# Transactional Memory
# Architecture and Implementation for IBM System z

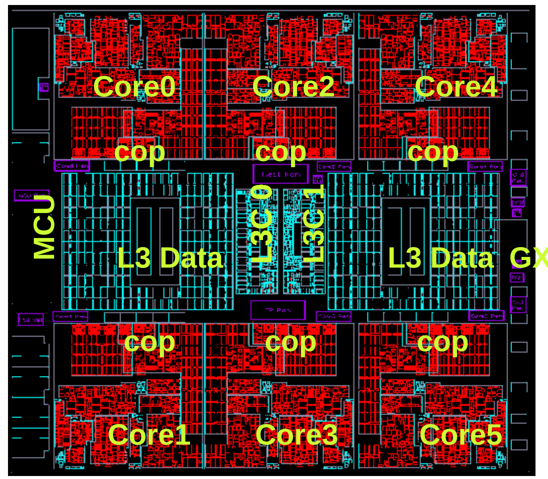
Christian Jacobi, Timothy Slegel, Dan Greiner
IBM Systems and Technology Group

# Transactional Memory in System z


- Transactional Execution introduced in latest mainframe generation: zEC12
- First commercially available TM implementation in general-purpose CPU
- Available since September 2012

- zEC12 is descendent of original System 360 mainframe from 1964
- System z design is focused on
  - performance
    5.5GHz, out-of-order 6-way superscalar CISC processor
  - scalability
    101 customer CPUs, rich SMP-fabric, and large caches
  - availability
    99.999% availability, through stack-optimization for RAS
  - virtualization
    up to 60 partitions and thousands of virtual images

- targeting mission critical computing for commercial workload
  - Transactional & analytical database workloads
  - SAP, Java, Websphere, …
- IBM System z is IT backbone of many of the worlds largest corporations



# Transactional Execution
# Requirements for introduction into commercial computer system

- Ease of use → simple semantics
  - System z has “strong memory ordering” architecture, easing parallel software design
  - Limit exposure of micro-architectural behavior at architecture level
- Stepwise introduction into existing software
  - millions of lines of code won’t be rewritten in a year
  - transactions need to co-exist with lock-based serialization
- Future compatibility
  - Software cannot be adjusted for every generation of hardware
- Debugging & RAS
  - parallel programming is hard – cannot make it harder with debugging and reproducibility problems
- Introduction into existing CPU & Cache base design
  - CPU & SMP-fabric evolutionary designs, revolutions are too expensive & risky

# Transactions in System z

Transactions are regions of code starting with *TBEGIN* and ending with *TEND.*

- *atomicity*

  instructions within a transaction execute all or none

- *isolation*

  other CPUs or IO devices do not observe transaction's memory updates, and transaction does not observe memory updates by other CPUs or IO

  - isolation is *strong:* transaction is isolated against other CPU's non-transactional operations
  - *strong* isolation is imperative for realistic introduction of TX into existing software

  abort & rollback if isolation cannot be guaranteed

- *opacity: isolated in non-committed state*

  transactions that abort are *isolated* up to the point of abort

  - another key requirement for introduction into existing software
  - alternative *validate* instruction to limit "zombie transactions" is too complex

- *flattened nesting*

  the entire nest of transactions is rolled back on abort

  maximum nesting depth is 16

# Use Cases for Transactional Execution

Three main use cases considered during definition phase:

- **Transactional Lock Elision**
  - replace *lock acquire/release* with transaction to ensure isolation of critical code section

```
TRANSACTION BEGIN
IF LOCK!=0 THEN ABORT
.. perform critical section ..
TRANSACTION END
```

```
@abort:
IF (count < threshold)
  retry transaction
ELSE
  OBTAIN LOCK
  .. perform critical section..
  RELEASE LOCK
```

Benefits:

  - reduced cache miss latency for exclusive fetch of lock-word cache line
  - better scalability in false contention cases

- General code optimization
- Lock-free data structures

# Use Cases for Transactional Execution

Three main use cases considered during definition phase:

- Transactional Lock Elision
- **General code optimization**
  - enable more aggressive compiler speculation, exploiting atomicity & isolation
  - aggressively optimize dominant code path, moving rare code paths into the transaction abort path

```
IF (cond && C!=0)
  A=B/C
  STORE A, mem
ELSE
  ..
ENDIF
```

```
TRANSACTION BEGIN
A=B/C        ;; aborts if C=0
STORE A,mem ;; speculatively store
IF(!cond)
  TABORT
TRANSACTION END
```

- Lock-free data structures

# Use Cases for Transactional Execution

Three main use cases considered during definition phase:

- Transactional Lock Elision
- General code optimization
- **Lock-free data structures**
  - potential for rapid exploitation through common data structure libraries
  - transactions are easier / more powerful than Compare & Swap based algorithms
  - priority work queues, hash tables, double-linked lists, …

# Transaction Aborts

- Successful TBEGIN sets Condition Code to 0 and executes sequential stream
- Transaction-abort returns to instruction after TBEGIN, with CC /= 0
  - CC=2 → likely transient condition, recommend retry (with threshold)
  - CC=3 → likely persistent condition, recommend fallback path

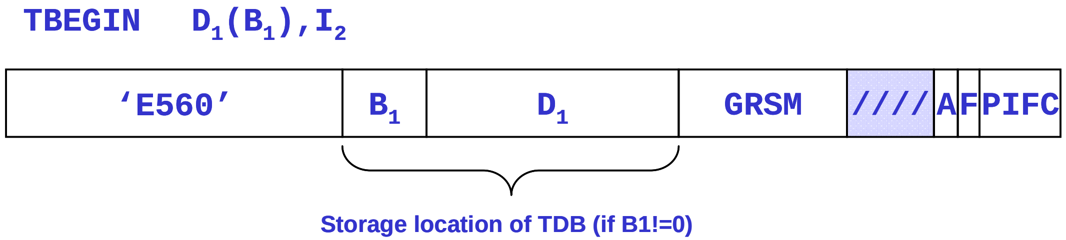
| Abort Code | Abort Reason | Condition Code |
|---|---|---|
| 2 | External interruption | 2 |
| 4 | Program interruption (unfiltered) | 2 or 3 |
| 5 | Machine-check interruption | 2 |
| 6 | I/O interruption | 2 |
| 7 | Fetch overflow | 2 or 3 |
| 8 | Store overflow | 2 or 3 |
| 9 | Fetch conflict | 2 |
| 10 | Store conflict | 2 |
| 11 | Restricted instruction | 3 |
| 12 | Program-interruption condition | 3 |
| 13 | Nesting depth exceeded | 3 |
| 14 | Cache fetch-related | 2 or 3 |
| 15 | Cache store-related | 2 or 3 |
| 16 | Cache other | 2 or 3 |
| 255 | Miscellaneous condition | 2 or 3 |
| >256 | TABORT instruction | 2 or 3 |

```
        LHI       R0,0              *initialize retry count=0
loop    TBEGIN                      *begin transaction
        JNZ       abort             *go to abort code if CC!=0
        LT        R1,lock           *load&test the fallback lock
        JNZ       lckbzy            *branch if lock busy
        ...perform operation...
        TEND                        *end transaction
        ...
lckbzy  TABORT                      *abort if lock busy; this resumes after TBEGIN

abort   JO        fallback          *no retry if CC=3
        AHI       R0,1              *increment retry count
        CIJNL     R0,6,fallback     *give up after 6 attempts
        PPA       R0,TX             *random delay based on retry count
        ... potentially wait for lock to become free
        J         loop              *jump back to retry

fallback
        OBTAIN    lock              *using Compare&Swap
        ...perform operation...
        RELEASE   lock
        ...
```

# Advanced TBEGIN features




## Register save/restore

- TBEGIN General-Register-Save-Mask (GRSM) specifies which registers to save/restore
- Access and Floating Point Registers (AR/FPRs) are never saved/restored
  - A/F flag to avoid unintended overwriting of registers inside transaction

## Program Interruption Handling

- Exceptions inside transaction abort and trap into OS
- Aggressive compiler optimization can lead to speculative program interruptions (unchecked pointers, unchecked data exceptions, etc)
- *Program Interruption Filtering Control (PIFC):*
  - Program can specify which classes of interrupts not to report to OS
  - Transaction still aborts, but without OS interrupt handler

# Constrained Transactions

- Most transactions are expected to be short and touch few memory locations
  - in particular lock-free data structures, and many Java synchronized blocks (lock elision)
- Having to code & test fallback path is significant burden on software design & verification

```
TBEGINC    *begin constrained transaction
...perform operation...
TEND       *end transaction
...
```

- *Constrained Transactions*
  - always eventually complete
  - start with TBEGINC
  - no fallback path, CPU goes back to TBEGINC after abort

- List of constraints must be met, otherwise Constrained Exception detected
  - maximum data footprint of 4 x 32 Bytes
  - maximum of 32 assembler instructions, no backward branches (i.e. no loops)
  - maximum code footprint of 256 bytes, non-overlapping with data footprint on 4K pages

# Software RAS & Debugging Features

- Software RAS & debugging are essential for enterprise class hardware & software
- Transactions pose serious problems
  - state (partially) rolled back at abort → new corner cases with aborts at different points
  - limited  visibility of why abort occurred (e.g. access exception)
  - no ability to instruction-step through transaction

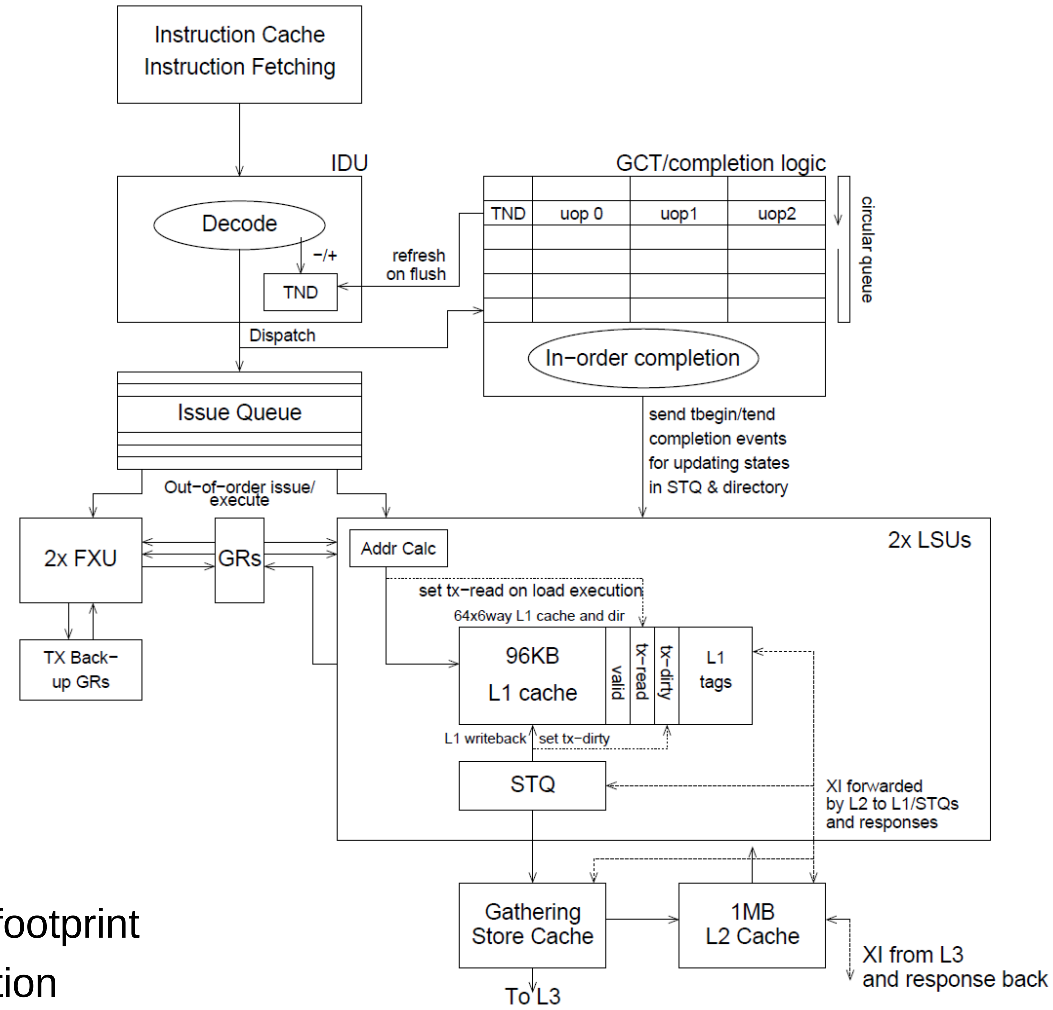
- TBEGIN can have *Transaction Diagnostic Block* parameter to store abort information
  - detailed abort code & hardware-internal abort reason
  - instruction & conflict address
  - General Register content before abort

- Enhancements for break- & watch-points
  - programmable to suppress events inside transactions, trigger at TEND

- Transaction Diagnostic Control
  - force random aborts to increase code verification coverage

# Transaction Cache Management

- L1-Data Cache Unit tracks transactional footprint
- new state bits for each 256 byte cache line
  - tx-read bit set during execution
  - tx-dirty bit set during L1 write back
- Gathering Store Cache buffers stores before L2
  - 64 entries x 128 byte capacity
  - CAM compare for gathering
  - circular queue for write back
- Abort handling
  - clear pending stores from STQ
  - clear all tx stores from Gathering Store Cache
  - clear dirty cache lines from L1
- L1/L2 track MESI-invalidations from L3 against TX footprint
  - reject:"stiff-arm" other CPU to try finish transaction
  - Abort transaction after reject thresholds

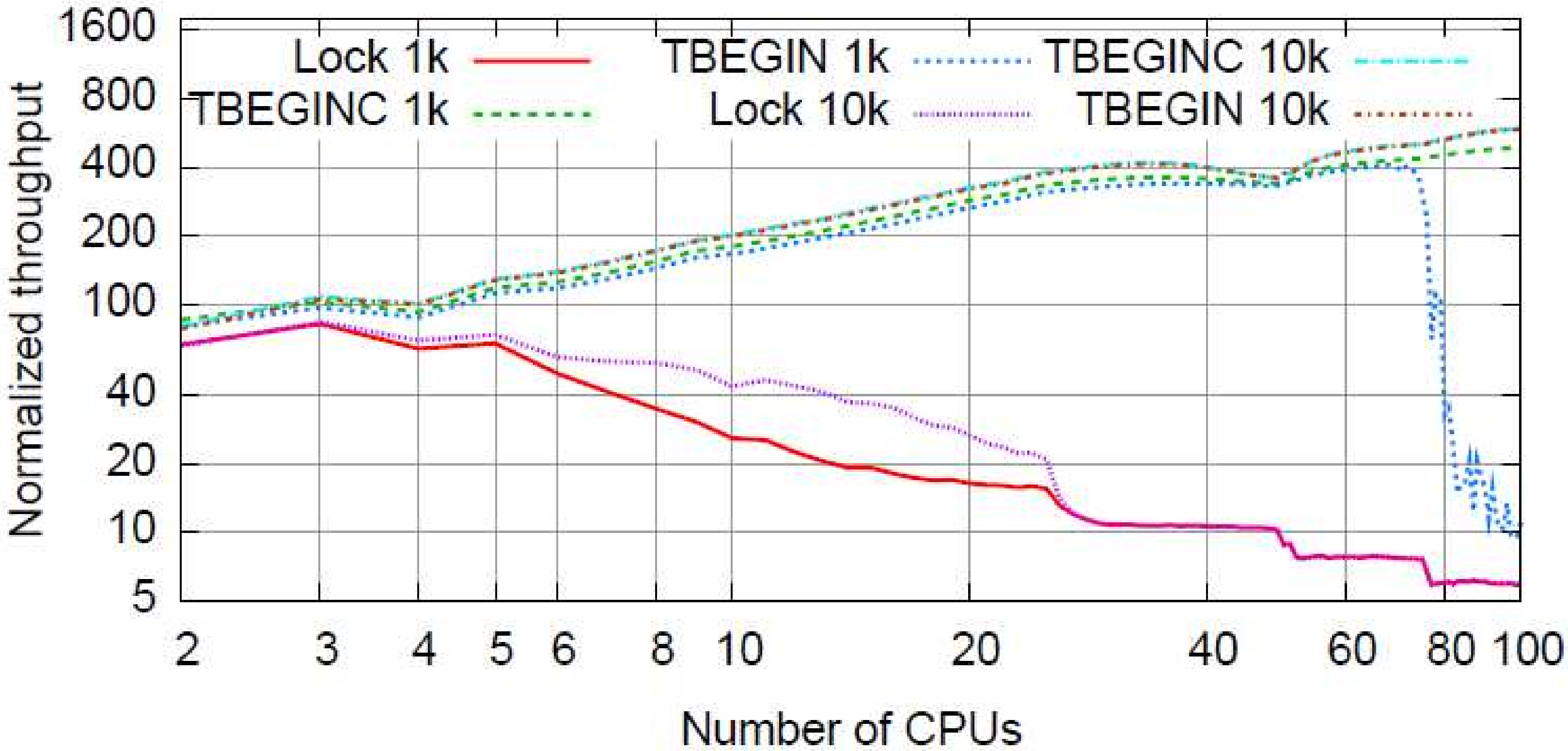


# Implementation of Constrained Transactions

- Most constrained transactions complete on 1st or 2nd attempt without special handling
- Elaborate mechanisms implemented to guarantee eventual progress on repeated aborts
  - artificial instruction streams, e.g. self-modifying code around transaction
    - step-wise reduction of pipeline speculation in CPU
    - disable branch prediction, out-of-order execution, pipelining
  - unnaturally high contention or artificial instruction streams
    - disable speculative fetching
    - introduce random delays, tailored to specific machine circumstances
    - interlock conflicting processors in LPAR or system-wide
- Escalation modes controlled by millicode, adjust with number of aborts & system specifics
- Lab bring-up found very interesting harmonics, with as few as 2 and as many as 100 CPUs

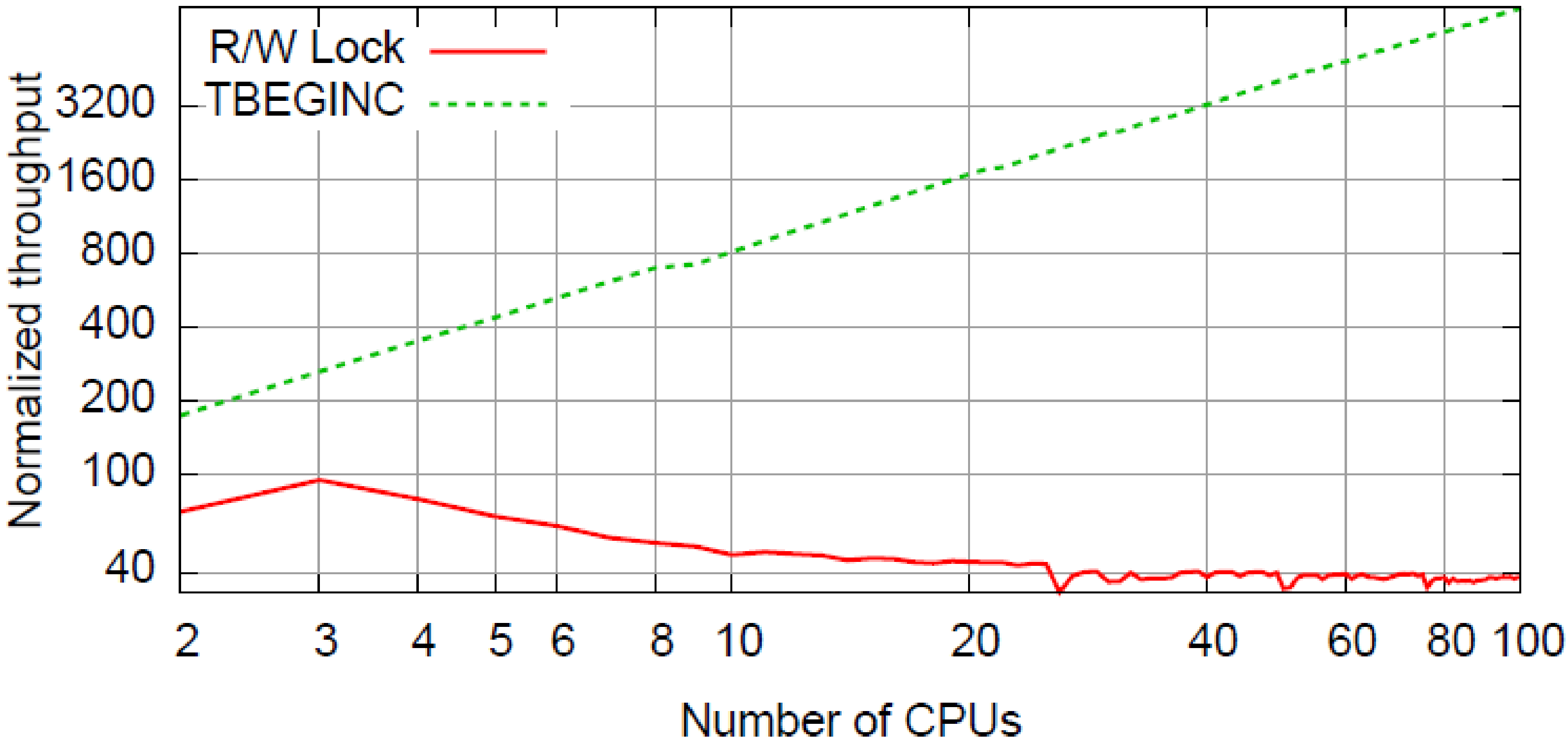
- Similar machine-specific delays are available for non-constrained transactions
- *Perform Processor Assist (PPA-TX)* instruction
  - introduces random delay optimal for abort count and system specifics
  - prevents adjustments of software to CPU generation or system size

# Performance Measurements




(a) TX vs locks, four variables, poolsizes 1k/10k

# Performance Measurements



(d) TX vs read-write lock, four variables read, poolsize 10k

# Summary

- System z is first commercially available general-purpose platform for Transactional Memory
- Architected with software reliability, debug ability, and future compatibility in mind
- Implementation without disrupting existing CPU and SMP micro-architecture
- Micro-benchmarks show scalability potential versus fine and coarse grained locks

- Software team finding more and more exploitation scenarios:
  - IBM XL C/C++ compiler support
    - published comparison of pthread locks vs transactions on subset of STAMP
  - z/OS Real Storage Manager lock avoidance
  - IBM Java ConcurrentLinkedQueue using Constrained Transactions
    - near-linear speed-up with number of CPUs
  - future IBM Java using lock elision
  - future IBM compilers to exploit general code reordering optimization using TX

- TX very promising tool to enable new wave of hardware & software co-innovation

# Thank You

Christian Jacobi

cjacobi@us.ibm.com

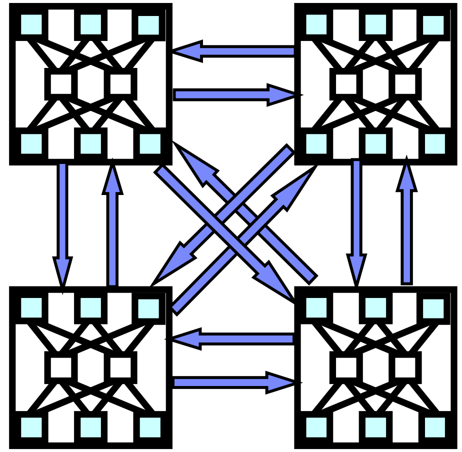
# zEC12 Design Overview

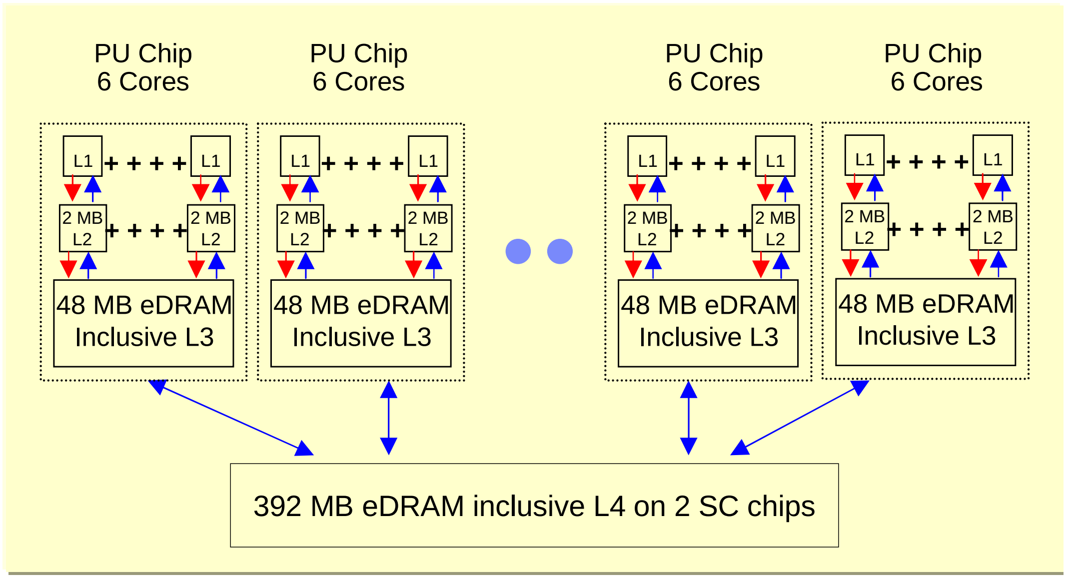
- 6 cores per chip CP chip
- 6 CP chips + 2SC chips per MCM
- Up to 4 MCMs

- Four level cache hierarchy
  - private L1/L2 store through 96KB+1MB data, 64KB+1MB itext
  - shared L3/L4 caches store-in 48MB+384MB
  - *Inclusive* caches: L1 ⊆ L2 ⊆ L3 ⊆ L4

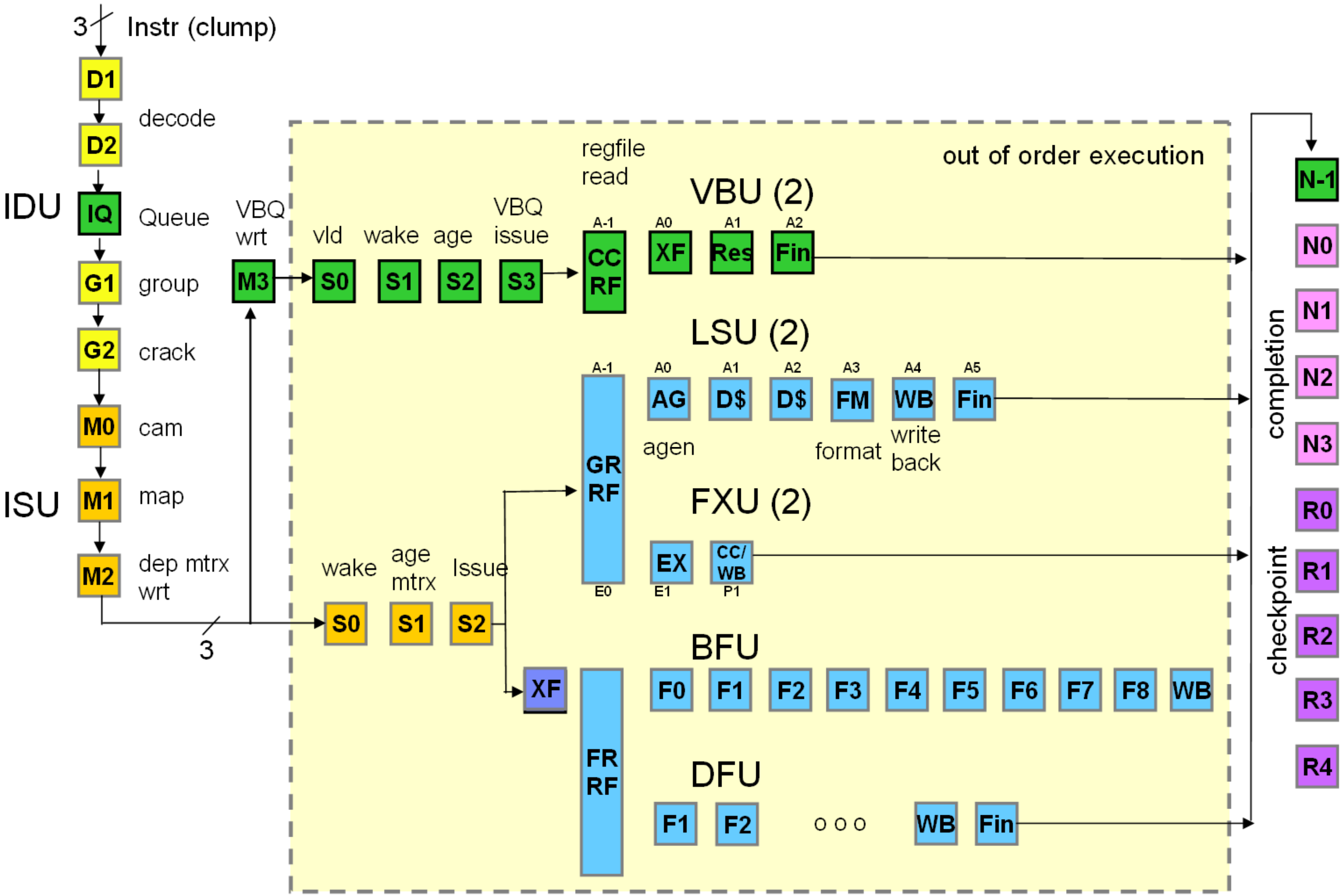
- variant of MESI protocol
  - “Cross Interrogate” (XI) between caches to invalidate cache lines
    - reject XI if hit against pending store
  - LRU-XI when higher-level cache evicts cache line


**Fully connected 4 Book system:**





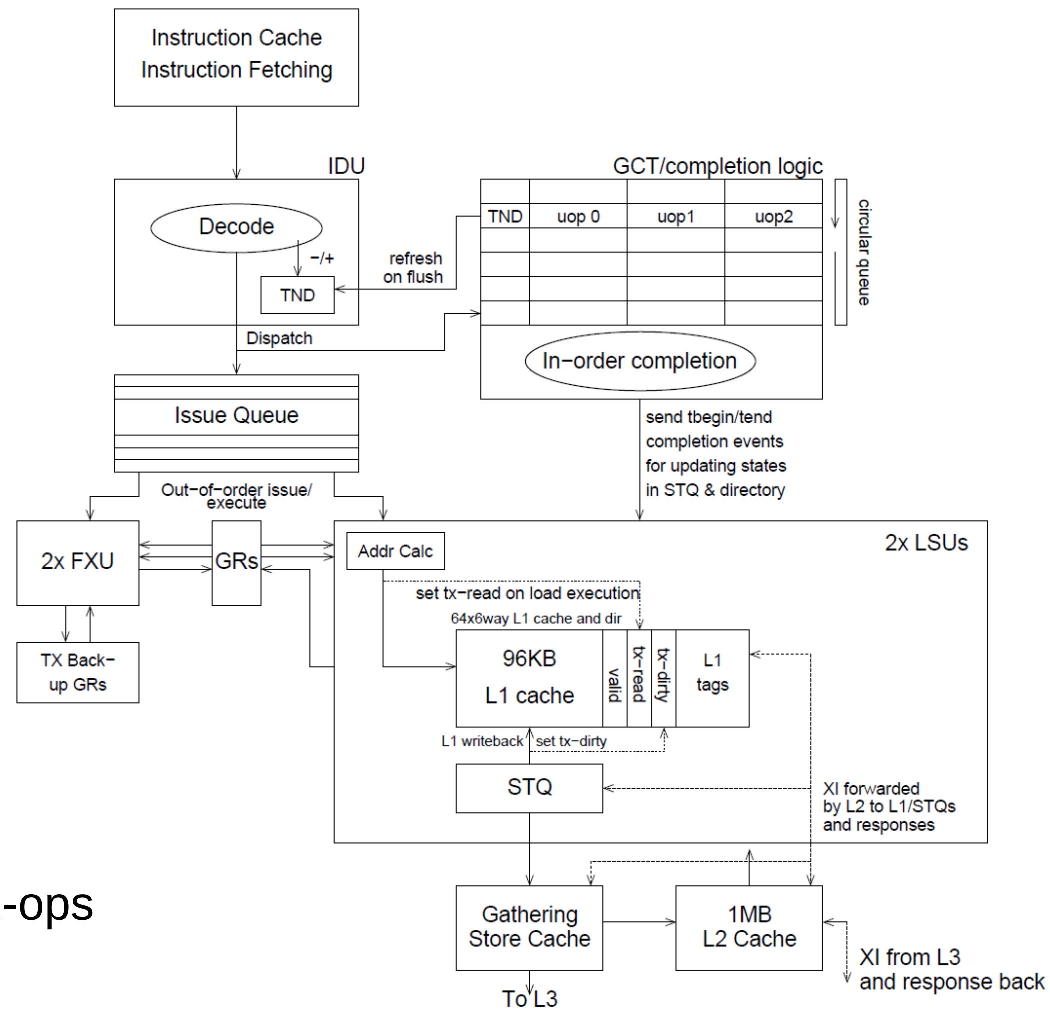
# Pipeline



3
Instr (clump)
D1
decode
D2
IDU
IQ
Queue
G1
group
G2
crack
M0
cam
ISU
M1
map
M2
dep mtrx
wrt
3
VBQ
wrt
M3
vld
wake
age
VBQ
issue
S0
S1
S2
S3
regfile
read
out of order execution
VBU (2)
A-1
A0
A1
A2
CC
RF
XF
Res
Fin
LSU (2)
A-1
A0
A1
A2
A3
A4
A5
AG
D$
D$
FM
WB
Fin
agen
format
write
back
GR
RF
FXU (2)
EX
CC/
WB
E0
E1
P1
wake
age
mtrx
Issue
S0
S1
S2
XF
BFU
F0
F1
F2
F3
F4
F5
F6
F7
F8
WB
FR
RF
DFU
F1
F2
o o o
WB
Fin
completion
checkpoint
N-1
N0
N1
N2
N3
R0
R1
R2
R3
R4

# Key components of TX implementation

- Instruction Decode Unit
  - detects & cracks TBEGIN/TEND
  - counts nesting depth, tracks A/F/PFIC
  - passes current tx-state into Issue Queue per µ-ops
  - detects restricted instructions and requests abort
- Fixed Point Unit
  - copies GPRs for restore on abort
  - read by millicode during abort-process to restore GPRs
- Global Completion Table
  - tracks transactional state per “group” of µ-ops
  - refreshes IDU on flush
  - updates “architected” TX state at TBEGIN/TEND completion



# Other instructions

| | |
|---|---|
| ▪ ETND | Extract Transaction Nesting Depth |
| ▪ NTSTG | Non-transactional store |
| | Gathering Store Cache has byte-mask of which bytes are transactional vs non-transactional. During abort, NTSTG's stores survive. |
| ▪ PPA | Perform random delay based on system characteristics. |

# Trademarks

**The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.**

| | | | |
|---|---|---|---|
| AIX* | FICON* | Parallel Sysplex* | System z10 |
| BladeCenter* | GDPS* | POWER* | WebSphere* |
| CICS* | IMS | PR/SM | z/OS* |
| Cognos* | IBM* | System z* | z/VM* |
| DataPower* | IBM (logo)* | System z9* | z/VSE* |
| DB2* | | | zEnterprise* |

* Registered trademarks of IBM Corporation

**The following are trademarks or registered trademarks of other companies.**

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license there from.
Java and all Java-based trademarks are trademarks of Oracle Corporation in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
InfiniBand is a trademark and service mark of the InfiniBand Trade Association.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

* All other products may be trademarks or registered trademarks of their respective companies.

**Notes**:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.