Core Dispatch and SMT Enablement in z13 processor

Fadi Busaba
Lisa Heller
Mark Farrell

System z CP Millicode Development
busaba@us.ibm.com
Trademarks

The following are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Those trademarks followed by ® are registered trademarks of IBM in the United States; all others are trademarks or common law marks of IBM in the United States.

For a more complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

*BladeCenter®, CICS®, DataPower®, DB2®, e business(logo)®, ESCON, eServer, FICON®, IBM®, IBM (logo)®, IMS, MVS, OS/390®, POWER®, POWER6+, POWER7®, Power Architecture®, PowerVM®, PureFlex, PureSystems, S/390®, ServerProven®, Sysplex Timer®, System p®, System p5, System x®, z Systems®, System z9®, System z10®, WebSphere®, X-Architecture®, z13™, z Systems™, z9®, z10®, z/Architecture®, z/OS®, z/VM®, z/VSE®, zEnterprise®, zSeries®

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 therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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.

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 Sync 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 Sync 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.
IBM z Systems

An integrated, highly scalable computer system that allows many different pieces of work to be handled at the same time, sharing the same information as needed with protection, handling very large amounts of information for many users with security, without users experiencing any failures in service.
Millicode in System z13

- Vertical microcode with its own internal architecture.
- Millicode resides in a protected area of storage called the hardware system area, which is not accessible to the normal operating system or application program.
- Millicode runs on the same hardware processor as customer software.
- Instructions inside Millicode are handled like any other architected instructions in the way they are fetched, decoded and executed.
- Processor is augmented with an additional micro-architected state called “Millicode mode" or simply Millimode.
- Millicode has the highest authorization level.
Millicode in System z13 - cont

• Millicode Architecture defines additional registers that are only visible to Millicode: milli-GPRs, milli-ARs, milli-CRs, Millicode instruction address register plus other Millicode registers

• Most architected instructions that are implemented in hardware are also available to Millicode

• The Millicode architecture includes additional instructions and registers that are not available in the external architecture

• There are Millicode Instructions that can move data to/from architected registers
Millicode in System z13 - Cont

Millicode augments the hardware to provide:

• System configuration functions
• System initialization functions
• Virtualization support for logical partitioning
• Complex instructions
• I/O functions
• Interruptions and other control functions
• RAS, Recovery, Logouts
• Instrumentation
Reasons for MilliCode Execution for Core Dispatch

- **Performance**
  - Best balance between PR/SM and core hardware/MilliCode. PR/SM manages the partition at core level. MilliCode/hardware synchronizes threads and maps logical threads to physical threads
  - MilliCode has high performance execution and knowledge and is well integrated with hardware

- **Authorization**
  - MilliCode runs with high(est) authorization

- **Flexibility**
  - MilliCode can adjust implementation as needed for performance and system integration optimization

- **Transparency**
  - Shields PR/SM logical core dispatch from underlying core hardware
Simultaneous Multithreading (SMT)
Simultaneous Multithreading (SMT)

- SMT enables to run multiple threads on a single core
  - Other processor families (i.e. x86, IBM Power processors, etc.) already have similar support
  - Each thread runs slower than a non-SMT core, but the ‘combined threads’ throughput is typically higher. The overall throughput benefit depends on the workload

- SMT can help improve overall capacity or throughput of the core

Which approach is designed for the highest volume** of traffic? Which road is faster?

**Two lanes at 50 carry 25% more volume if traffic density per lane is equal
Simultaneous Multithreading – Core hardware

Shared Facilities
- Core common control registers. Apply to both threads e.g. SMT controls, workaround logic.
- In space: active threads share Caches, Issue Queues, Physical Register Renames, etc.
- In time: active threads share Pipeline Slots, Execution Units, Address Translator, etc.

Dedicated Thread Resources
- Instruction buffers, instrumentation counters, etc.
- Thread architected state (Control registers, GPRs, ARs, FPRs, PSW/IA, etc.)
The z13 high-level instruction and execution flow.

- **Branch prediction**
- **Instruction cache / buffer**
- **Instruction decode/ crack / dispatch / map**
- **Issue queue side0**
- **Issue queue side1**
- **Branch queue**

**Additional Execution Units**
- LSU pipe 0
- LSU pipe 1
- FXU 0a
- FXU 0b
- FXU 1a
- FXU 1b
- BFU0
- BFU1
- DFU0
- DFU1

**New Registers / Execution Units**
- Vector0 / FPR0 register
- Vector1 / FPR1 register
- 128b string/int SIMD0
- 128b string/int SIMD1

**Data Cache**

Additional instruction flow for higher core throughput
Additional execution units for higher core throughput
New registers / execution units to accelerate business analytics workloads
The z13 Microprocessor pipeline and SMT operation.
Snapshot showing simultaneous execution of instructions from thread 0 and thread 1 in pipeline stages.
z13 - Simultaneous Multithreading (SMT)

- z13 is the first z System Processor to support SMT
  - Enable continued scaling of per-processor capacity
  - z13 supports 2 threads per core on Integrated Facility for Linux (IFL) and Integrated Information Processor (zIIP) processor cores

- It typically increases per-core and system throughput versus single thread design

- SMT can be turned on or off on an LPAR-by-LPAR basis by operating system parameters
  - Operating system must be explicitly enabled for SMT
  - The SMT switch is uni-directional

- Processors can run in single-thread operation for workloads needing maximum thread speed
  - Operating systems whose workload requires maximum thread speed may opt to run in a single-threaded hardware mode.
  - If a core is running multi-threaded but thread 1 is not being used by the operating system, the hardware can run in "effective single-threaded mode" to potentially improve performance of that thread.

- Functionally transparent to middleware and applications
  - No changes required to run in SMT partition
z13 Core Virtualization

- CPU Address changes with SMT
  - Sixteen bit CPU Id consists of a fifteen bit Core ID and one bit Thread ID
  - CPU ID 6 (b'0000000000000110') means core 3 Thread 0
  - CPU ID 7 (b'0000000000000111') means core 3 Thread 1

- On z13, z/OS will exploit SMT for zIIPs and z/VM will exploit SMT for IFLs
- On z13, only thread 0 is usable on any general purpose core
- SMT-aware operating systems (z/VM running as a hypervisor, or z/OS) must opt-in at an IPL to exploit SMT over the life of that IPL
  - Hardware makes both threads usable on each core
SMT Support Implementation

- CPU address expansion
  - Without SMT
    - CPU x0014 = 0000 0000 0001 0100
  - With SMT
    - Core x0014 thread 0 = 0000 0000 0010 1000 (CPU x0028)
    - Core x0014 thread 1 = 0000 0000 0010 1001 (CPU x0029)
  - Non-IFL or non-ZIIP processor odd address unavailable or unused
SMT-aware OS informs PR/SM that it intends to exploit SMT by opting-in

- PR/SM can dispatch any “guest” OS logical core to any physical core
- On dispatch of a logical core, PR/SM indicates to Millicode whether or not that OS has opted-in so that the hardware can be configured correctly
- OS control the whole core – must follow rules
  - Maximize core throughput (Drive cores with high Thread Density)
  - Maximize core availability (Meet workload goals using fewest cores)

SMT is transparent to applications
Logical Core Dispatch on an SMT2 Capable Physical Core

TID0 = thread0

Core dispatch on Physical TID0 (SIE instruction)

Phase 1: Execution- single threaded on Physical TID0

Phase 2: initialize core controls to enable execution of physical TID1

Phase 4: Physical TID0 maps and loads logical TID0 into physical TID0

Phase 5: Physical TID0 executes logical TID0’s instruction stream

Core dispatch execution ends on TID0

TID1 = thread1

Physical TID1 is idle (not used)

Phase 3: Physical TID1 initialized and ready

Phase 4: Physical TID1 maps and loads logical TID1 to physical TID1

Phase 5: Physical TID1 executes logical TID1’s instructions stream

Core dispatch execution ends on TID1

Core is in SMT1 execution.

Core is in SMT2 execution.
Millicode Implementation for Core Dispatch

TID0 = Thread0; TID1 = Thread1

Phase 1

1. Checks how many threads/guests are specified in the core dispatch.
   - In z13 it is a maximum of 2 threads.

2. Checks if there are enough available hardware threads
   - Physical threads may be running internal firmware code for functions like system management.

   Millicode can either wait for the threads to become available or nullify and retry the core dispatch instruction

3. Checks if there are any exception conditions preventing the core dispatch from proceeding

4. Decides on the logical threads to the physical threads mapping. i.e. logical TID0 to physical TID0.

5. Disable any interruption (including internal firmware interruptions) from being taken on all physical threads
Millicode Implementation for Core Dispatch

Phase 2

1. Updates core management controls e.g. controls for multithreading, partition number, TLB, etc.
2. Prepares TID1 for execution by updating its milli-IA (Millicode Instruction address) and by updating its state to be in Millicode
3. Enables TID1 to start executing to complete core dispatch on TID1

Phases 3/4

1. Millicode maps each logical thread into a physical thread, and each thread loads its own state into the hardware. A thread’s state includes PSW (Program Status Word), IA (Instruction Address), GRs (General Registers), FPRs (Floating Point Registers), CRs (Control Registers), etc.
2. Threads are not synchronized at the end of core dispatch execution routine. One thread may start executing the guest OS instruction stream while the other thread is still inside core dispatch Millicode

Phase 5

1. Physical threads, independently, run guest instruction stream
Exiting Guest Thread Execution when Running Two Guest Threads

Physical TID0 is executing logical TID0

Detects a reason to intercept to Hypervisor

Stops guest instruction execution. Enters Millicode routine to exit guest execution

Signals other thread to exit guest execution

TID0 saves the state of the guest (or logical TID0)

TID0 waits until TID1 responds and reaches the guest exit sync-up point

TID0 restores the hypervisor state back into physical TID0.

Guest thread execution end on TID0
Core un-dispatch execution ends

TID0 executes Hypervisor code

Physical TID1 is executing logical thread1

TID1 gets interrupted and enters Millicode routine to exit guest execution

TID1 saves the state of the logical TID1.

TID0 = Thread0
TID1 = Thread1

Core is in SMT2 execution.

Core is in SMT1 execution.

Guest thread execution ends on TID1
Physical TID1 is idle (not used)
Millicode implementation for Exiting Guest Execution

1. Millicode coordinates the exit (undispatch) from guest execution between the threads

2. An existing thread (Ta) enters the Millicode routine to exit guest execution
   - For this sequence, assume that Ta is running on TID0 and Tb on TID1

3. Exiting thread (Ta) signals the other dispatched thread (Tb) to exit guest execution

4. Ta saves its guest state (i.e. unmaps it from hardware resources) and waits for Tb to exit

5. Tb receives the interruption to exit and honors it at a “convenient” time. E.g., If Tb is in a long running instruction, it may defer taking the interruption until after the instruction completes execution

6. Tb enters the Millicode routine to exit guest execution, saves its state and syncs up with Ta which is already waiting in the guest exit routine sync up point

7. Ta and Tb save the rest of their states, and Tb (running on TID1) finishes its exit Millicode

8. Ta (running on TID0), in parallel, loads the hypervisor (PR/SM) state back into TID0 hardware

9. Ta updates core controls including SMT controls, and finishes its exit Millicode

10. PR/SM runs single threaded on TID0

11. TID1 is idle and not consuming core hardware resources
References


Thank You

Gracias!

Merci

Bedankt

Obrigado

Danke

شكراً

多謝

ありがとうございます

Thai

Nederlands

Brazilian Portuguese

German

Arabic

Simplified Chinese

Japanese

Traditional Chinese

Russian

Spanish

Hindi

Brazilian Portuguese

Danke

Danke

Danke

Danke