IBM OCR project

Workload Optimization on Hybrid Architectures

IBM T.J. Watson Research Center
May 4, 2011

Chiron & Achilles

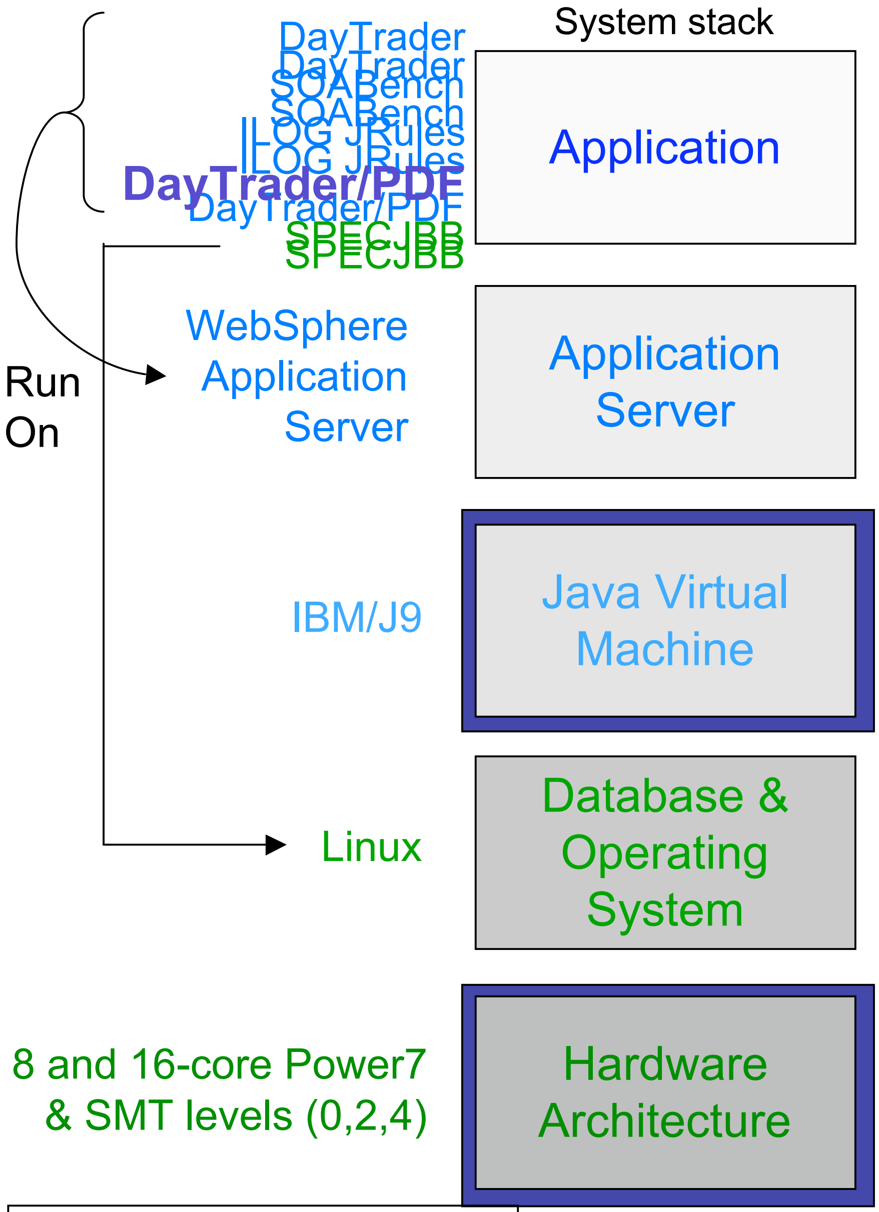
Goal

- **Parallelism with hundreds and thousands of threads**
  - Hardware is ready
    - Multi-core processor
    - IBM “POWER7 is designed for multi-socket systems that scale up to 32 sockets, which means that a full 32-socket system of 8-core parts would support 1024 threads.”
  - Software stack that is able to exploit/adapt to the parallelism provided by the hardware

Our practice

- **Our experience: locking/resource sharing has huge performance impact on hybrid systems/accelerators**
  - Focus on scalability/performance
- **Identify/configure the shared resources**
  - Hardware
    - Power7: L3 cache shared by sockets
  - Software
    - DB2 connections, JVM GC and JIT threads, WAS servant regions, thread pools
- **Identify/analyze performance bottleneck**
  - Tooling
    - Oprofile: profiling the whole system running on Linux
    - IBM WAIT Performance Tool: profiling JVM
    - JLM (Java Lock Monitor): profiling JVM lock access
    - Self developed LWT: profiling JVM JNI
  - Apply general practice of locking/data sharing

Study of scalability & lock contention on multicore/SMT sys



System stack

DayTrader
DayTrader
SOABench
SOABench
ILOG JRules
ILOG JRules
DayTrader/PDF
DayTrader/PDF
SPECJBB
SPECJBB

Application

Run On

WebSphere Application Server

Application Server

IBM/J9

Java Virtual Machine

Linux

Database & Operating System

8 and 16-core Power7 & SMT levels (0,2,4)

Hardware Architecture

Locks on

- Array lists (jfree chart, ILOG JRules)
- Date and calendar entry manipulations
- System.properties

Locking: no issues yet found
Scalability: many tuning parameters

- Synchronization on primitive data structures (hash tables, vector …)
- Synchronization deep in the JVM subsystems (GC, JIT)
- Lock effect visible when # of threads > 10

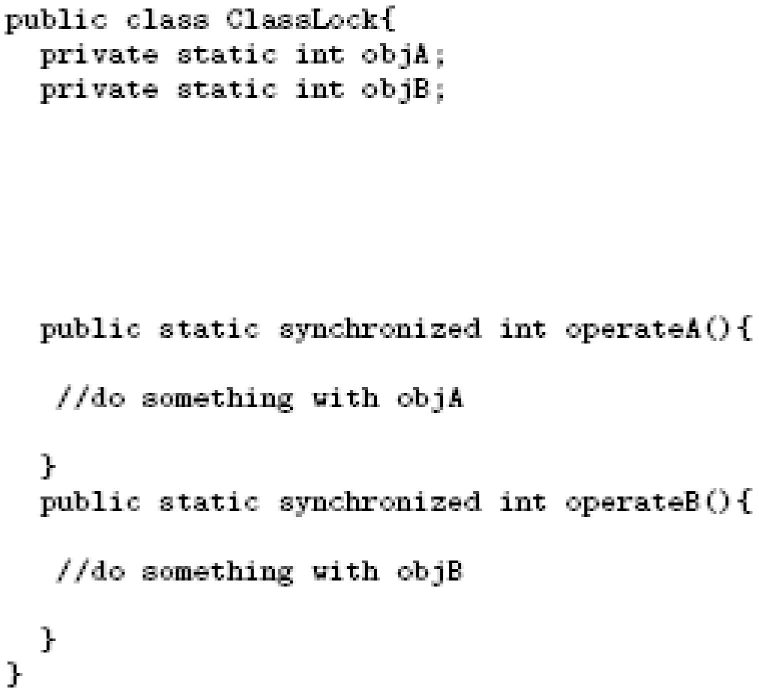
- OS -- JVM interaction: scheduling policy in Linux & JVM setting; e.g. kernel yield and JVM uses 3-tier locking
- OS – if not completely core/SMT aware and resulting issues in load-balancing

Multi-socket power systems: significant performance impact
•if JVM has threads on more than one socket
•If memory is allocated across banks

Experimental platforms

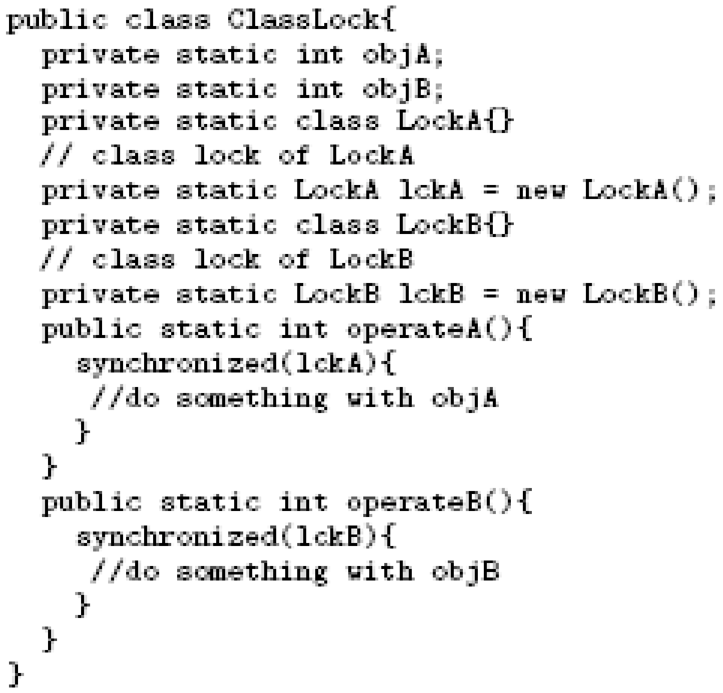
Findings

Exemplary class lock contention in JVM



```
public class ClassLock{
  private static int objA;
  private static int objB;








  public static synchronized int operateA(){

   //do something with objA

  }
  public static synchronized int operateB(){

   //do something with objB

  }
}
```

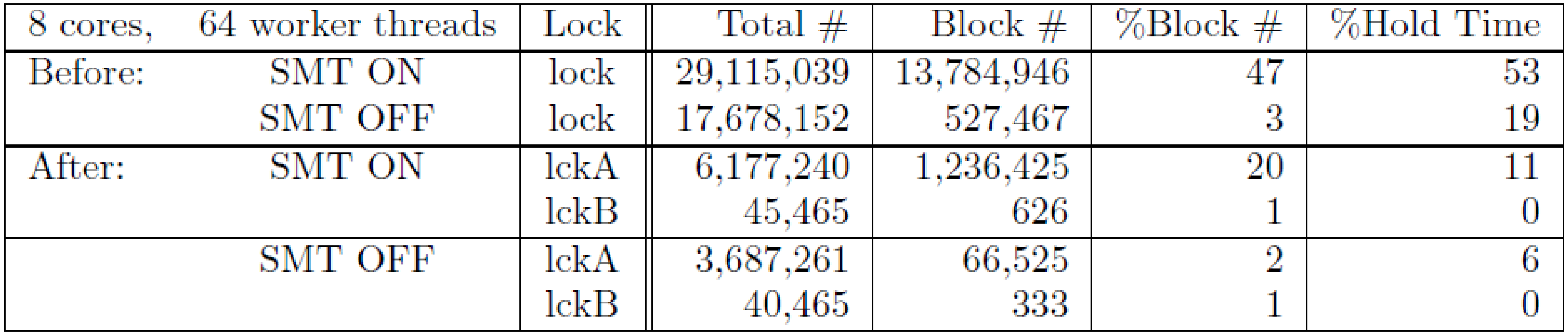
Figure 3. Example class ClassLock with a class lock



```
public class ClassLock{
  private static int objA;
  private static int objB;
  private static class LockA{}
  // class lock of LockA
  private static LockA lckA = new LockA();
  private static class LockB{}
  // class lock of LockB
  private static LockB lckB = new LockB();
  public static int operateA(){
    synchronized(lckA){
     //do something with objA
    }
  }
  public static int operateB(){
    synchronized(lckB){
     //do something with objB
    }
  }
}
```

Figure 4. Splitting locks in ClassLock

DayTrader/PDF



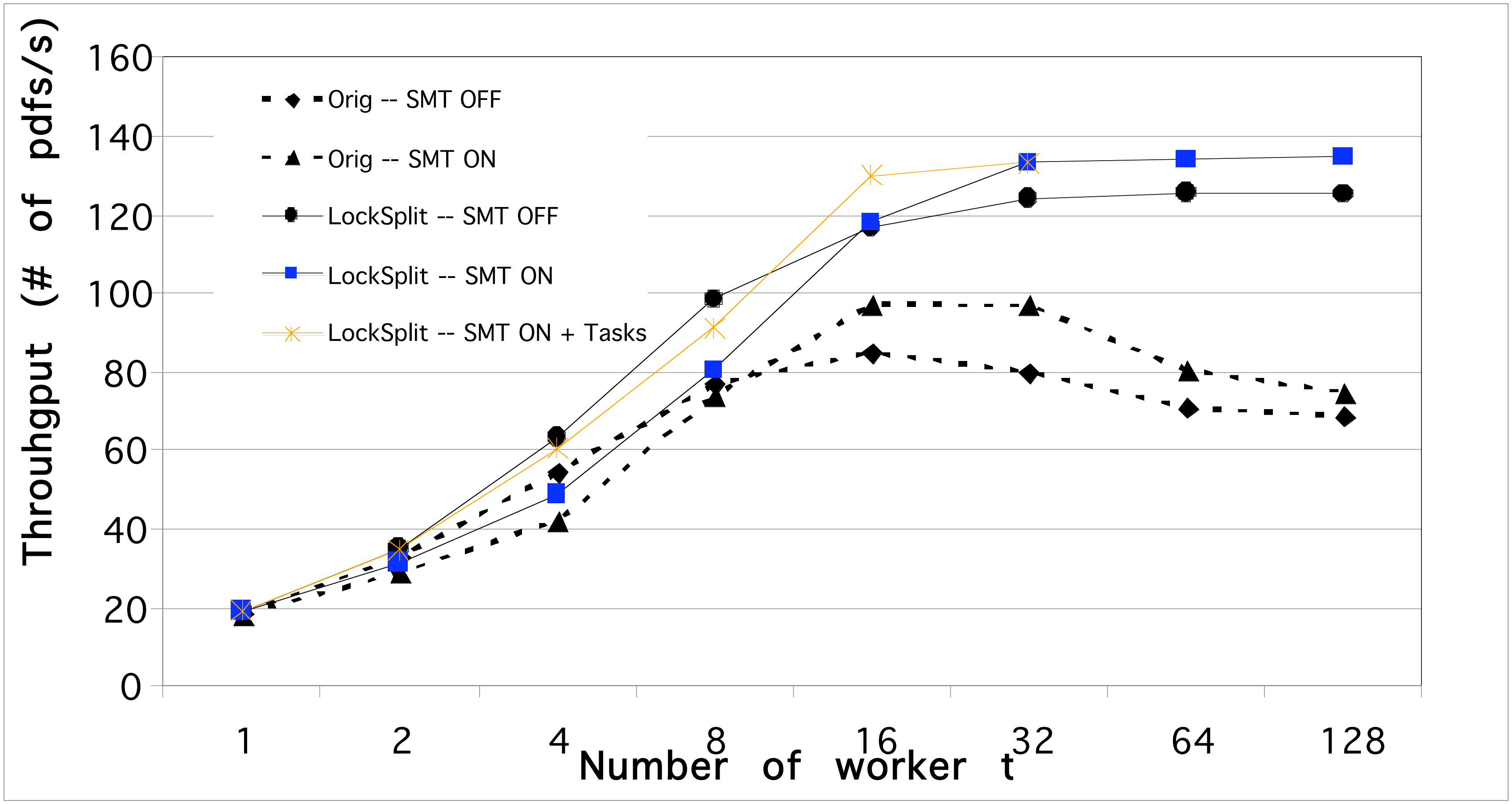
8 cores,	64 worker threads	Lock	Total #	Block #	%Block #	%Hold Time
Before:	SMT ON	lock	29,115,039	13,784,946	47	53
	SMT OFF	lock	17,678,152	527,467	3	19
After:	SMT ON	lckA	6,177,240	1,236,425	20	11
		lckB	45,465	626	1	0
	SMT OFF	lckA	3,687,261	66,525	2	6
		lckB	40,465	333	1	0

** *Improvement to concurrency of middleware will positively benefit most applications*

** *Concurrent programming: development & verification tooling is important*

Exemplary OS and core/SMT interaction

Application: DayTrade/PDF



- Low # of worker threads: SMT-off out-performs SMT-on due to unbalance thread assignments
- High # of worker threads: SMT-on out-performs SMT-off, as supposed to
- Taskset binding provides predicatable worker to thread assignment

***** core/SMT aware workload management is important & possible***

***** IBM owns hardware architecture and many OS’s for easy cooperation between layers***

Goal

- **Parallelism with hundreds and thousands of threads**
  - Hardware is ready
    - Multi-core processor
    - IBM “POWER7 is designed for multi-socket systems that scale up to 32 sockets, which means that a full 32-socket system of 8-core parts would support 1024 threads.”
  - Software that is able to exploit/adapt to the parallelism provided by the hardware

Challenges of high parallelism

- **Complex commercial workload**
  - Java workload
  - Non-intrusive to existing application
    - No/little modification of application level code, No need for annotation
  - Methodology/tools to identify parallelism of a workload
    - Identify parallelism bottleneck
    - Identify peak parallelism
    - Identify parallelism potential
- **Hybrid execution environment**
  - Loosely-coupled, hybrid execution components (multi-tier)
    - Web Server, Application Server, DB server…,
    - Each tier can be a hybrid
  - Configurable hardware/execution environment
  - Methodology/tools to identify parallelism of an environment
- **Combination/match of commercial workload and execution environment**
  - Identify which workload is best for which environment

backup

Notes

- **(Enterprise) Commercial workload**
  - Java workload
  - Un-intrusive (no modification of application, no specific language)
- **Heterogeneous environment**
  - Identify parallelism of a workload
  - Identify parallelism of an environment
  - Match between workload and environment
- **Possible project**
- **How to avoid the lock delay at first place?**
  - Deterministic lock?
    - Sequential access to the resource without performance dropping
    - Maximum of threads # that it will work
- **Relatively isolate component**

- WAIT report before
- WAIT report after

OCR: Chiron

- **Scope of the research**

  (2) Best practice for software development to exploit hybrid systems:

  - IBM lead : Grace Liu
  - Current experience: locking/data sharing has huge performance impact on hybrid systems / accelerators
  - *New deterministic lock paradigm for parallel/threaded programs*
    - Identify systematic lock usage in middleware and utility software
    - Establish the usage of the deterministic locking mechanisms on hybrid systems
    - Perform performance study with new locking mechanism for selected open source benchmark on hybrid systems
    - Study productivity improvement in debugging and test of the new lock mechanisms
  - Data-sharing
    - Data-sharing in general is protected by locks
    - Data-race-free enabled by deterministic locks

  MIT related project: Kendo

  - Prof. Saman Amarasinghe & student
  - Working framework for deterministic multi-threading on different hardware and Linux that can be used to identify locking problem
  - Strong or weak deterministic interleaving access to shared data
  - Data-race-free program executions

General Practice of Lock

- **Amdahl’s law**
  - The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.
- **Sharing nothing**
  - Identify false sharing
  - Duplicate resource
    - Large on-chip cache to remove bus contention on SMP
- **Differentiate read/write locks**
- **Partial Sharing**
  - Db, table, rows locking
  - Class lock versus object lock in java
- **Minimize synchronized code**
- **Limit # of threads**
  - Too many threads create higher contention and eat up cache and memory space
- **Mutli-thread programming is difficult and error-prone → we are more concerned of performance issue**