CS 451 / 551 / ECE 541

ADVANCED
COMPUTER ARCHITECTURE

SESSION no. 23

- ILP
- DLP
- TLP

PROCESS - "CONTEXT", OWN PRIVATE MEMORY SPACE.

- PROC. SWITCH: 100 - 10,000 CLOCK CYCLES

THREAD - SHARE CONTEXT & MEMORY AMONG THREADS IN A PROCESS

- THREAD SWITCH: 1-2 CLOCK CYCLES

THREAD BENEFITS

1. CONCURRENCY (WITHOUT PARALLELISM)
   - ONE THREAD STALLS, RUN ANOTHER
     - IMPROVES RESPONSIVENESS
     - KEEPS RESOURCES BUSY
       COMPUTING.
     - ACCOMODATE SHORT STALLS:
       CACHE MISS

NEEDS SUPPORT

- HARDWARE - ~~FAST~~ FAST SWITCHING
- O/S - & THREAD SCHEDULING
- ~~FOR~~ PROGRAMMER - LANGUAGE - COMPILER
  - SYNCHRONIZATION ...

2. SIMULTANEOUS MULTITHREADING

- MULTIPLE EXECUTION UNITS
- THREADS SHARE PARALLEL PATHS THRU PROCESSOR
- GRANULARITY: INSTRUCTION
- PARALLELISM

SUPPORT

- HARDWARE - "ISSUE SLOTS", CONTROL
- O/S - SCHEDULING
- PROGRAMMER + ...

3. MULTIPLE PROCESSORS - MIMD

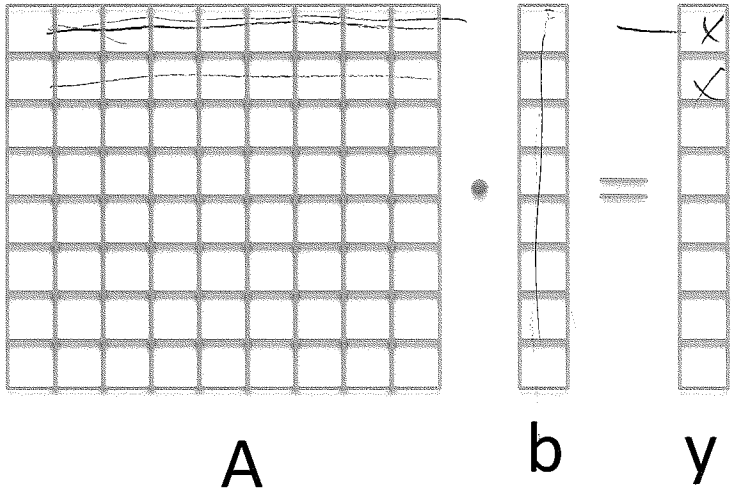
a. CENTRALIZED, SHARED MEMORY
- UNIFORM MEMORY ACCESS (TIME) (UMA)
- "MULTICORE" - ON CHIP

b. DISTRIBUTED MEMORY
- NONUNIFORM MEMORY ACCESS (TIME) (NMA)
- INTERCONNECT NETWORK
  - MEMORY NEAR PROCESSOR - FASTER ACCESS

Data Parallel Example with Threads

$$A \cdot \bar{b} = \bar{y}$$

$m \times n$ Matrix-Vector Multiplication



Thread 1: $[a_{11}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_1$

Thread 2: $[a_{21}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_2$

Thread 3: $[a_{31}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_3$

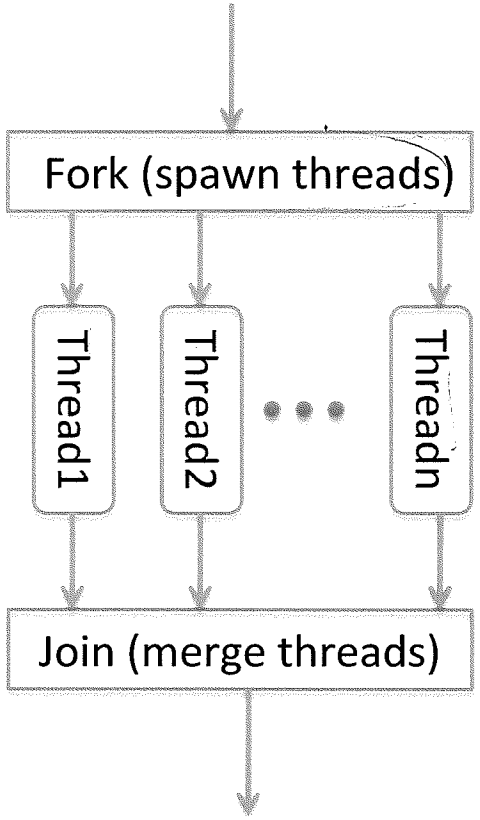
⋮

Thread n: $[a_{n1}\text{-}a_{nn}][b_1\text{-}b_n]^T = y_n$

We can assign one *process thread* to compute each element of y.

Each process needs:

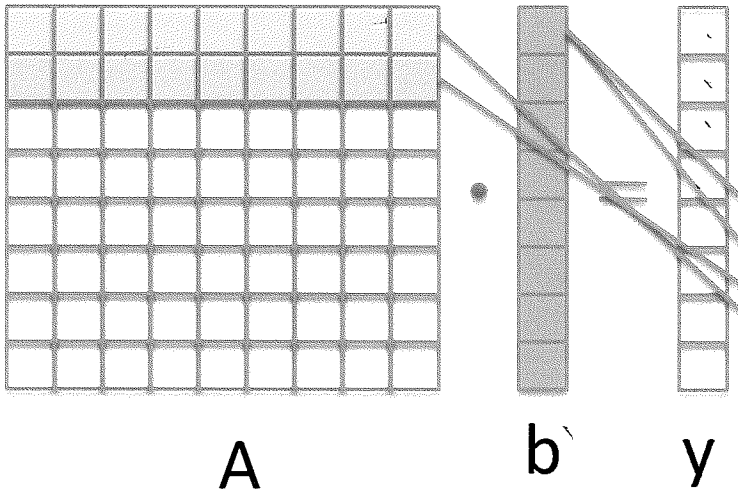
- Access to (a copy of) a row of A
- Access to (or a copy of) of b



Threads Example

$A \cdot \bar{b} = \bar{y}$

$m \times n$ Matrix-Vector Multiplication

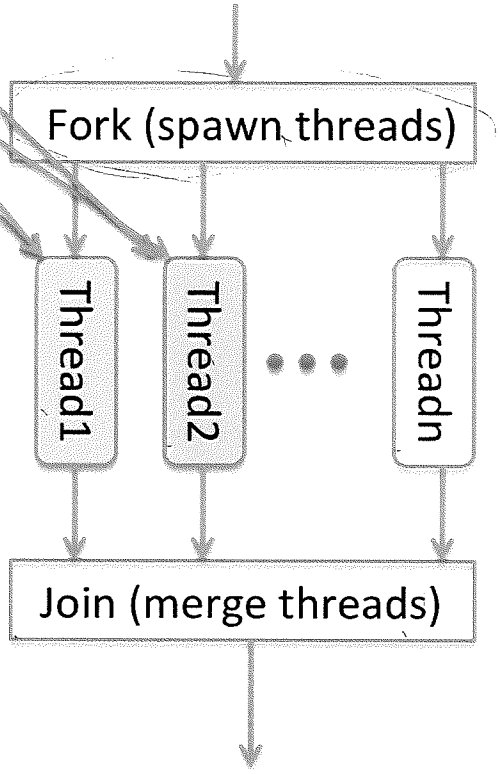


Get it going:

- In Pthreads:
- for (i=1; i<=n; i++) { create_thread(…) }

How do we know when we are done?

- When all of the threads have completed
- In Pthreads: pthread_join()



Thread 1: $[a_{11}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_1$

Thread 2: $[a_{21}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_2$

Thread 3: $[a_{31}\text{-}a_{1n}][b_1\text{-}b_n]^T = y_3$

⋮

Thread n: $[a_{n1}\text{-}a_{nn}][b_1\text{-}b_n]^T = y_n$

University of Idaho

CHALLENGES IN THREAD-LEVEL CONCURRENCY

1. PARTITION DESIGN OF S/W INTO LOGICAL THREADS THAT CAN BENEFIT FROM CONCURRENCY.
2. IMPLEMENT IN LANGUAGE THAT SUPPORTS THREADS
3. THREAD SYNCHRONIZATION

PROBLEM
- MEMORY COHERENCY
- RACE CONDITIONS

MUTEX - MUTUAL EXCLUSION
- LOCK A RESOURCE UNTIL DONE
- "LOCK OUT" OTHER THREADS.

PROBLEM - DEADLOCK

- SITUATION IN WHICH TWO OR MORE COMPETING ACTIONS (THREADS) ARE EACH WAITING FOR THE OTHER TO FINISH, AND THUS NEITHER EVER DOES.

CS 411/511	PARALLEL PROGRAMMING
CS 412/512	PARALLEL ALGORITHMS
CS 513	CONCURRENT SYSTEMS

BIG CHALLENGE - CACHE COHERENCY

- VIRTUALLY ALL PROCESSORS USE CACHE
  - MULTIPLE LEVELS
  - DIFFERENT CACHES HOLD DIFFERENT VERSIONS OF SAME DATA
    - WHICH ONE IS CORRECT?

MEMORY MODEL

- MULTIPLE PROCESSORS "SEE" SAME ADDRESS SPACE

IEEE COMPUTER MAG OCT '13
SPECIAL ISSUE

COHERENCE - WHAT IS WRITTEN & READ

CONSISTENCY - WHEN IS IT WRITTEN & READ (ORDER)

EACH PROCESS SHOULD "SEE" SAME ADDRESS SPACE
- SAME DATA
- CHANGES IN SAME ORDER

LEVELS OF COHERENCY

1. EVERY WRITE OPERATION APPEARS TO OCCUR INSTANTANEOUSLY
   - SEQUENCE IS PRESERVED

2. ALL PROCESSORS SEE SAME SEQUENCE OF CHANGES OF VALUES. (NOT NECESSARILY AT SAME TIME)
"STALE" DATA POSSIBLE

3. DIFFERENT PROCESSORS SEE A DIFFERENT SEQUENCE OF DATA
INCOHERENT - BAD!

CONDITIONS FOR COHERENCE

1. PROGRAM ORDER PRESERVATION

- P1 WRITES TO MEM LOCATION X, THEN P1 READS FROM X (AND NO OTHER PROCESSOR WRITES TO X)
  → P1 SHOULD READ WHAT IT WROTE.

EX P1 WRITES 'A' TO mem[x]
P1 READS mem[x], gets 'A'

2. COHERENT VIEW OF MEMORY

P2 WRITES TO LOC. X

WAIT "LONG ENOUGH"

P1 READS FROM X

→ P1 READS READS WHAT P2 WROTE

ex

P2 WRITES 'A' to mem[x]

P1 READS mem[x], gets 'A'

3. SEQUENCE

P1 WRITES 'A' TO MEM[x]

P2 WRITES 'B' TO MEM[x]

P3 READS FROM MEM[x]

P3 READS FROM MEM[x] AGAIN

→ P3 SHOULD SEE 'A', THEN 'B'

NEVER 'B', THEN 'A'

. IF P3 WAITS LONG ENOUGH,
IT MIGHT NOT SEE 'A';
IT SEES 'B'; THEN 'B'

COHERENCE MECHANISMS

SNOOPING - CACHES WATCH ADDRESS LINES FOR ADDRESSES THEY HAVE COPIES OF.

- IF ANOTHER PROCESSOR WRITES TO ONE OF THIS, IT INVALIDATES ITS OWN COPY.
- NEXT - GET A FRESH COPY READ

PRO: FAST - (IF BUS B/W IS AVAILABLE)

CON: NOT VERY SCALABLE

University of Idaho

DIRECTORY - "FILTER" THROUGH WHICH PROCESSORS LOAD A MEMORY BLOCK TO CACHE.

- WHEN A BLOCK IS WRITTEN, DIRECTORY INVALIDATES ALL OTHER CACHES THAT HAVE THAT BLOCK.

TELLS EVERYONE WHEN THEIR COPY IS OUT OF DATE.

PRO: SCALABLE

CON: SLOWER - MULTIPLE STATES FOR A TRANSACTION