GHz
  - 8KB L1D, 512KB L2, 4 pending misses
  - 3GB PC2100 SDRAM
- Workloads
  - TPC-H *orders* and *lineitems* tables (250MB)
  - Sequential scan, hash join

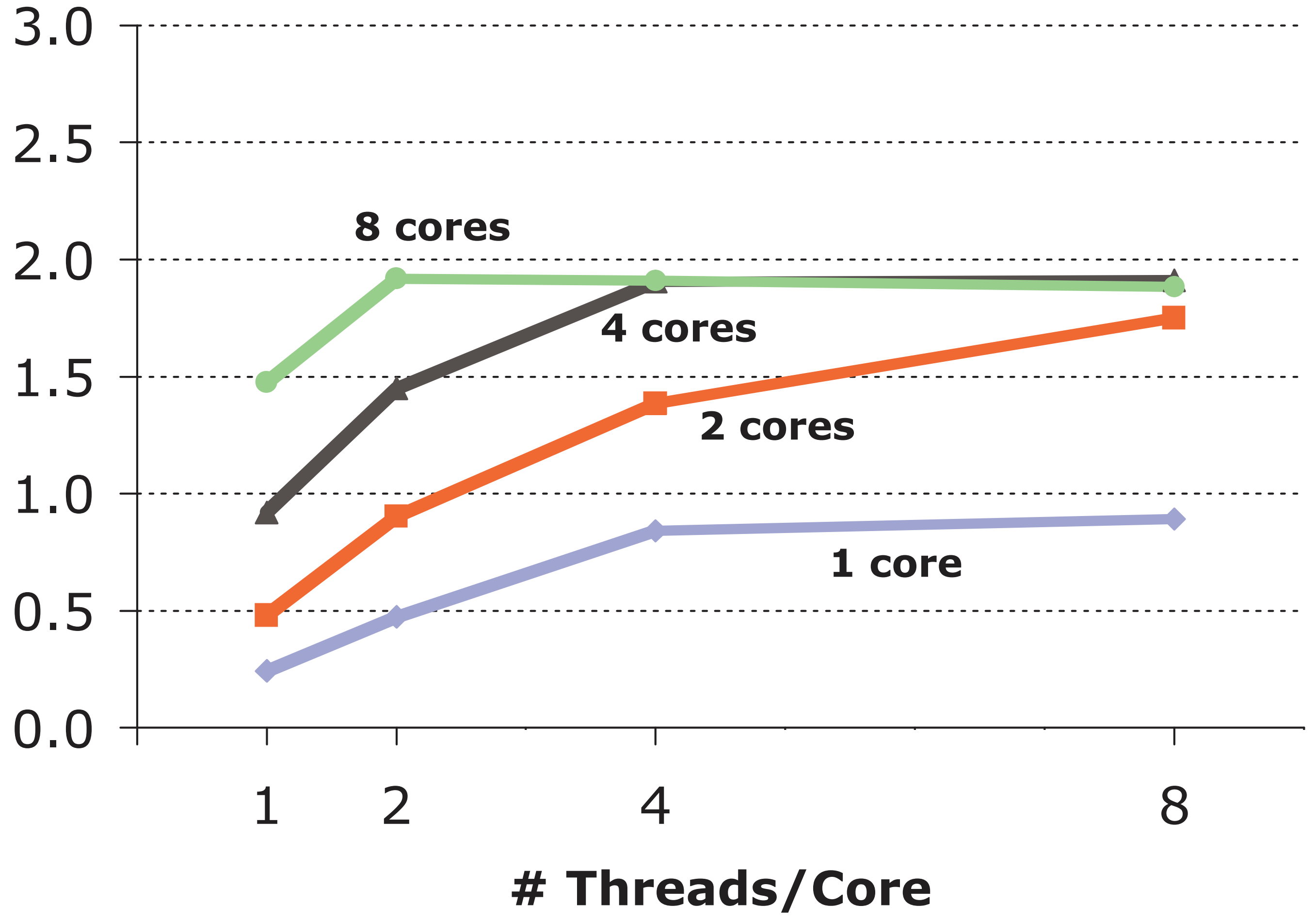
Sequential Scan

- Use slotted page layout (8KB)



- Network processor implementation
  - Each page scanned by threads on one core
  - Overlap individual record access within core

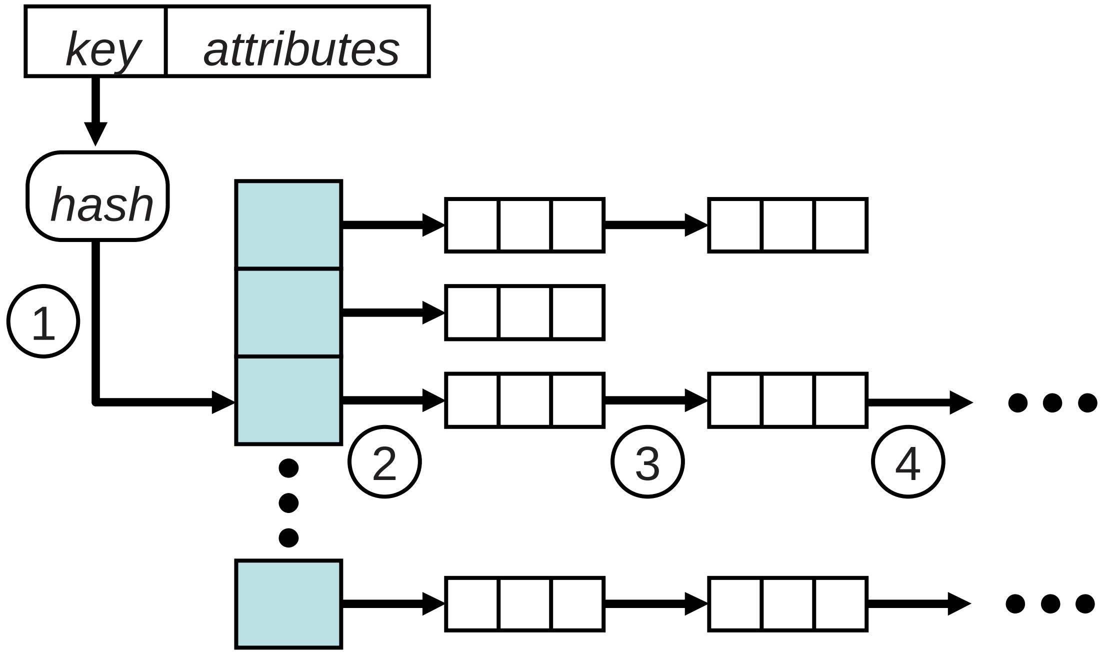
Sequential Scan Performance



Performance limited by DRAM controller

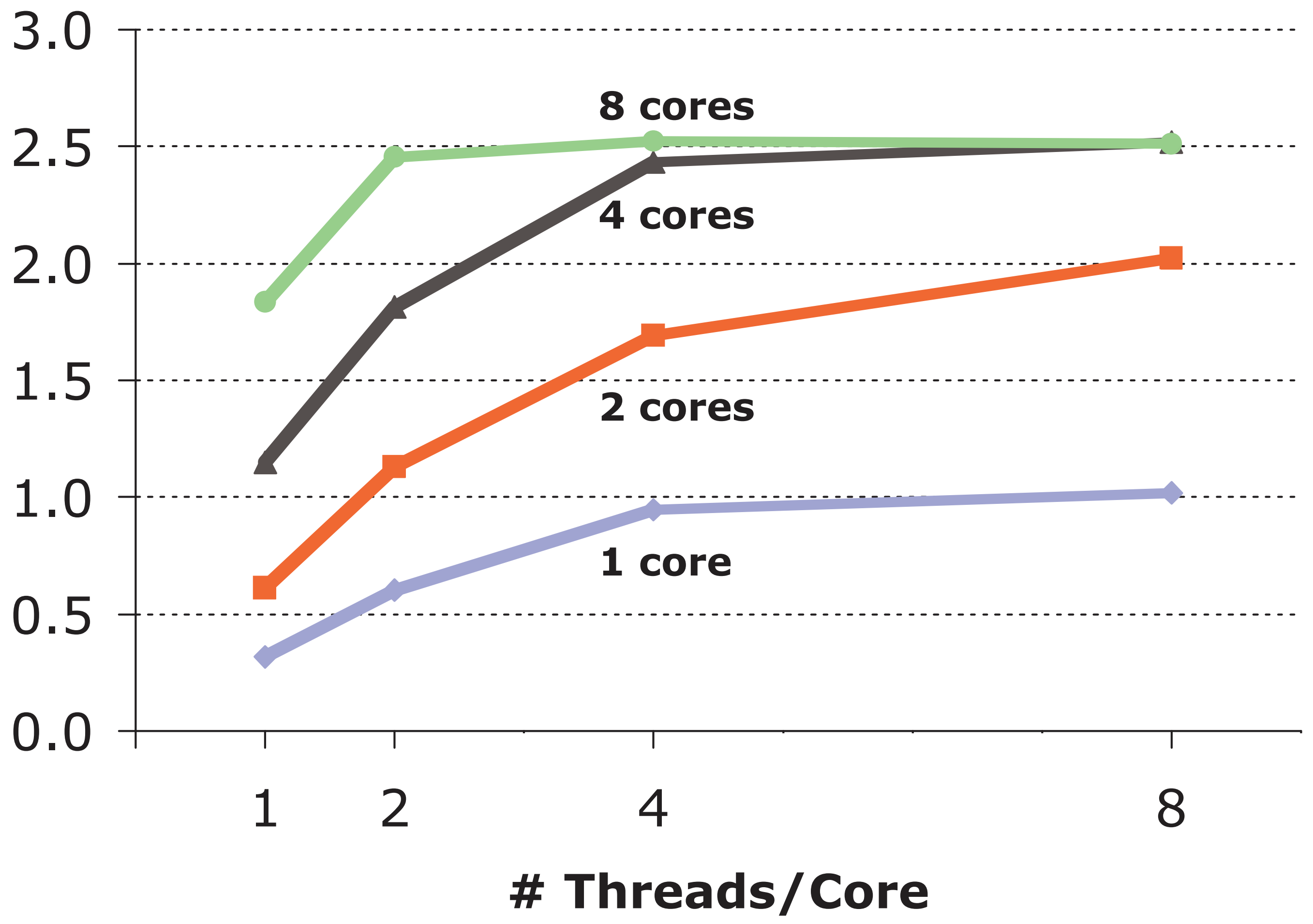
Hash Join Setup

- Model ‘probe’ phase



- Assign pages of outer relation to one core
  - Each thread context issues one probe
  - Overlap dependent accesses within core

Hash Join Performance



Network processor finds MLP across tuples

Conclusions

- DBMS on modern processor
  - Need to exploit memory parallelism
- Require thread-level parallelism
  - Multi-threaded, multi-core architecture
  - Hide memory latency
- Evaluation using network processors
  - Simple hardware, lots of threads
  - Beat Pentium 4 by 2X-2.5X on DB operators

Thank You!

Backup Slides

What about Niagara?

- 8 cores, 4 threads each
- Originally targeted network applications
- Sound familiar? Some differences:
    - Larger L1/L2 caches
    - More friendly programming environment (?)
    - Less programmer control (?)
    - Larger memory bandwidth/parallelism
    - Availability?