Welcome to the Concurrent World

Why is your tablet faster than your old computer?

2016: Clock Frequency = 1.5GHz

2000: Clock Frequency = 1.5GHz
Welcome to the Concurrent World

Why is your tablet faster than your old computer?

Concurrent programs exploit multiple CPU cores.

2016: Clock frequency = 1.5 GHz

2000: Clock frequency = 1.5 GHz
Welcome to the Concurrent World

Why is your tablet faster than your old computer?

But programmers don’t like concurrent code

Concurrent programs exploit multiple CPU cores

Multiple CPUs

Single CPU

Clock frequency = 1.5GHz

2016: 1.5GHz

2000: 1.5GHz

But programmers don’t like concurrent code
Multi-threaded environment
Problem
• A slow thread may slow others down
• Block other threads while accessing resource

Lock
Readers use current copy without noticing

Writers create and update a resource copy

Copy on Write (COW)
Compare-And-Swap (CAS)

- Switch from one version to another
- Allow the system to always make progress
  - If I cannot switch, then someone else could!
  - Otherwise, I proceed
Threads try to access resource:

- If conflicts cannot be resolved, then abort.
- Otherwise proceed and commit.
New Paradigms
New Paradigms

- Minimize contention, postpone restructuring
  - Pb: The growing core count induces contention
  - Idea: Relax structural invariants during load bursts
  - Level distribution, O(log n) depth, load factor

New Paradigms

ICDCS'13, PPoPP'14a, PPoPP'14c

Contention-Friendliness [PPoPP'12, EuroPar'13]

Tree

Tree

Tree

Tree

Tree
New Synchronizations
New Synchronizations

- Combining in the same application
- Multiple transaction semantics

Polyomorphic TM [DISC’09, CACM’14, ECOOP’14]

- Shares similarities with CopyOnWrite
- Writers copy the data to be modified
- Readers proceed uninterrupted

Read-copy-update [ASPLOS’12, PODC’14]
New Synchronizations
• A micro-benchmark suite in Java and C++

A+micro<benchmark+suite+in+Java+and+C/C+++

Goal: to evaluate on a common ground

− Data structure algorithms
− Synchronization techniques

Abstractions: Set, Queue, Map

Available:

− Open-source (GPLv3/Apache)
− Online: https://github.com/gramoli/synchrobench

V. Gramoli - ANU 2016
How does it speed up sequential performance?
Are k subsequent iterations part of k JVM instances?
Does 50% update mean modification 25% of the time?

Micro-benchmarks are appealing but customized
Profiling tools are architecture specific
The causes of data structure bottlenecks
Macro-benchmarks cannot help at nailing down

Synchrobench [PPoPP’15]
Status

40+ concurrent structures in 2 languages (Java and/or C/C++)

1. Compare-and-Swap (non-blocking)
2. Locks (eager, lazy acquisition): macro with pthread-shouldlock in C/C++ by default
3. Compare-and-Swap (non-blocking)
4. Read-Copy-Update (user-level)
5. Read-Copy-Update (user-level)

- Most already open sourced (interface was simply adapted)
- We redeveloped few from scratch
- 4 required changes in Synchronbench threading model
- 1/3 are recent (published in the 2012-2015)
- 1/3 are recent (published in the 2012-2015)

Transactional memory (software libraries)
Parameters

- d, duration (ms)
- U, unbalance rate
- A, alternated values
- f, effective update
- u, update ratio
- r, value range
- i, initial size
- t, thread number
- a, ratio of write-all ops
- s, ratio of snapshots
- b, benchmark (algo.)
- n, iterations
- W, warmup (s)
- Java specific

...
(each point is the average throughput of 5 runs)

- Intel: 32-way Xeon E5-2450 machine
  - 2 sockets of 8 hyperthreaded cores at 2.165GHz

- AMD: 64-way Opteron 6378 machine
  - Fedora 18, Java 7.7.0-09 IcedTea and gcc 4.7.2.
  - 4 sockets of 16 cores each running at 1.46GHz

- Sparc: 64-way UltraSPARC T2
  - Solaris 10, Java 7.0-05 Bos Hotspot and Sparc-Sun-Solaris2.10-gcc 4.0.4
  - 8 cores of up to 8 hardware threads running at 1.166GHz

- Tilera: 144-way TILExTreme
  - Linux kernel 3.10, Java 7.6, gcc 4.4.6
  - 4x 36 cores running at 1.2GHz

- AMD: 64-way Opteron 6378 machine
  - 2 sockets of 8 hyperthreaded cores at 2.165GHz

- Intel: 32-way Xeon E5-2450 machine
  - 2 sockets of 8 hyperthreaded cores at 2.165GHz

- Sparc: 64-way UltraSPARC T2
  - Solaris 10, Java 7.0-05 Bos Hotspot and Sparc-Sun-Solaris2.10-gcc 4.0.4
  - 8 cores of up to 8 hardware threads running at 1.166GHz

- Tilera: 144-way TILExTreme
  - Linux kernel 3.10, Java 7.6, gcc 4.4.6
  - 4x 36 cores running at 1.2GHz
Scalability

Intel

Parameters: u[0..50], f1-15000-50-40, [16384..65536], [32768..132072], W0

Skip lists
Linked lists
Hash tables
Binary trees
Scalability

Parameters: \([0.05, 0.15, 0.4, 0.65] \times 10^4\) - \([5000, 15000, 40000] \times 10^3\) - \([132072, 32768, 65536] \times 10^2\) - W0

Intel

Categories:
- Skip lists
- Linked lists
- Hash tables
- Binary trees
Overall CAS helps boosting perf, except for trees.

Scalability
Complexity of Non-Blocking Trees

- Some do not rebalance [PODC'10]
- Some had data races [PPoPP'14b]
- Some are proprietary code [ICDE'13]
- Overall memory reclamation and lock-freedom is hard [CACM'13]
- Complexity of Non-Blocking Trees

V. Gramoli - ANU 2016
Thread pinning

SCAT: scatter pinning policy (different NUMA node first)

COMP: compact pinning policy (same NUMA)

- \( f_{[132768, 132072]} \) - W20
- Parameters: ut0-T1-15000-50-0-16384.65536

AMD
Thread pinning

- SCAT: ok for scalability in TM
- COMP: ok for rapid peak perf.

COMP: compact pinning policy (same NUMA

SCAT: scatter pinning policy (different NUMA

Node first)

| Parameters: u10-15000-50-30-16384-65536 |
| AMD [132768-132072]-W20-16384-65536 |
Thread pinning

- **COMP**: ok for rapid peak perf.
- **SCAT**: ok for scalability in TM
- **RCU**: may not tolerate contention as well as TM

- COMP: compact pinning policy (same NUMA node first)
- SCAT: scatter pinning policy (different NUMA)

- Parameters: U10-F1 [15000-50-0] [16384..65536]
- AMD T32768 [132072-2]-W20

Thread pinning
Reusability

Parameters: u[0.10]-F1-W20-15000-50.0-1.024

SPARC
Only few techniques to ensure composition + extensibility
Only few techniques to ensure composition + extensibility

- COW may not tolerate contention as well as TM
Power Efficiency

Lock-free hash table benchmark

Manycore: Tilera Tile-Gx, 36 cores, 40 nm, 1.2 GHz, 28 W TDP
Multicores: Intel Xeon, 32 cores, 32 nm, 2.1 GHz, 95 W TDP
Multicores-SE: Tilera Tile-Gx, 36 cores, 40 nm, 1.2 GHz, 28 W TDP
Multicores-EGX: Tilera Tile-Gx, 36 cores, 40 nm, 1.2 GHz, 28 W TDP

Number of threads

Operations per Joule

K ops per second

2^16 elmts, 10% upd
Conclusions

• CAS+ is a double-edged sword
  – Helps boosting performance
  – Difficult to use efficiently
  – Helps boosting performance

• CAS is a double-edged sword

• Concurrent alg. can reach higher performance over
  – Terrible under contention
  – Good for read-only execution
  – Suffer more from contention than RCU/COW

energy ratio on manycores than on multicore
Conclusions

Research:

- Yahoo! Labs, Israel
- Tel-Aviv, Israel
- ETHZ, Switzerland
- EPFL, Switzerland
- Neuchatel, Switzerland
- MIT, USA
- Aarhus, Denmark
- Tromso, Norway
- Technion, Israel
- INRIA, France

Teaching:

- Politecnique, France
- MIT, USA
- ENST, France

1/02/16
Future Work

- Online courses at USyd
- Winter School in March
- Integration to an autograding system for teaching
- Hardware Transactional Memory
- Porting Syncrabortench to Power8 and BlueGene/Q
Any help is welcome
References