Multithreaded Programming

Challenges, current practice, and languages/tools support

Yuan Lin

Systems Group

Sun Microsystems

Multi-core Architectures

- What?
  - A chip contains some number of "cores", each of which can run some number of strands (“hardware threads").
  - Different cores or chips may share caches at different levels. All caches are coherent.
  - All strands have access to the same shared memory.
- Why?
  - More and more difficult to get good ROI by improving single-core performance only.
    - Design complexity, power consumption, heat production, ...
  - Throughput is as important as speed.

Multi-core Architectures

- 2 strand/core, 1 core/chip
  - > Intel Pentium 4 with Hyperthreading
- 1 strand/core, 2 cores/chip
  - > Intel Dual Core Xeon, AMD Dual Core Opteron, Sun UltraSPARC IV
- 2 strand/core, 2 cores/chip
  - > IBM Power 5
- 4 strand/core, 8 cores/chip
  - > Sun UltraSPARC T1

Example: UltraSPARC T1

- Eight cores (individual execution pipelines) per chip.
- Four strands share a pipeline in each core.
- Different strands are scheduled on the pipeline in round-robin order.
- The slot of a stalled strand is given to the next strand.
- Four strands on a core share L1 cache, all strands share L2 cache.
- A single FPU is shared by all cores.
- OS sees the each strand as a processor. OS schedules LWPs on strands. HW schedules strands in the core.

Class of Applications

- Multi-process applications
  - > e.g. Oracle database, SAP, PeopleSoft, ...
- Multi-threaded applications
  - > e.g. Siebel CRM, Sybase engine, ...
- Single-threaded applications
  - > Cannot be directly benefitted from multi-core architectures
  - > Solution: make it multi-threaded or multi-process

Multithreaded Programming

- Are developers ready now?
  - > Language developers
  - > Compiler / tool developers
  - > Library / runtime developers
  - > OS / VM developers
  - > Application developers
- What are the challenges?

Topics

- Multi-core Architectures
- Brief Overview of Pthreads, OpenMP and Java
- The Challenges

Topics

- Multi-core Architectures
- Brief Overview of Pthreads, OpenMP and Java
- The Challenges

Current Popular Parallel Programming Languages and APIs

- Shared Memory
  - > Memory is “shared” unless declared “private”.
  - > Accessing shared memory by direct read or write.
  - > Examples: Pthreads, OpenMP, Java, C#
  - > Closer to multi-core architectures than the rest two.
- Global Address Space
  - > Memory is “private” unless declared “shared”.
  - > Accessing shared memory by direct read or write.
  - > Examples: UPC, Co-array Fortran
- Message Passing
  - > No shared memory. Exchange data via messages.
  - > Example: MPI

Current Popular Parallel Programming Languages and APIs

- Shared Memory
  - > Memory is “shared” unless declared “private”.
  - > Accessing shared memory by direct read or write.
  - > Examples: Pthreads, OpenMP, Java, C#
  - > Closer to multi-core architectures than the rest two.
- Global Address Space
  - > Memory is “private” unless declared “shared”.
  - > Accessing shared memory by direct read or write.
  - > Examples: UPC, Co-array Fortran
- Message Passing
  - > No shared memory. Exchange data via messages.
  - > Example: MPI

POSIX Threads (Pthreads)

- IEEE POSIX 1003.1c-1995 standard:
  A standardized C language thread API
  > thread creation and termination
  > synchronization
  > scheduling
  > data management
  > process interaction
- Now incorporated into Single UNIX Specification, Version 3.
  > barriers, read/write locks, spin locks, etc.

Pthreads – a simple example

```
g = g + compute(1) + compute(2)
```

Pthreads – a simple example

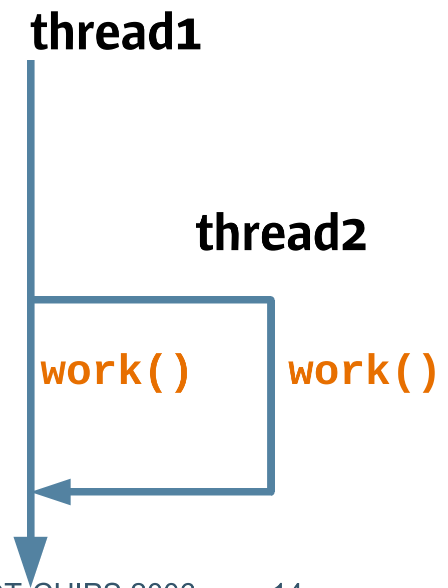
```
g = g + compute(1) + compute(2)
```

```
main()
{
    pthread_t tid;
    void *status;
    int part;

    part = 1;
    pthread_create(&tid, NULL, work, (void *)part);

    part = 2;
    work((void *)part);

    pthread_join(tid, &status);
}
```

Pthreads – a simple example

```
g = g + compute(1) + compute(2)
```

```
main()
{
    pthread_t tid;
    void *status;
    int part;

    part = 1;
    pthread_create(&tid, NULL, work, (void *)part);

    part = 2;
    work((void *)part);

    pthread_join(tid, &status);
}
```



Pthreads – a simple example

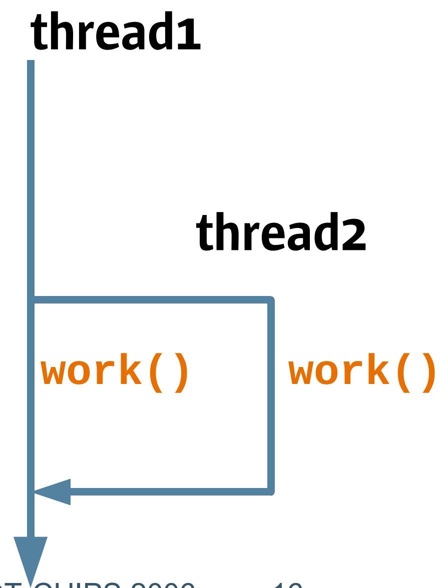
```
g = g + compute(1) + compute(2)
```

```
void *work(void *arg)
{
    int result = compute((int)arg);
    pthread_mutex_lock(&lock);
    g += result;
    pthread_mutex_unlock(&lock);
    return NULL;
}

main()
{
    pthread_t tid;
    void *status;
    int part;

    part = 1;
    pthread_create(&tid, NULL, work, (void *)part);

    part = 2;
    work((void *)part);

    pthread_join(tid, &status);
}
```

Pthreads – a simple example

- Understand Concurrency

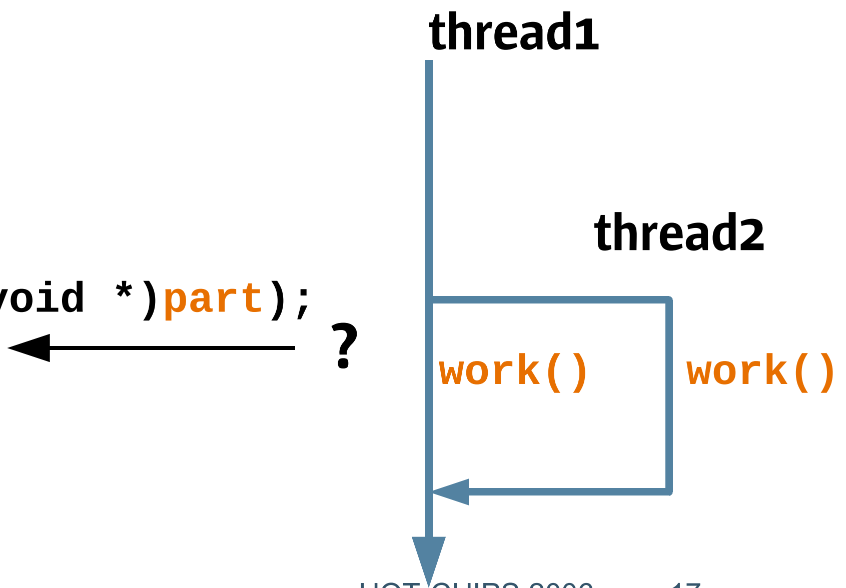
```
main()
{
    pthread_t tid;
    void *status;
    int part;

    part = 1;
    pthread_create(&tid, NULL, work, (void *)part);

    part = 2;
    work((void *)part);

    pthread_join(tid, &status);
}
```



Pthreads – a simple example

- Understand Concurrency: what is the status of thread 2 when pthread_create() returns?
  - > a) thread 2 has not started executing work().
  - > b) thread 2 is executing work().
  - > c) thread 2 has done work() and returned from work().

```
main()
{
    pthread_t tid;
    void *status;
    int part;

    part = 1;
    pthread_create(&tid, NULL, work, (void *)part);

    part = 2;
    work((void *)part);

    pthread_join(tid, &status);
}
```

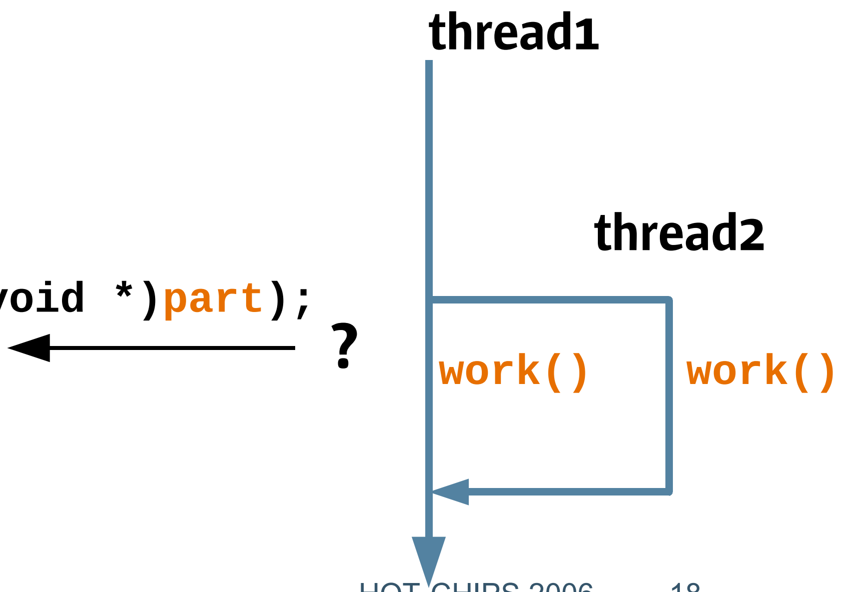


Pthreads – a simple example

- Understand Concurrency: what is the status of thread 2 when pthread_create() returns?
  - a) thread 2 has not started executing work().
  - b) thread 2 is executing work().
  - c) thread 2 has done work() and returned from work().

Could be any of the three!

```
main()
{
    pthread_t tid;
    void *status;
    int part;

    part = 1;
    pthread_create(&tid, NULL, work, (void *)part);

    part = 2;
    work((void *)part);

    pthread_join(tid, &status);
}
```



The Beauty of Pthreads

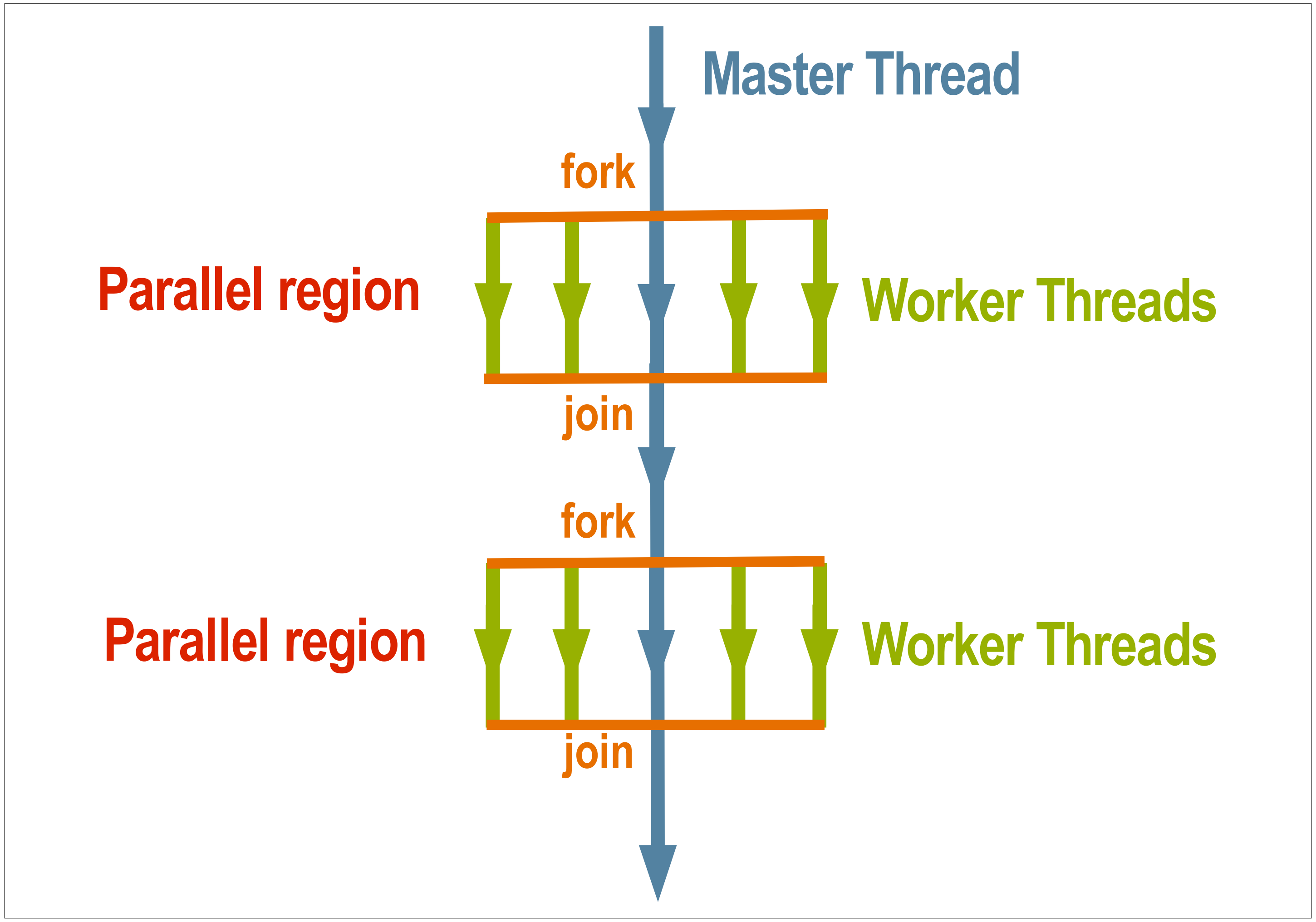
- Comprehensive Set of Functions and Specifications
  - > Thread management
  - > Synchronizations
  - > Scheduling
  - > Data management
  - > Interaction with process, signals, ...

  - ✓ Flexible
  - ✓ Building block of higher level runtime

OpenMP

- An API specification for writing shared memory parallel programs in C, C++ and Fortran
- Consists of:
  - > Compiler directives
  - > Runtime routines
  - > Environment variables
- Specification maintained by the OpenMP Architecture Review Board http://www.openmp.org
- Latest Specification: Version 2.5
- Language committee meetings for V3.0: since 9/2005

Thread Execution Model: fork and join



OpenMP – a simple example

```
g = g + compute(1) + compute(2)
```

```
for (int i=1; i<=2; i++) {
    int result;

    result = compute(i);
    g += result;
}
```

OpenMP – a simple example

```
g = g + compute(1) + compute(2)
```

```
#pragma omp parallel for reduction(+:g)
for (int i=1; i<=2; i++) {
    int result;

    result = compute(i);
    g += result;
}
```

The Beauty of OpenMP

- Possible to write a sequential program and its “functionally-equivalent” parallel version in the same code.

```
#pragma omp parallel for reduction(+:g)
for (int i=1; i<=2; i++) {
    int result;

    result = compute(i);
    g += result;
}
```

- ✔ makes debugging easier
- ✔ allows incremental parallelization

Java (J2SE)

- Basic threading support in the original spec
  - > new Thread(), Object.wait, notify, notifyAll, synchronized
- Threading related updates for J2SE5
  - > JSR-133: java memory model
  - > JSR-166: java.util.concurrent package
    - > Task scheduling framework
    - > Concurrent collections
    - > Atomic variables
    - > Synchronizers (e.g. barriers, semaphores, latches, ...)
    - > Locks
    - > Timing

Java – a simple example

```
g = g + compute(1) + compute(2)
```

```
import java.util.concurrent.atomic.*;
public class A implement Runnable {
 private int g;
 private AtomicInteger id = new AtomicInteger(0);
 ...
 public void run() {
   int myid, myg;
   myid = id.incrementAndGet()
   myg = compute(myid);
   synchronized(this) {
     g += myg;
   }
 }
```

```
  public void go() {
    Thread t[];
    t = new Thread[2];
    id = 1;
    for (i=0; i<2; i++) {
      t[i] = new Thread(this);
      t[i].start();
    }
    for (i=0; i<2; i++) {
      try {
        t[i].join();
      }
      catch(InterruptedException iex){}
    }
  }
}
```

Java – a simple example

```
g = g + compute(1) + compute(2)
```

```
import java.util.concurrent.atomic.*;
public class A implement Runnable {
 private int g;
 private AtomicInteger id = new AtomicInteger(0);
 ...
 public void run() {
   int myid, myg;
   myid = id.incrementAndGet()
   myg = compute(myid);
   synchronized(this) {
     g += myg;
   }
 }
```

thread creation start, and join

```
  public void go() {
    Thread t[];
    t = new Thread[2];
    id = 1;
    for (i=0; i<2; i++) {
      t[i] = new Thread(this);
      t[i].start();
    }
    for (i=0; i<2; i++) {
      try {
        t[i].join();
      }
      catch(InterruptedException iex){}
    }
  }
}
```

Java – a simple example

```
g = g + compute(1) + compute(2)
```

```
import java.util.concurrent.atomic.*;
public class A implement Runnable {
 private int g;
 private AtomicInteger id = new AtomicInteger(0);
 ...
 public void run() {
   int myid, myg;
   myid = id.incrementAndGet()
   myg = compute(myid);
   synchronized(this) {
     g += myg;
   }
 }
```

atomic variables

```
  public void go() {
    Thread t[];
    t = new Thread[2];
    id = 1;
    for (i=0; i<2; i++) {
      t[i] = new Thread(this);
      t[i].start();
    }
    for (i=0; i<2; i++) {
      try {
        t[i].join();
      }
      catch(InterruptedException iex){}
    }
  }
}
```

Java – a simple example

```
g = g + compute(1) + compute(2)
```

```
import java.util.concurrent.atomic.*;
public class A implement Runnable {
 private int g;
 private AtomicInteger id = new AtomicInteger(0);
 ...
 public void run() {
   int myid, myg;
   myid = id.incrementAndGet()
   myg = compute(myid);
   synchronized(this) {
     g += myg;
   }
 }
```

```
  public void go() {
    Thread t[];
    t = new Thread[2];
    id = 1;
    for (i=0; i<2; i++) {
      t[i] = new Thread(this);
      t[i].start();
    }
    for (i=0; i<2; i++) {
      try {
        t[i].join();
      }
      catch(InterruptedException iex){}
    }
  }
}
```

synchronized block

The Beauty of Java

- A well studied and specified memory model.
- A set of easy to use, reliable, and performant concurrency utilities with many commonly used data structures.

Topics

- Multi-core Architectures
- Brief Overview of Pthreads, OpenMP and Java
- The Challenges

Challenges in Multithreaded Programming

1. Finding and creating concurrent tasks
2. Mapping tasks to threads
3. Defining and implementing synchronization protocols
4. Dealing with race conditions
5. Dealing with deadlocks
6. Dealing with memory model
7. Composing parallel tasks
8. Achieving scalability
9. Achieving portable & predictable performance
10. Recovering from errors
11. Dealing with all single thread issues

For Each Challenge

- What is it?
- How do programmers deal with it now?
- Any help from Pthreads, OpenMP or Java? **P/O/J**
- Any tools that can help now?

1

Find and Create Concurrent Tasks

- Concurrency at Data Level
  - e.g. 6 crying babies waiting for diaper-change
- Concurrency at Function Level
  - e.g. changing diaper for a baby, complaining to the spouse, while playing sudoku in mind
- Granularity

Automatic Parallelization

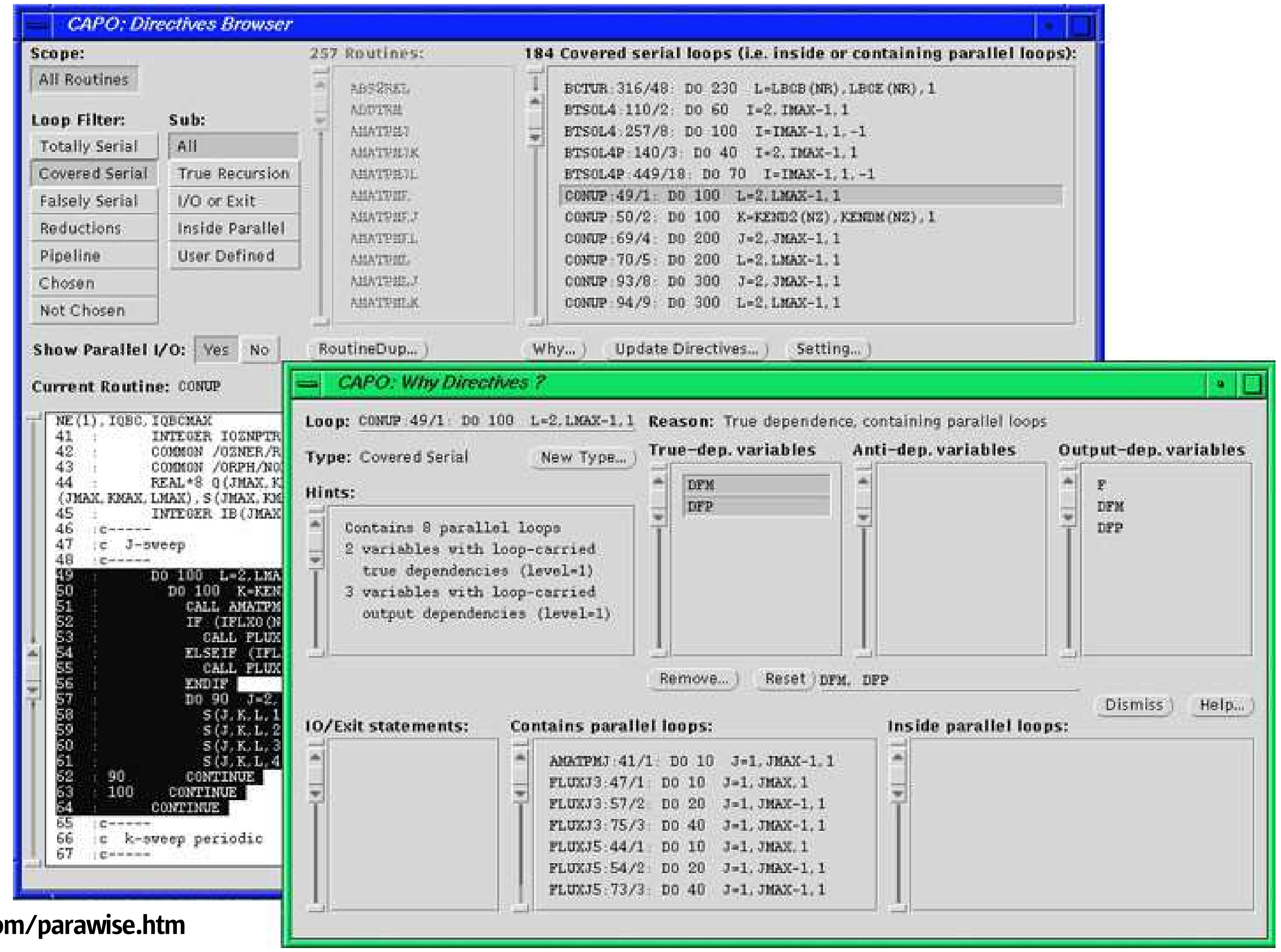
- Available in many compilers
  - SGI, Sun, Intel, IBM, ...

```
> f90 -xautopar old-serial.f
```

- Long history
- Mostly loop based parallelization
- Limited adoption

Parallelization Assistance Tool

- ParaWise from Parallel Software Products, Inc.
  - a semi-automatic parallelization tool for Fortran programs
  - Long history
  - Limited adoption



184 Covered serial loops (i.e. inside or containing parallel loops):

```
BCTUR:316/48:  DO 230   L=LBCB(NR),LBCE(NR),1
BTSOL4:110/2:  DO 60   I=2,IMAX-1,1
BTSOL4:257/8:  DO 100   I=IMAX-1,1,-1
BTSOL4P:140/3:  DO 40   I=2,IMAX-1,1
BTSOL4P:449/18:  DO 70   I=IMAX-1,1,-1
CONUP:49/1:  DO 100   L=2,LMAX-1,1
CONUP:50/2:  DO 100   K=KEND2(NZ),KENDM(NZ),1
CONUP:69/4:  DO 200   J=2,JMAX-1,1
CONUP:70/5:  DO 200   L=2,LMAX-1,1
CONUP:93/8:  DO 300   J=2,JMAX-1,1
CONUP:94/9:  DO 300   L=2,LMAX-1,1
```

Current Routine: CONUP

```
NE(1),IQBC,IQBCMAX
41  :         INTEGER IOZNPTR
42  :         COMMON /OZNER/R
43  :         COMMON /ORPH/NO
44  :         REAL*8 Q(JMAX,K
(JMAX,KMAX,LMAX),S(JMAX,KM
45  :         INTEGER IB(JMAX
46  :c-----
47  :c  J-sweep
48  :c-----
49  :         DO 100  L=2,LMA
50  :           DO 100  K=KEN
51  :             CALL AMATPM
52  :             IF (IFLXO(N
53  :               CALL FLUX
54  :             ELSEIF (IFL
55  :               CALL FLUX
56  :             ENDIF
57  :             DO 90  J=2,
58  :               S(J,K,L,1
59  :               S(J,K,L,2
60  :               S(J,K,L,3
61  :               S(J,K,L,4
62  : 90          CONTINUE
63  : 100       CONTINUE
64  :         CONTINUE
65  :c-----
66  :c  k-sweep periodic
67  :c-----
```

Loop: CONUP:49/1: DO 100 L=2,LMAX-1,1

Reason: True dependence, containing parallel loops

Type: Covered Serial

Hints:

```
Contains 8 parallel loops
2 variables with loop-carried
  true dependencies (level=1)
3 variables with loop-carried
  output dependencies (level=1)
```

True-dep. variables	Anti-dep. variables	Output-dep. variables
DFM		F
DFP		DFM
		DFP

DFM, DFP

IO/Exit statements:

Contains parallel loops:

```
AMATPMJ:41/1:  DO 10   J=1,JMAX-1,1
FLUXJ3:47/1:  DO 10   J=1,JMAX,1
FLUXJ3:57/2:  DO 20   J=1,JMAX-1,1
FLUXJ3:75/3:  DO 40   J=1,JMAX-1,1
FLUXJ5:44/1:  DO 10   J=1,JMAX,1
FLUXJ5:54/2:  DO 20   J=1,JMAX-1,1
FLUXJ5:73/3:  DO 40   J=1,JMAX-1,1
```

Inside parallel loops:

Screenshots from http://www.parallelsp.com/parawise.htm

Parallelization Assistance Tool

- A valuable tool may not need to be fancy.
- Dieter an Mey (RWTH, Aachen University) :
  - “I consider the following a very valuable assistance tool: a tool that can find all the places in the source code where global variables are declared and all the places where the global variables are referenced, and show these places with a different color.”
  - “I got the functionality from the Foresys tool from Simulog for Fortran and I found it quite useful. It would be nice to have such features in an IDE. ”

Find and Create Concurrent Tasks P/O/J

- Pthread
  - > No direct support.
- Java
  - > No direct support.
  - > The use of Executor/Thread Pools makes the concept of task explicit. (See details later.)

Find and Create Concurrent Tasks P/O/J

- OpenMP
  - Use *worksharing* loops for basic data level parallelism

```
#pragma omp for
for (i=1; i<n; i++)
   work(data[i]);
```

  - Use *worksharing* sections for basic function level parallelism

```
#pragma omp sections
{
  #pragma omp section
  func1();
  #pragma omp section
  func2();
}
```

2

Tasks and Threads Mapping

- Threads vs Tasks
  - Thread: parallel execution engines provided by the OS
  - Tasks: concurrency unit in program logic
- A program can have tens of thousands of concurrent tasks.
  - e.g. players in a game server, components in a simulation system, requests in a web server
- The system can only supply hundreds of threads.
  - e.g. because of resource limitation
- There is overhead associated with creating and destroying threads.

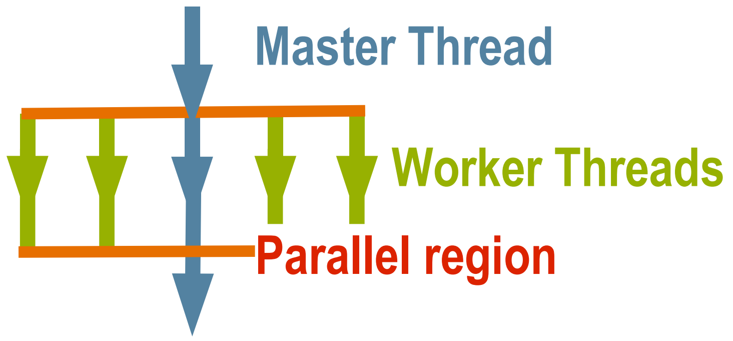
Tasks and Threads Mapping

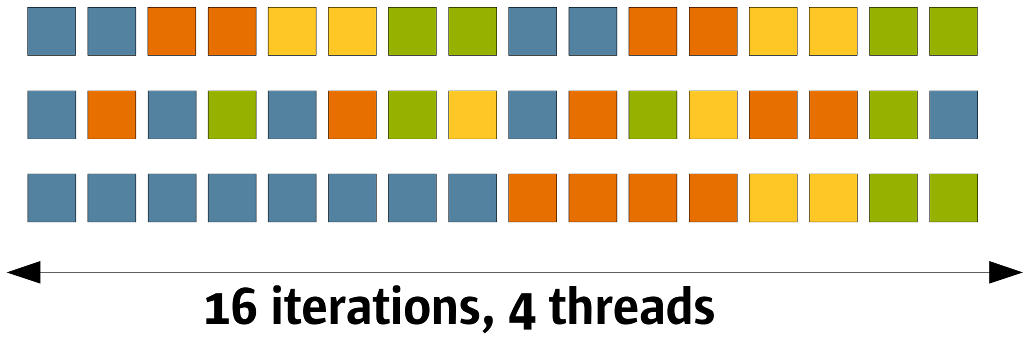
- Use well-designed data structures to hold task state.
- Deploy a task-to-thread model to deliver scalable and predictable performance.
  - master/slave
    - Master thread dispatches slave threads from a thread-pool to work on tasks. The threads in the pool are pre-allocated.
  - pipeline
    - Each thread performs a specific operation on a data item, and passes the data to the next thread in the pipeline.
  - task pool
    - Each thread execute a task from a pool and put new task generated to the pool.
  - ...

Tasks and Threads Mapping

P/O/J

- Pthreads : No direct support
- OpenMP :
  - > Basic master/slave model
  - > Three loop scheduling methods
    - > Static
    - > Dynamic
    - > Guided





Tasks and Threads Mapping

P/O/J

- Java
  - Thread pool utilities in J2SE5 provide pre-built thread mechanism to run tasks to a bounded number of threads.
  - To use thread pool,
    - instantiate an implementation of the ExecutorService interface and hand it a set of tasks, or
    - use configurable implementations: ThreadPoolExecutor and ScheduledThreadPoolExecutor
  - Use Callable and Future interfaces to get result from tasks.

```
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize) throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }
   public void serve() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) {
       pool.shutdown();
     }
   }
}
class Handler implements Runnable {
   private final Socket socket;
   Handler(Socket socket) { this.socket = socket; }
   public void run() {
     // read and service request
   }
}
```

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ExecutorService.html

Thread pool

```
class NetworkService {
    private final ServerSocket serverSocket;
    private final ExecutorService pool;

    public NetworkService(int port, int poolSize) throws IOException {
      serverSocket = new ServerSocket(port);
      pool = Executors.newFixedThreadPool(poolSize);
    }
    public void serve() {
      try {
        for (;;) {
          pool.execute(new Handler(serverSocket.accept()));
        }
      } catch (IOException ex) {
        pool.shutdown();
      }
    }
 }
 class Handler implements Runnable {
    private final Socket socket;
    Handler(Socket socket) { this.socket = socket; }
    public void run() {
      // read and service request
    }
 }
```

Task class

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ExecutorService.html

```
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize) throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }
   public void serve() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) {
       pool.shutdown();
     }
   }
}
class Handler implements Runnable {
   private final Socket socket;
   Handler(Socket socket) { this.socket = socket; }
   public void run() {
     // read and service request
   }
}
```

Create thread pool

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ExecutorService.html

```
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize) throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }
   public void serve() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) {
       pool.shutdown();
     }
   }
}
class Handler implements Runnable {
   private final Socket socket;
   Handler(Socket socket) { this.socket = socket; }
   public void run() {
     // read and service request
   }
}
```

Generate new tasks

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ExecutorService.html

```
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize) throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }
   public void serve() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) {
       pool.shutdown();
     }
   }
}
class Handler implements Runnable {
   private final Socket socket;
   Handler(Socket socket) { this.socket = socket; }
   public void run() {
     // read and service request
   }
}
```

Assign task to thread pool

```
class NetworkService {
   private final ServerSocket serverSocket;
   private final ExecutorService pool;

   public NetworkService(int port, int poolSize) throws IOException {
     serverSocket = new ServerSocket(port);
     pool = Executors.newFixedThreadPool(poolSize);
   }
   public void serve() {
     try {
       for (;;) {
         pool.execute(new Handler(serverSocket.accept()));
       }
     } catch (IOException ex) {
       pool.shutdown();
     }
   }
}
class Handler implements Runnable {
   private final Socket socket;
   Handler(Socket socket) { this.socket = socket; }
   public void run() {
     // read and service request
   }
}
```

Shutdown thread pool

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/ExecutorService.html

3

Defining and Implementing Synchronizations

- Understand the application logic and design the high-level synchronization scheme
  - e.g. traffic light
- Use the right synchronization primitives at the right place.
  - e.g. need to coordinate threads so they are executed in phases
    - use condition variables?
    - use barriers?
    - join threads and create new threads?

Defining and Implementing Synchronizations

P/O/J

- Pthreads:
  - > Mutex locks, condition variables, semaphores, barriers, reader-writer locks
- OpenMP:
  - > Locks, barriers, critical sections, ordered regions
- Java
  - > Synchronized method/block, wait/notify
  - > Condition variables, semaphores, barriers, countdown latch, exchanger, reader-writer locks

4

Race Conditions

- Race conditions occur when different threads access shared data without explicit synchronization.
- Race conditions can cause programs to behave in ways unexpected by the programmer.
- Bugs caused by race conditions are notoriously difficult to debug.

Race Conditions – An Example

Assume a = b = 0;

Thread 1:

```
a = a + 5;

lock (a_lock);
   b = b + 10;
unlock (a_lock);

barrier();
```

Thread 2:

```
lock (a_lock);
   b = b - 10;
unlock (a_lock);

a = a – 5;

barrier();
```

What's the value of 'a' and 'b' after the barrier?

Race Conditions – An Example

Thread 1:

```
a = a + 5;

lock (a_lock);
   b = b + 10;
unlock (a_lock);

barrier();
```

Assume a = b = 0;

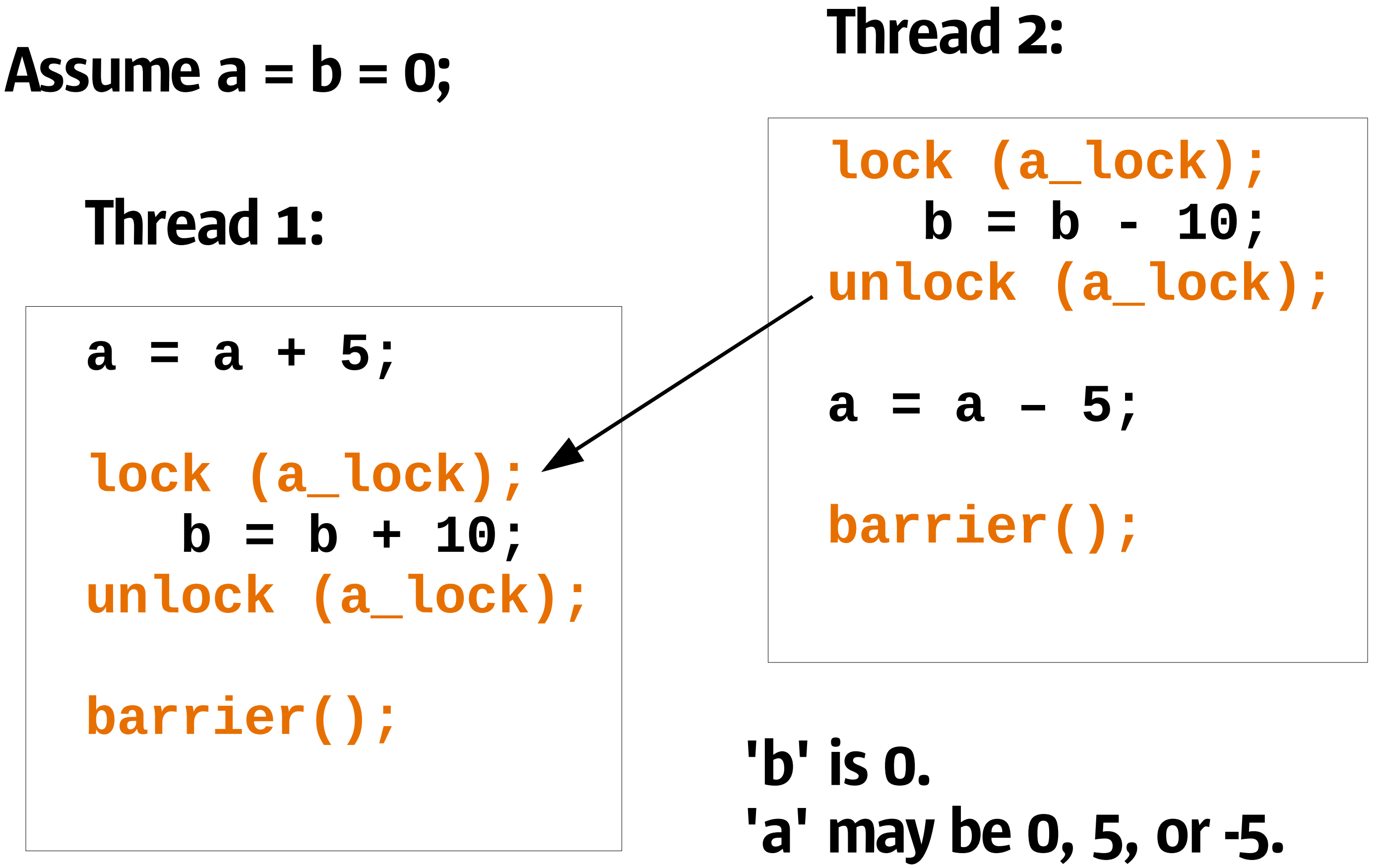
Thread 2:

```
lock (a_lock);
   b = b - 10;
unlock (a_lock);

a = a – 5;

barrier();
```

We have a = 0 and b = 0.

Race Conditions – An Example

Assume a = b = 0;



Thread 1:

```
a = a + 5;

lock (a_lock);
   b = b + 10;
unlock (a_lock);

barrier();
```

Thread 2:

```
lock (a_lock);
   b = b - 10;
unlock (a_lock);

a = a – 5;

barrier();
```

'b' is 0.
'a' may be 0, 5, or -5.

Two Kinds of Race Conditions (1/2)

- Data Race
  - Concurrent accesses (at least one is a write) to shared memory are not protected by critical sections. Atomicity of the accesses is lost.

Thread 1

```
lock();
acnt1 = acnt1 + delta_x;
acnt2 = acnt2 - delta_x;
unlock();
```

Thread 2

```
lock();
acnt1 = acnt1 + delta_y;
acnt2 = acnt2 - delta_y;
unlock();
```

Two Kinds of Race Conditions (2/2)

- General Race
  - > The order of accesses to shared memory is not enforced by synchronization.

Thread 1

```
put_job_in_queue(job)
post();
```

Thread 2

```
wait();
job = get_job_from_queue();
```

Data Race vs General Race

- A data race is also a general race.
- A data race can be fixed by using critical sections or adding synchronizations.
- A general race can be fixed by adding synchronizations to enforce ordering.
- Watch out for general races when the shared-memory accesses should occur in specific order.
- Watch out for data races in asynchronous programs.
- Tools are available to detect data races.

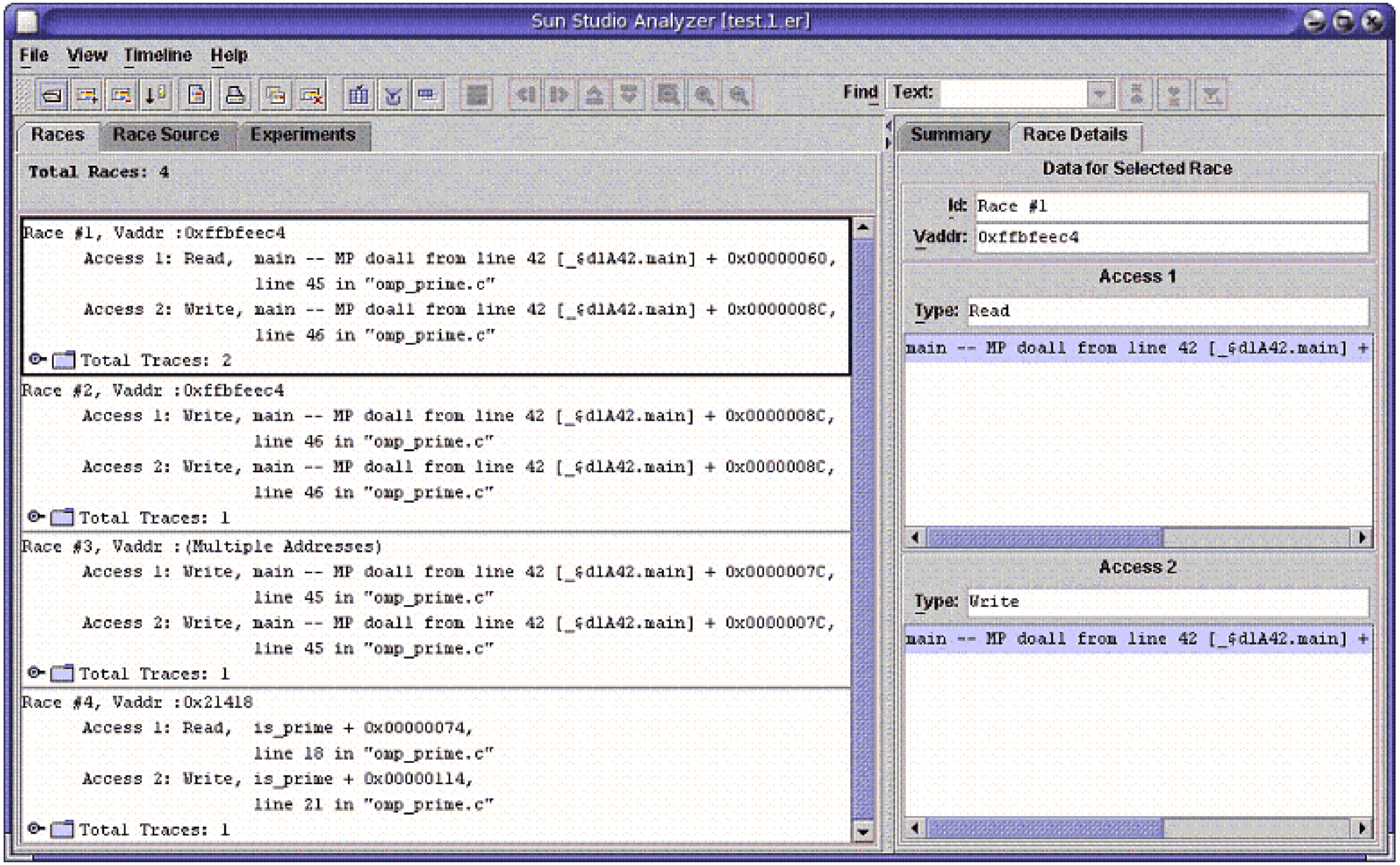
Data Race Detection Tools

- Static Detection
  - > + Can be fast and consume little memory.
  - > + Does not affect the behavior of program.
  - > + Is not affected by input data and scheduling
  - > + Can be used to check OS kernels and device drivers
  - > - False positives
- Example: LockLint from Sun Studio
  - > Reports data races and deadlocks due to inconsistent use of locking techniques.
  - > Originates from *WARLOCK*, which was designed to detect such errors in Solaris kernels and device drivers.

Data Race Detection Tools

- Runtime Detection
- Examples:
  - Helgrind from Valgrind
  - HP Visual Threads
  - Intel Thread Checker
  - Sun Data Race Detection Tool

Data Race Detection Tools


Sun Studio Analyzer [test.1.er]
File View Timeline Help
Find Text:
Races Race Source Experiments
Summary Race Details

```
Total Races: 4

Race #1, Vaddr :0xffbfeec4
      Access 1: Read,  main -- MP doall from line 42 [_$d1A42.main] + 0x00000060,
                       line 45 in "omp_prime.c"
      Access 2: Write, main -- MP doall from line 42 [_$d1A42.main] + 0x0000008C,
                       line 46 in "omp_prime.c"
   Total Traces: 2
Race #2, Vaddr :0xffbfeec4
      Access 1: Write, main -- MP doall from line 42 [_$d1A42.main] + 0x0000008C,
                       line 46 in "omp_prime.c"
      Access 2: Write, main -- MP doall from line 42 [_$d1A42.main] + 0x0000008C,
                       line 46 in "omp_prime.c"
   Total Traces: 1
Race #3, Vaddr :(Multiple Addresses)
      Access 1: Write, main -- MP doall from line 42 [_$d1A42.main] + 0x0000007C,
                       line 45 in "omp_prime.c"
      Access 2: Write, main -- MP doall from line 42 [_$d1A42.main] + 0x0000007C,
                       line 45 in "omp_prime.c"
   Total Traces: 1
Race #4, Vaddr :0x21418
      Access 1: Read,  is_prime + 0x00000074,
                       line 18 in "omp_prime.c"
      Access 2: Write, is_prime + 0x00000114,
                       line 21 in "omp_prime.c"
   Total Traces: 1
```

Data for Selected Race

Id: Race #1
Vaddr: 0xffbfeec4

Access 1

Type: Read

main -- MP doall from line 42 [_$d1A42.main] +

Access 2

Type: Write

main -- MP doall from line 42 [_$d1A42.main] +

What to Do After a Data-race is Found?

- 1) Check whether it is a false positive.
  - A false positive is a reported data race that actually does not exist in the program.
  - No tool is perfect.
  - For example, the tool may not understand the synchronizations in your program.

What to Do After a Data-race is Found?

- 1) Check whether it is a false positive.
  - > A false positive is a reported data race that actually does not exist in the program.
  - > No tool is perfect.
  - > For example, the tool may not understand the synchronizations in your program.
- 2) Check whether it is a benign race.
  - > The programmer may put data race there in order to get better performance.
    - > Yes? Are you sure? Check again!
  - > Programs with intentional data races are very difficult to get right.
    - > Are you relying on sequential consistency?

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - A data race could be a bug or caused by a bug.

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void*),
                   void *arg);
```

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void*),
                   void *arg);
```

```
void *work(void *arg)
{
     int myid = *(int *)arg;
     data[myid] = update(data[myid]);
     return NULL;
}

for (i=0; i<THREADS; i++)
     pthread_create(tids[i], NULL, work, &i);
```

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - A data race could be a bug or caused by a bug.

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void*),
                   void *arg);
```

```
void *work(void *arg)
{
     int myid = *(int *)arg;
     data[myid] = update(data[myid]);
     return NULL;
}

for (i=0; i<THREADS; i++)
     pthread_create(tids[i], NULL, work, &i);
```

Data race reported

Bug!

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - A data race could be a bug or caused by a bug.
  - Most tools detect *certain* kind of data races only!

thread 1

```
acnt1 += x;

acnt2 -= x;
```

thread 2

```
acnt1 += y;

acnt2 -= y;
```

thread 3

```
print acnt1, acnt2
```

Data Race! Accounts are not balanced!
“account1 + account2” is not constant.

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - > A data race could be a bug or caused by a bug.
  - > Most tools detect *certain* kind of data races only!

thread 1

```
lock();
acnt1 += x;
unlock();
lock();
acnt2 -= x;
unlock();
```

thread 2

```
lock();
acnt1 += y;
unlock();
lock();
acnt2 -= y;
unlock();
```

thread 3

```
lock();
print acnt1, acnt2
unlock();
```

A wrong fix. No data race is reported.
But accounts are not balanced at certain stages!
"account1 + account2" is not constant.

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - A data race could be a bug or caused by a bug.
  - Most tools detect *certain* kind of data races only!

thread 1

```
lock();
acnt1 += x;


acnt2 -= x;
unlock();
```

thread 2

```
lock();
acnt1 += y;


acnt2 -= y;
unlock();
```

thread 3

```
lock();
print acnt1, acnt2
unlock();
```

A correct fix.

What to Do After a Data-race is Found?

- 3) Fix the bug, not the data race!
  - > A data race could be a bug or caused by a bug.
  - > Most tools detect *certain* kind of data races only!
- 4) Run the detection tool again after code change!
  - > A tool may not be able to detect all data races by design.
  - > The code change may introduce or reveal another data race.

Making Global Variables Private

- Pthreads: thread specific data
  - > `pthread_key_create()`, `pthread_key_delete()`
  - > `pthread_setspecific()`, `pthread_getspecific()`
- OpenMP: “threadprivate” directive

```
int total_for_this_thread;
#pragma omp threadprivate (total_for_this_thread)
```

- Java: java.lang.ThreadLocal class
  - > private static ThreadLocal<int> totoal_for_this_thread;
- Thread Local Storage
  - > Provided by compiler and runtime linker

```
__thread int total_for_this_thread;
```

Use MT-Safe Routines

- Four levels of MT safe attributes for library interfaces.
- 1) Unsafe
  - > Contains global and static data that are not protected. User should make sure only one thread at time to execute the call.

Unsafe Function	**Reentrant counterpart**
`ctime`	`ctime_r`
`localtime`	`localtime_r`
`asctime`	`asctime_r`
`gmtime`	`gmtime_r`
`ctermid`	`ctermid_r`
`getlogin`	`getlogin_r`
`rand`	`rand_r`
`readdir`	`readdir_r`
`strtok`	`strtok_r`
`tmpnam`	`tmpnam_r`

Use MT-Safe Routines

- 2) Safe
  - Global and static data are protected. Might not provide any concurrency between calls made by different threads.
  - Example: malloc in libc(3c)
- 3) MT-Safe
  - Safe and can provide a reasonable amount of concurrency.
  - Example: malloc in libmtmalloc(3LIB).

Use MT-Safe Routines

- 4) Async-signal-safe
  - Can be safely called from a signal handler.
  - Example:
    - Not async-signal-safe: malloc(), pthread_getspecific()
    - Async-signal-safe: open(), read()

5

Deadlocks

- Two or more competing actions are waiting for the other to finish. None can finish.
- A common misunderstanding: a deadlock must involve mutex locks.
  - > Deadlocks may be caused by mis-use of synchronizations or communications, such as barriers.
- Livelock
  - > Two or more competing actions continually change their state in response to changes in the other actions. None will finish.

Necessary Conditions for a Deadlock

1. Mutual exclusion: a resource is exclusively owned by the process to which it is assigned.
2. Hold and wait: processes request new resources while holding some resources.
3. No preemption: only the process holding a resource can release it.
4. Circular wait: there is a circular chain of processes in which each process waits for a resource that the next process in the chain holds.

Deadlock: Prevention and Avoidance

- Prevention
  - > Use protocols to ensure at least one of the four conditions does not hold.
  - > Example: enforce lock hierarchy to ensure circular-wait never happens.
- Avoidance
  - > Use global resource information to decide whether the current request can be satisfied or delayed.
  - > Example: Banker's algorithm

Deadlock: Prevention and Avoidance

- It may be easy to apply the text book rules to deadlocks that involve obvious resources.
- In many applications, the four conditions are subtle, especially when the deadlock is caused by communication.
- Always examine the program logic!

Deadlocks: Example 1

```
thread 1:
        Lock(a);
        Lock(b);
thread 2:
        Lock(b);
        Lock(c);
thread 3:
        Lock(c);
        Lock(a);
```

The program may or may not deadlock in a particular run.

Deadlocks: Example 2

```
thread tid1:
        pthread_join(tid2, NULL);

thread tid2:
        pthread_join(tid1, NULL);
```

Deadlocks: Example 3

```
#pragma omp parallel num_threads(4)
{
    if (omp_thread_num()<3) {
        #pragma omp barrier
    }
    else {
        #pragma omp barrier
    }
}
```

Illegal OpenMP program!

Deadlocks: Example 4

```
          int flag = 0;

thread t1: while (!flag);
           pthread_exit(0);

thread t2: flag = 1;
           pthread_join(t1, NULL);
```

May loop forever.

Deadlocks: Example 4

May loop forever.

```
thread t1: while (!flag);
           pthread_exit(0);
```

Compiler optimization

```
  reg = flag;
  if (!reg) {
        while (1);
  }
  pthread_exit(0);
```

Deadlocks

P/O/J

- Pthreads
  - Some error checking on locks, but not deadlocks.
    - If the mutex type is PTHREAD_MUTEX_ERRORCHECK and a thread attempts to relock a mutex that it has already locked, an error will be returned.
    - If the mutex type is PTHREAD_MUTEX_ERRORCHECK and a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, an error will be returned.
- OpenMP
  - Makes one kind of deadlock illegal.
    - Barriers in a parallel region must be encountered by all threads in the team in the same order.

Deadlocks

P/O/J

- Java
  - > Structured synchronized method/block makes the critical regions clearer than unstructured locks.
  - > Locks associated with *synchronized* keyword are always released when the thread leaves (due to exception or not) the synchronized scope.
    - > This automatic-releasing could be a bad thing for error recovery since it may leave the data in an inconsistent state that is not protected by locks.

Deadlock Detection Tools

- Static Checking
  - > LockLint from Sun
- Runtime Checking
  - > Thread Checker from Intel
  - > OpenMP Runtime Error Detection from Sun

```
> setenv SUNW_MP_WARN TRUE
> a.out
WARNING (libmtsk): Threads at barrier from different directives.
   Thread at barrier from init.c:12.
   Thread at barrier from forbidden.c:17.
   Possible Reasons:
   Worksharing constructs not encountered by all threads in the
   team in the same order.
   Incorrect placement of barrier directives.
```

6

Memory Consistency Model

- The order in which memory operations will appear to be executed to a programmer.
- What affects the memory consistency?
  - > Language
  - > Compiler
  - > Hardware

Example: Lazy Initialization

- The Sequential Version

```
typedef struct
{
   int data1;
   int data2;
   ...
} A;
```

```
A *init_single_A()
{
  static A *single_A;
  if (single_A == NULL) {
     single_A = malloc(sizeof(A));
     single_A->data1 = ...;
     ...
  }
  return single_A;
}
```

It does not work in mt applications.

Example: Lazy Initialization

- Multi-threaded Versions

```
A *init_single_A()
{
  static A *single_A;
  lock();
  if (single_A == NULL) {
     single_A = malloc(sizeof(A));
     single_A->data1 = ...;
     ...
  }
  unlock();
  return single_A;
}
```

Example: Lazy Initialization

- Multi-threaded Versions

```
A *init_single_A()
{
  static A *single_A;
  lock();
  if (single_A == NULL) {
     single_A = malloc(sizeof(A));
     single_A->data1 = ...;
     ...
  }
  unlock();
  return single_A;
}
```

Not efficient

```
A *init_single_A()
{
  static A *single_A;
  A *temp = single_A;
  if (temp == NULL) {
     lock();
     if (single_A == NULL) {
          temp = malloc(sizeof(A));
          temp->data1 = ...;
          ...
          single_A = temp;
     }
     unlock();
  }
  return temp;
}
```

Example: Lazy Initialization

- Multi-threaded Versions

```
A *init_single_A()
{
  static A *single_A;
  lock();
  if (single_A == NULL) {
     single_A = malloc(sizeof(A));
     single_A->data1 = ...;
     ...
  }
  unlock();
  return single_A;
}
```

Not efficient

"Double-checked Locking"

```
A *init_single_A()
{
  static A *single_A;
  A *temp = single_A;
  if (temp == NULL) {
     lock();
     if (single_A == NULL) {
         temp = malloc(sizeof(A));
         temp->data1 = ...;
         ...
         single_A = temp;
     }
     unlock();
  }
  return temp;
}
```

May be broken

Double-checked Locking

```
A *init_single_A()
{
  static A *single_A;
  A *temp = single_A;
  if (temp == NULL) {
    lock();
    if (single_A == NULL) {
        temp = malloc(sizeof(A));
        temp->data1 = ...;
        ...

        single_A = temp;
    }
   unlock();
  }
  return temp;
}
```

> The compiler may reorder these two writes.

> Even if the compiler does not reorder them, a thread on another processor may perceive the two writes in a different order.

> Therefore, a thread on another processor may read wrong value of single_A->data1.

```
A *p = init_single_A();
... = p->data1;
```

Double-checked Locking

```
A *init_single_A()
{
  static A *single_A;
  A *temp = single_A;
  if (temp == NULL) {
     lock();
     if (single_A == NULL) {
         temp = malloc(sizeof(A));
         temp->data1 = ...;
         ...
         memory_barrier();
         single_A = temp;
     }
    unlock();
  }
  return temp;
}
```

- A possible fix.
- Still broken on some architectures, e.g. Alpha.

Memory Consistency Model

P/O/J

- Pthreads
  - > No formal specification
  - > Shared accesses must be synchronized by calling pthread synchronization functions.
- C++/C
  - > Assumes single thread program execution.
  - > “volatile” restricts compiler optimization, but it does not address the memory consistency issue.
  - > Memory model for multithreaded C++ is being worked on.

Memory Consistency Model

P/O/J

- OpenMP
  - > Detailed clarification. No formal specification.
  - > Each thread has a temporary view of shared memory.
  - > A flush operation restricts the ordering of memory operations and synchronizes a thread's temporary view with shared memory.
  - > All threads must observe any two flush operations with overlapping variable lists in sequential order.

Memory Consistency Model

P/O/J

- Java: revised and clarified by JSR-133
  - > Volatile variables
  - > Final variables
  - > Immutable objects (objects whose fields are only set in their constructor)
  - > Thread- and memory-related JVM functionality and APIs such as class initialization, asynchronous exceptions, finalizers, thread interrupts, and the sleep, wait, and join methods of class Thread

Memory Consistency Model

- Avoid writing codes that have deliberate data races. It is tricky and difficult to understand and debug.

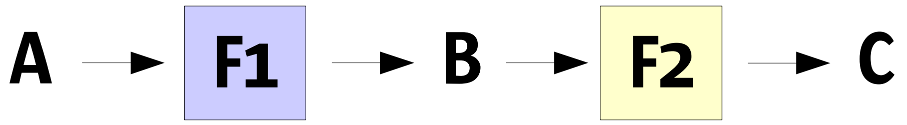
7

Composing Parallel Tasks

- Modular design is common in software development.
- How to compose modules that are multi-threaded?
- Three models of composition
  - Sequential composition
  - Parallel composition
  - Nested composition

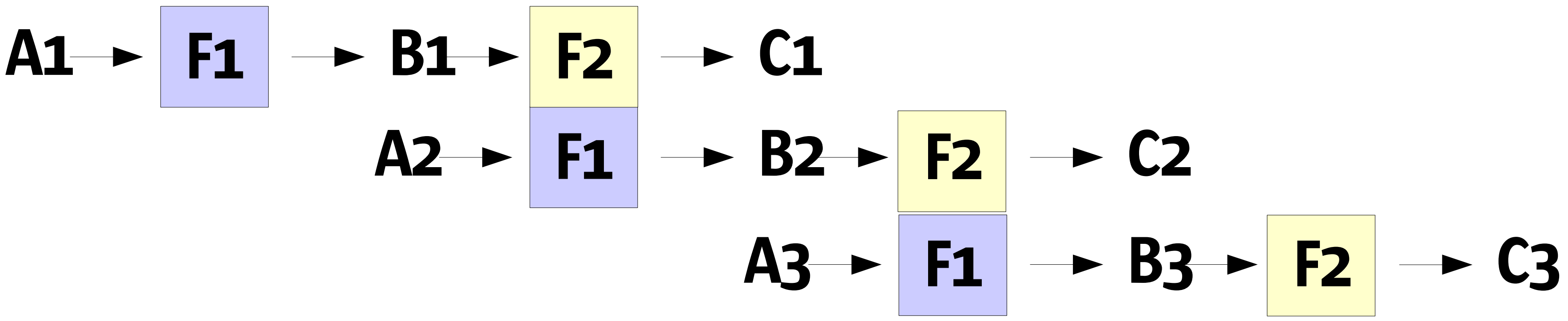
Composition: sequential vs parallel

- Sequential composition:



  > n threads for F1 and n threads for F2

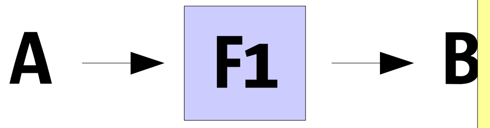
- Parallel composition:



  > n1 threads for F1 and n2 threads for F2, n1+n2=n

Composition: sequential vs parallel

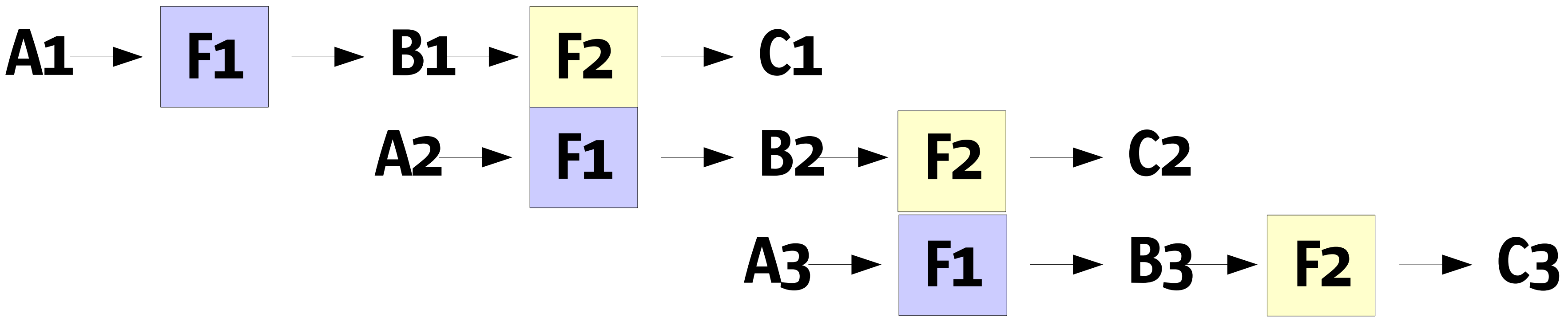
- Sequential compositi



  > n threads for F1 and

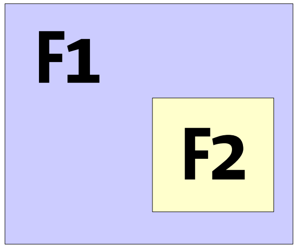
+ may improve resource utilization by overlapping F1 and F2
- may introduce extra communication, data migration, or thread migration

- Parallel composition:



  > n1 threads for F1 and n2 threads for F2, n1+n2=n

Nested Composition



n1 thread for F1 and n2 threads for F2, n1*n2 = n

- Straight-forward approach: outer-module and inner module create their own threads.
  - > Threads explosion.
- More complicated approach: outer-module and inner module get threads from the same thread pool.
  - > Need a common threading frame work.
  - > Implies API interface change or external global state
- Locks are not composable in general!
  - > Dead locks

Composing Parallel Tasks

P/O/J

- Pthreads and Java: No direct support
- OpenMP:
  - > Support nested composition of parallel tasks through nested parallel regions.

```
#omp parallel for
for (i=0; i<n; i++) {
     #omp parallel for
     for (j=0; j<m; j++) {
          ...
     }
}
```

  - > Inflexible and possible low utilization of threads

8

Scalability

- Performance does not scale with the number of threads used.
- Common reason 1: system oversubscribed
  - > e.g. 16 hardware threads, 32 software threads
- Common reason 2: not enough concurrency
  - > Amdahl's law
- Common reason 3: lock contention
  - > Frequent locking, unlocking activities
  - > Long lock holding time

plockstat(1M)

- A utility in Solaris 10 that gathers and displays user-level locking statistics.
- Uses plockstat DTrace provider.
- Three types of lock events can be traced.
  - > Contention events - probes for user level lock contention
  - > Hold events - probes for lock acquiring, releasing etc.
  - > Error events - error conditions.

plockstat(1M) - example

```
>plockstat ./a.out

Mutex block

Count     nsec Lock                         Caller
-------------------------------------------------------------------------------
  863  3822626 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x5c
  900  3423154 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x20
  340  2969890 a.out`mutex                  a.out`_$p1A12.main+0x30
15835  2894962 a.out`mutex                  a.out`_$p1A12.main+0x30
  415  2456968 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x20
  296  2208085 a.out`mutex                  a.out`_$p1A12.main+0x30
  441  2129956 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x5c
14032  2054952 a.out`mutex                  a.out`_$p1A12.main+0x30
    4    42900 libc.so.1`_uberdata          libc.so.1`_thr_exit_common+0xbc
    1    14800 libc.so.1`_uberdata          libmtsk.so.1`threads_fini+0x174
```

plockstat(1M) - example

```
>plockstat ./a.out

Mutex block

Count     nsec Lock                          Caller
-------------------------------------------------------------------------------
  863  3822626 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x5c
  900  3423154 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x20
  340  2969890 a.out`mutex                   a.out`_$p1A12.main+0x30
15835  2894962 a.out`mutex                   a.out`_$p1A12.main+0x30
  415  2456968 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x20
  296  2208085 a.out`mutex                   a.out`_$p1A12.main+0x30
  441  2129956 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x5c
14032  2054952 a.out`mutex                   a.out`_$p1A12.main+0x30
    4    42900 libc.so.1`_uberdata           libc.so.1`_thr_exit_common+0xbc
    1    14800 libc.so.1`_uberdata           libmtsk.so.1`threads_fini+0x174
```

libc_malloc_lock   a.out`mutex

plockstat(1M) - example

(plockstat output cont.)

```
Mutex spin

Count     nsec Lock                       Caller
-------------------------------------------------------------------------------
 4225    13553 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x20
72525    13518 a.out`mutex                  a.out`_$p1A12.main+0x30
 1711    13399 a.out`mutex                  a.out`_$p1A12.main+0x30
    7    11600 libc.so.1`_uberdata          libc.so.1`_thr_exit_common+0xbc
 8111    10778 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x5c

Mutex unsuccessful spin

Count     nsec Lock                       Caller
-------------------------------------------------------------------------------
 1315    31766 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x20
  636    31008 a.out`mutex                  a.out`_$p1A12.main+0x30
29867    30875 a.out`mutex                  a.out`_$p1A12.main+0x30
   31    30590 libc.so.1`_uberdata          libc.so.1`_thr_exit_common+0xbc
 1304    29273 libc.so.1`libc_malloc_lock   a.out`_$p1A12.main+0x5c
    7    26320 libc.so.1`_uberdata          libmtsk.so.1`threads_fini+0x174
   19    24328 libc.so.1`_uberdata          libc.so.1`_thr_exit_common+0xbc
```

plockstat(1M) - example

(plockstat output cont.)

```
Mutex spin

Count     nsec Lock                          Caller
-------------------------------------------------------------------------------
 4225    13553 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x20
72525    13518 a.out`mutex                   a.out`_$p1A12.main+0x30
 1711    13399 a.out`mutex                   a.out`_$p1A12.main+0x30
    7    11600 libc.so.1`_uberdata           libc.so.1`_thr_exit_common+0xbc
 8111    10778 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x5c

Mutex unsuccessful spin

Count     nsec Lock                          Caller
-------------------------------------------------------------------------------
 1315    31766 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x20
  636    31008 a.out`mutex                   a.out`_$p1A12.main+0x30
29867    30875 a.out`mutex                   a.out`_$p1A12.main+0x30
   31    30590 libc.so.1`_uberdata           libc.so.1`_thr_exit_common+0xbc
 1304    29273 libc.so.1`libc_malloc_lock    a.out`_$p1A12.main+0x5c
    7    26320 libc.so.1`_uberdata           libmtsk.so.1`threads_fini+0x174
   19    24328 libc.so.1`_uberdata           libc.so.1`_thr_exit_common+0xbc
```

libc_malloc_lock   a.out`mutex

plockstat(1M) - example

- libc.so.1`libc_malloc_lock
  - > Use libmtmalloc or libumem
- a.out`mutex
  - > Resize the critical sections
    - > scope
    - > lock holding time
  - > Replace critical sections with atomic operations

Atomic Operations

- atomic_ops
  - atomic_ops package from HP
  - atomic_ops(3c), available on Solaris 10

    atomic_add_16(), atomic_or_32(), atomic_add_32_nv(), ...
- OpenMP: “atomic” directive

  ```
  #pragma omp atomic
  a++;
  ```
- Java: atomic variable classes
  - Nine flavors of atomic variables.

9

Portable & Predictable Performance

- Use the same program on different platforms and achieve consistent performance.
- Ideal: performance is determined only by the algorithm used.
- What affects performance?
  - Algorithm
  - Compiler
  - Runtime (VM)
  - OS
  - HW

Performance Analysis Tools

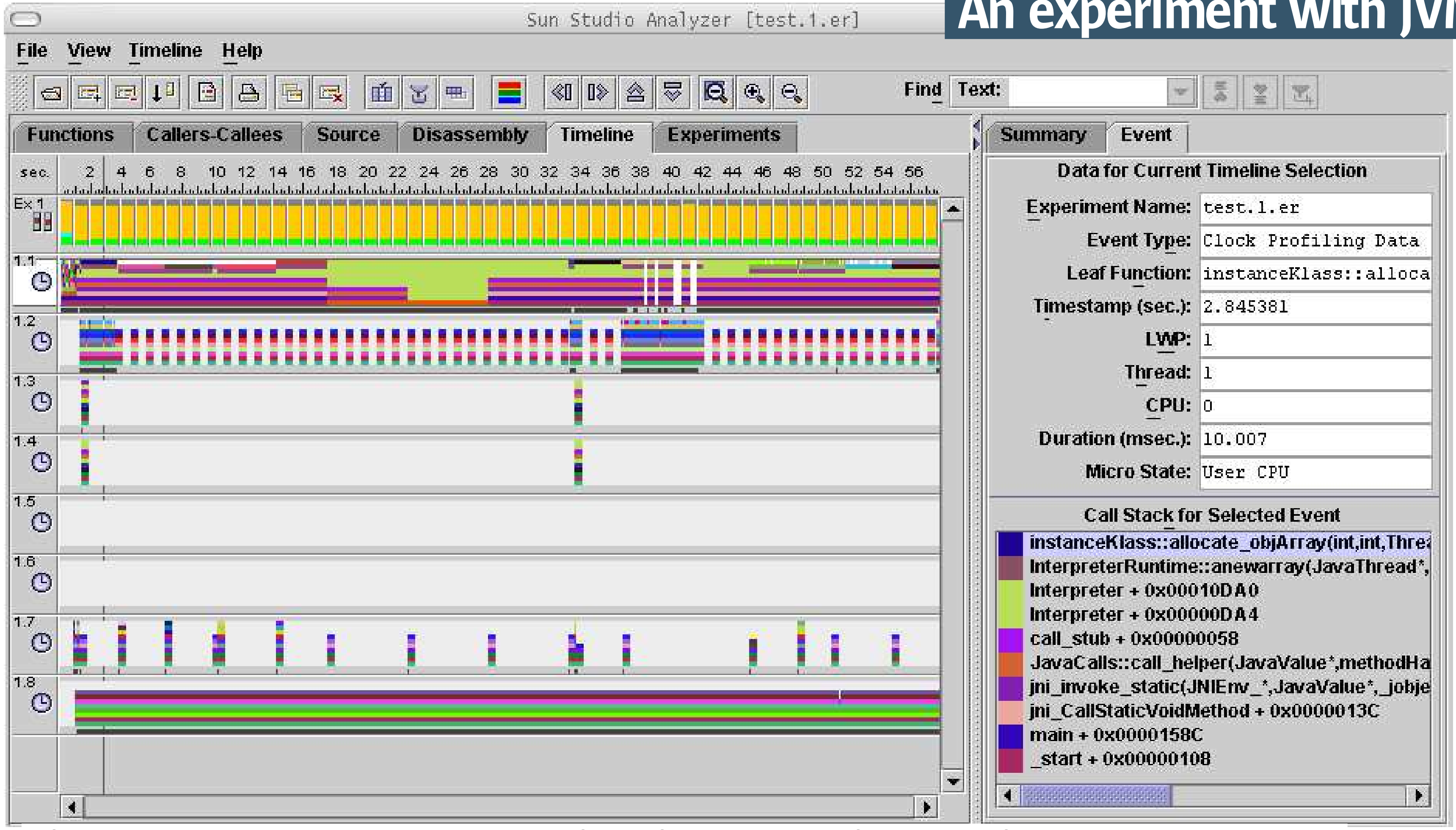
- Sun Studio Performance Analyzer
- Intel Vtune and Thread Profiler
- Quest Software JProbe
- Borland Optimizeit
- ...
- OS level tools: dtrace, mpstat, lockstat, ...

Sun Studio Performance Analyzer

- General purpose application level profiler
- Post-mortem analysis
- MT aware
- Supports dynamic libraries
- Clock profiling
- Hardware counter profiling
- Support for OpenMP

Sun Studio Performance Analyzer

An experiment with JVM


Sun Studio Analyzer [test.1.er]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Timeline Experiments
Summary Event
sec. 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56
Ex 1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8

Data for Current Timeline Selection

Experiment Name:	test.1.er
Event Type:	Clock Profiling Data
Leaf Function:	instanceKlass::alloca
Timestamp (sec.):	2.845381
LWP:	1
Thread:	1
CPU:	0
Duration (msec.):	10.007
Micro State:	User CPU

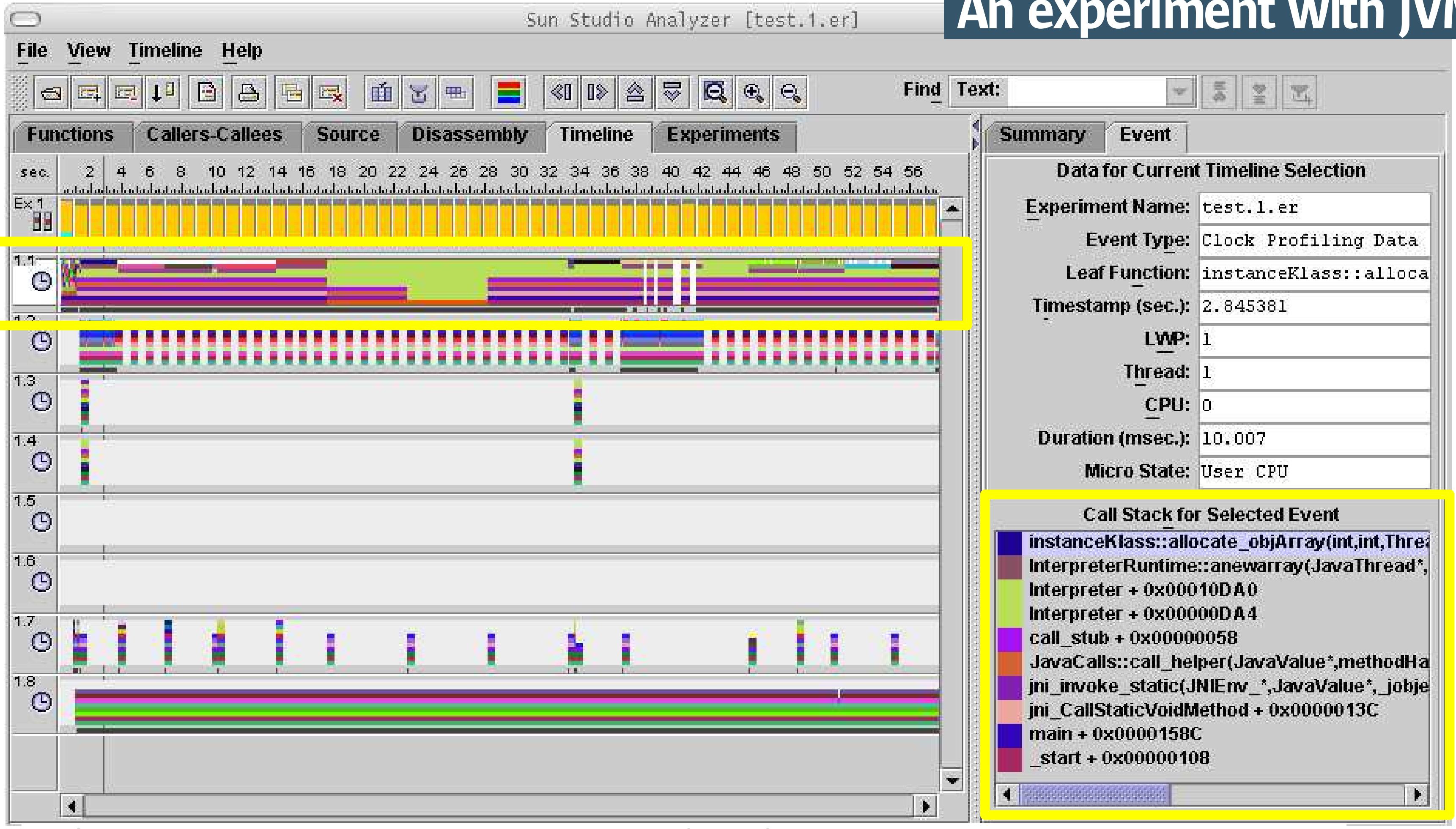
Call Stack for Selected Event

- instanceKlass::allocate_objArray(int,int,Threa
- InterpreterRuntime::anewarray(JavaThread*,
- Interpreter + 0x00010DA0
- Interpreter + 0x00000DA4
- call_stub + 0x00000058
- JavaCalls::call_helper(JavaValue*,methodHa
- jni_invoke_static(JNIEnv_*,JavaValue*,_jobje
- jni_CallStaticVoidMethod + 0x0000013C
- main + 0x0000158C
- _start + 0x00000108

Single threaded Java application. JVM is multi-threaded.

Sun Studio Performance Analyzer

An experiment with JVM



Sun Studio Analyzer [test.1.er]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Timeline Experiments
Summary Event
sec. 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56
Ex 1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8

Data for Current Timeline Selection

Experiment Name:	test.1.er
Event Type:	Clock Profiling Data
Leaf Function:	instanceKlass::alloca
Timestamp (sec.):	2.845381
LWP:	1
Thread:	1
CPU:	0
Duration (msec.):	10.007
Micro State:	User CPU

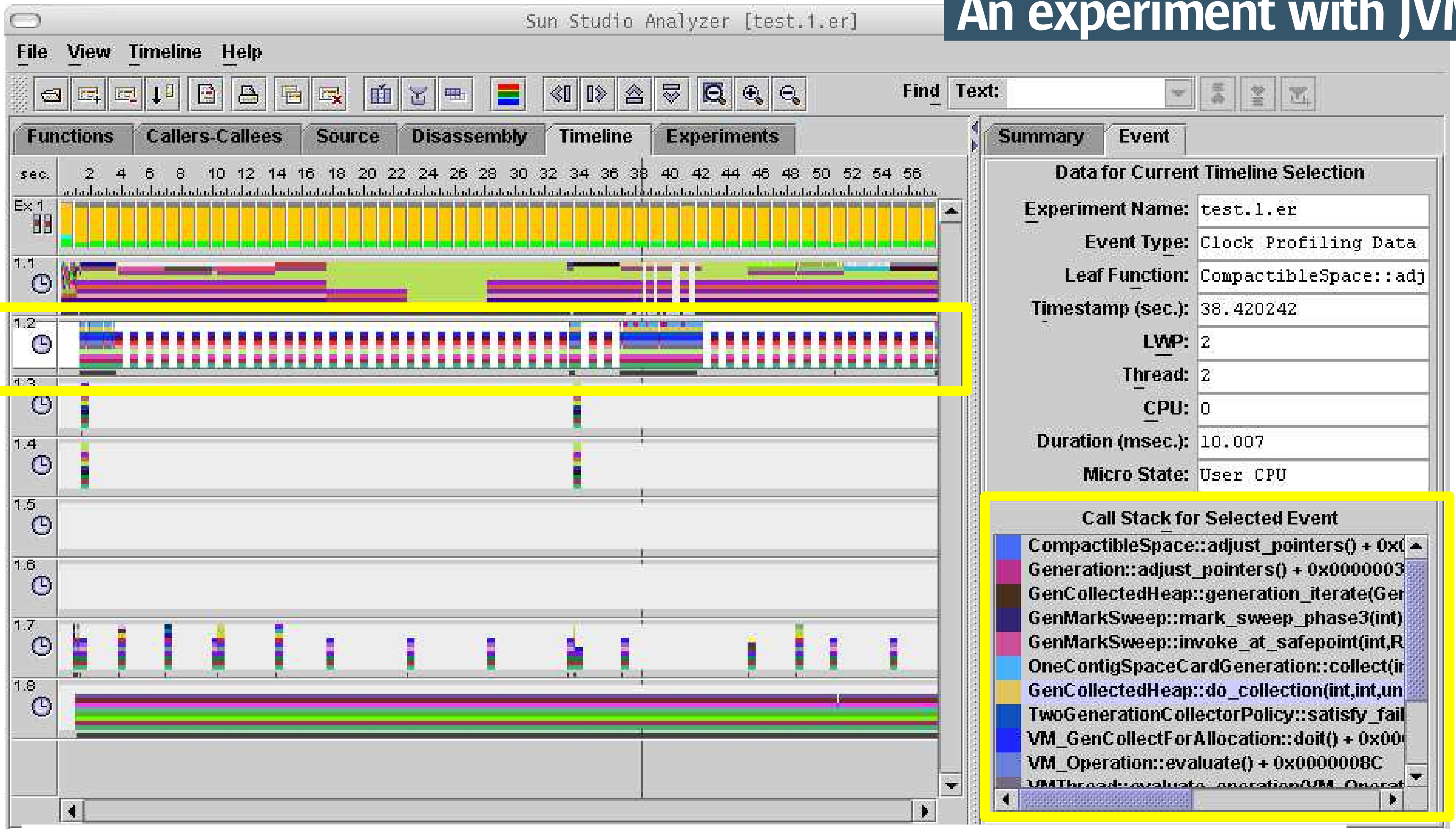
Call Stack for Selected Event

```
instanceKlass::allocate_objArray(int,int,Threa
InterpreterRuntime::anewarray(JavaThread*,
Interpreter + 0x00010DA0
Interpreter + 0x00000DA4
call_stub + 0x00000058
JavaCalls::call_helper(JavaValue*,methodHa
jni_invoke_static(JNIEnv_*,JavaValue*,_jobje
jni_CallStaticVoidMethod + 0x0000013C
main + 0x0000158C
_start + 0x00000108
```

This thread executes user application.

Sun Studio Performance Analyzer

An experiment with JVM



Data for Current Timeline Selection

Experiment Name:	test.1.er
Event Type:	Clock Profiling Data
Leaf Function:	CompactibleSpace::adj
Timestamp (sec.):	38.420242
LWP:	2
Thread:	2
CPU:	0
Duration (msec.):	10.007
Micro State:	User CPU

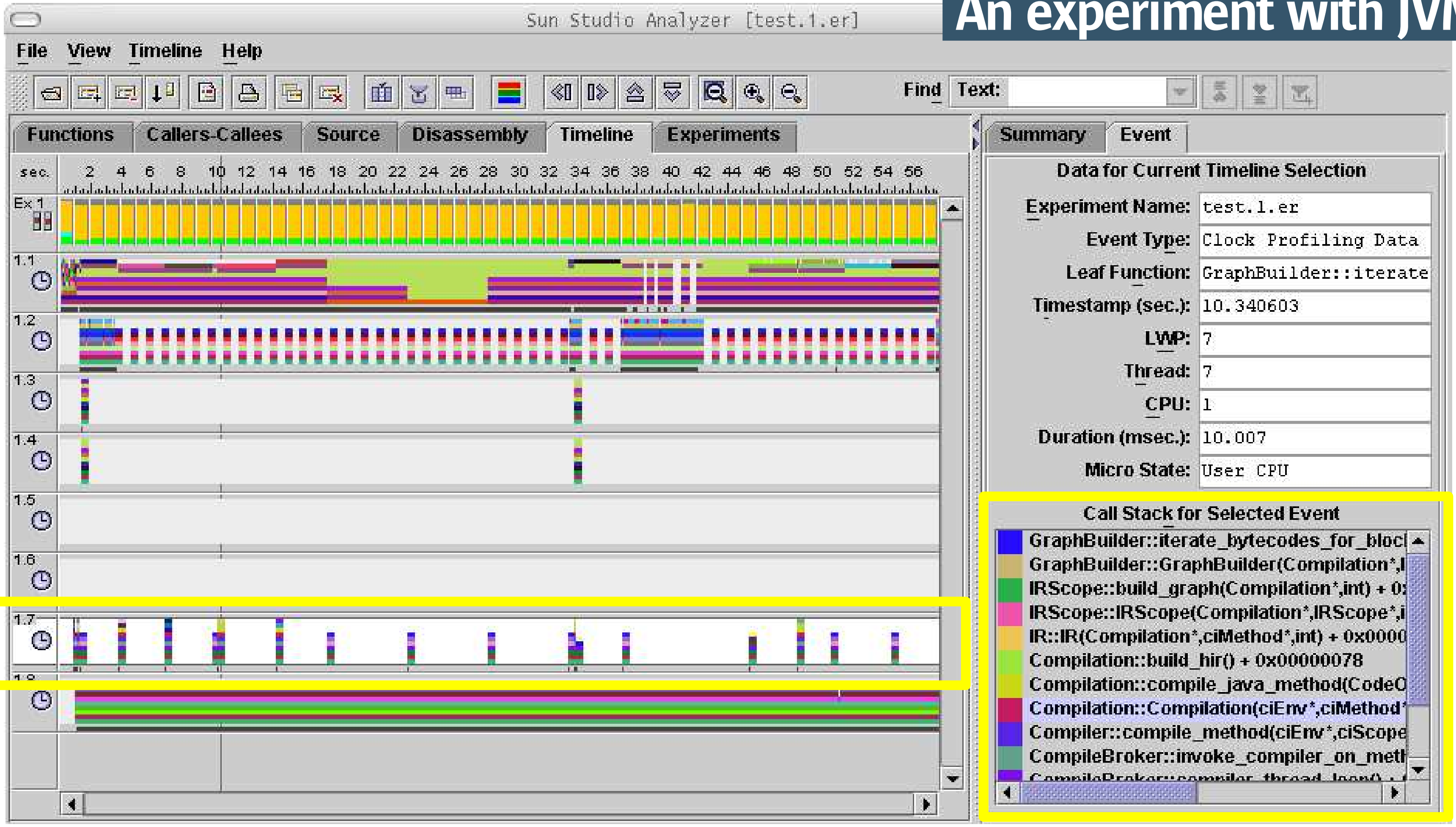
Call Stack for Selected Event

```
CompactibleSpace::adjust_pointers() + 0x0
Generation::adjust_pointers() + 0x0000003
GenCollectedHeap::generation_iterate(Ger
GenMarkSweep::mark_sweep_phase3(int)
GenMarkSweep::invoke_at_safepoint(int,R
OneContigSpaceCardGeneration::collect(ir
GenCollectedHeap::do_collection(int,int,un
TwoGenerationCollectorPolicy::satisfy_fail
VM_GenCollectForAllocation::doit() + 0x00
VM_Operation::evaluate() + 0x0000008C
```

This thread does garbage collection.

Sun Studio Performance Analyzer

An experiment with JVM


Sun Studio Analyzer [test.1.er]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Timeline Experiments
Summary Event
sec. 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56
Ex 1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8

Data for Current Timeline Selection

Experiment Name:	test.1.er
Event Type:	Clock Profiling Data
Leaf Function:	GraphBuilder::iterate
Timestamp (sec.):	10.340603
LWP:	7
Thread:	7
CPU:	1
Duration (msec.):	10.007
Micro State:	User CPU

Call Stack for Selected Event

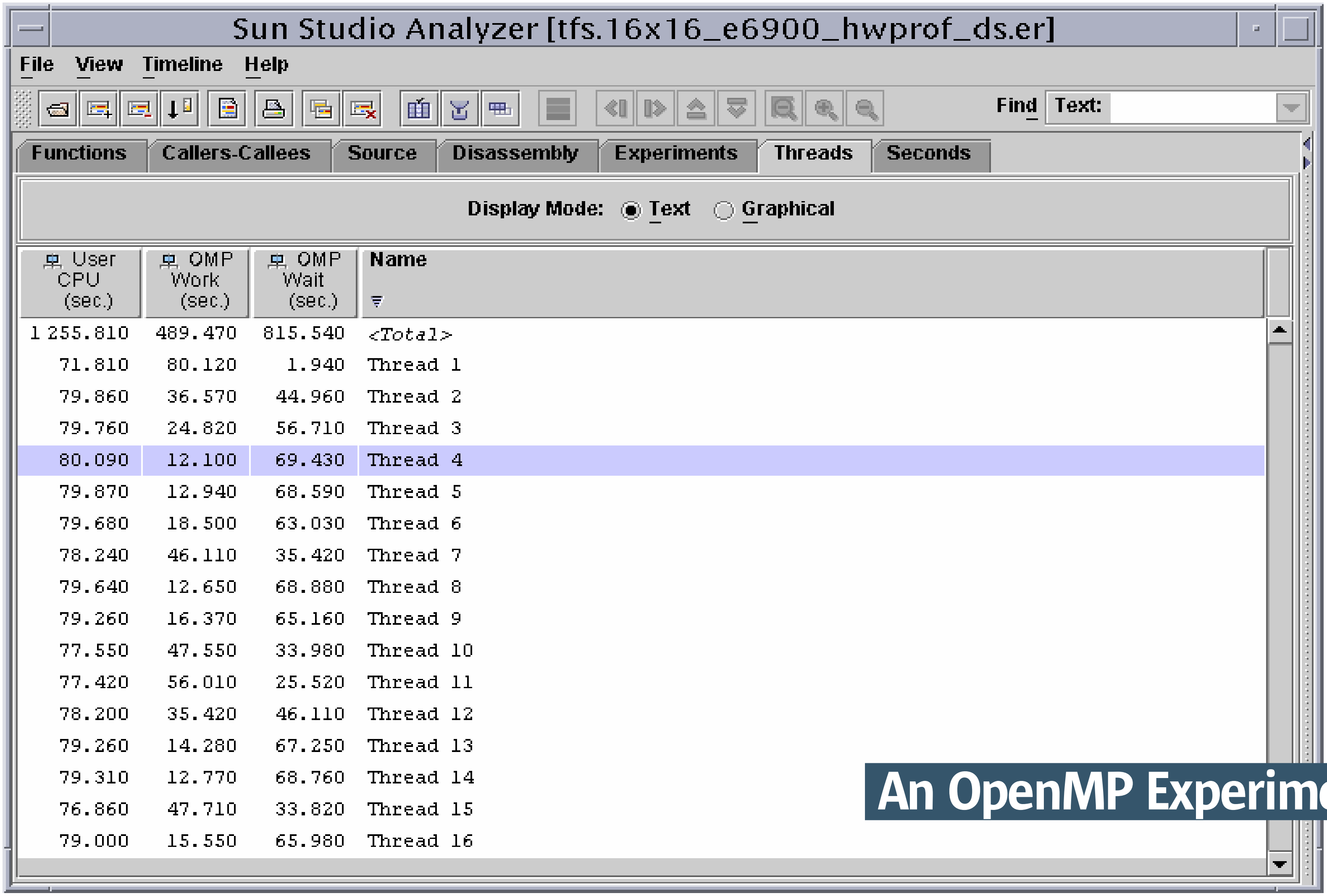
GraphBuilder::iterate_bytecodes_for_bloc
GraphBuilder::GraphBuilder(Compilation*,I
IRScope::build_graph(Compilation*,int) + 0:
IRScope::IRScope(Compilation*,IRScope*,i
IR::IR(Compilation*,ciMethod*,int) + 0x0000
Compilation::build_hir() + 0x00000078
Compilation::compile_java_method(CodeO
Compilation::Compilation(ciEnv*,ciMethod*
Compiler::compile_method(ciEnv*,ciScope
CompileBroker::invoke_compiler_on_meth
CompileBroker::compiler_thread_loop() +

This thread does JIT compilation.

Performance Tools: A Challenge

- Be aware of the threading model and provide relevant information.
  - Identify tasks, in addition to threads
  - Provide state information at different abstraction levels
  - Goals
    - Identify serial bottlenecks
    - Help resolve load imbalances

Sun Studio Performance Analyzer

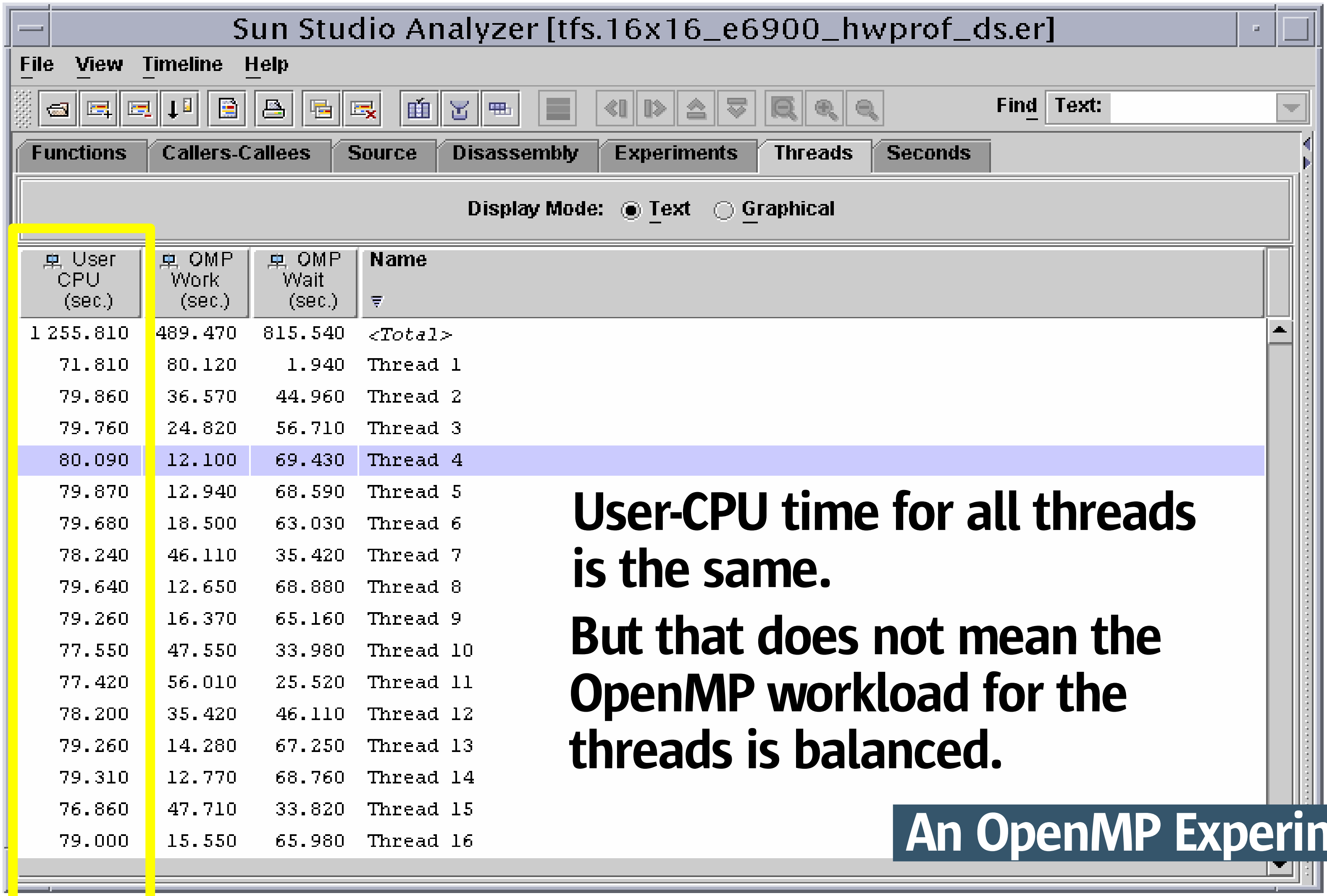

Sun Studio Analyzer [tfs.16x16_e6900_hwprof_ds.er]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Experiments Threads Seconds

Display Mode: [✓ Text] Graphical

User CPU (sec.)	OMP Work (sec.)	OMP Wait (sec.)	Name
1 255.810	489.470	815.540	*<Total>*
71.810	80.120	1.940	Thread 1
79.860	36.570	44.960	Thread 2
79.760	24.820	56.710	Thread 3
80.090	12.100	69.430	Thread 4
79.870	12.940	68.590	Thread 5
79.680	18.500	63.030	Thread 6
78.240	46.110	35.420	Thread 7
79.640	12.650	68.880	Thread 8
79.260	16.370	65.160	Thread 9
77.550	47.550	33.980	Thread 10
77.420	56.010	25.520	Thread 11
78.200	35.420	46.110	Thread 12
79.260	14.280	67.250	Thread 13
79.310	12.770	68.760	Thread 14
76.860	47.710	33.820	Thread 15
79.000	15.550	65.980	Thread 16

An OpenMP Experiment

Sun Studio Performance Analyzer


Sun Studio Analyzer [tfs.16x16_e6900_hwprof_ds.er]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Experiments Threads Seconds

Display Mode: [✓ Text] Graphical

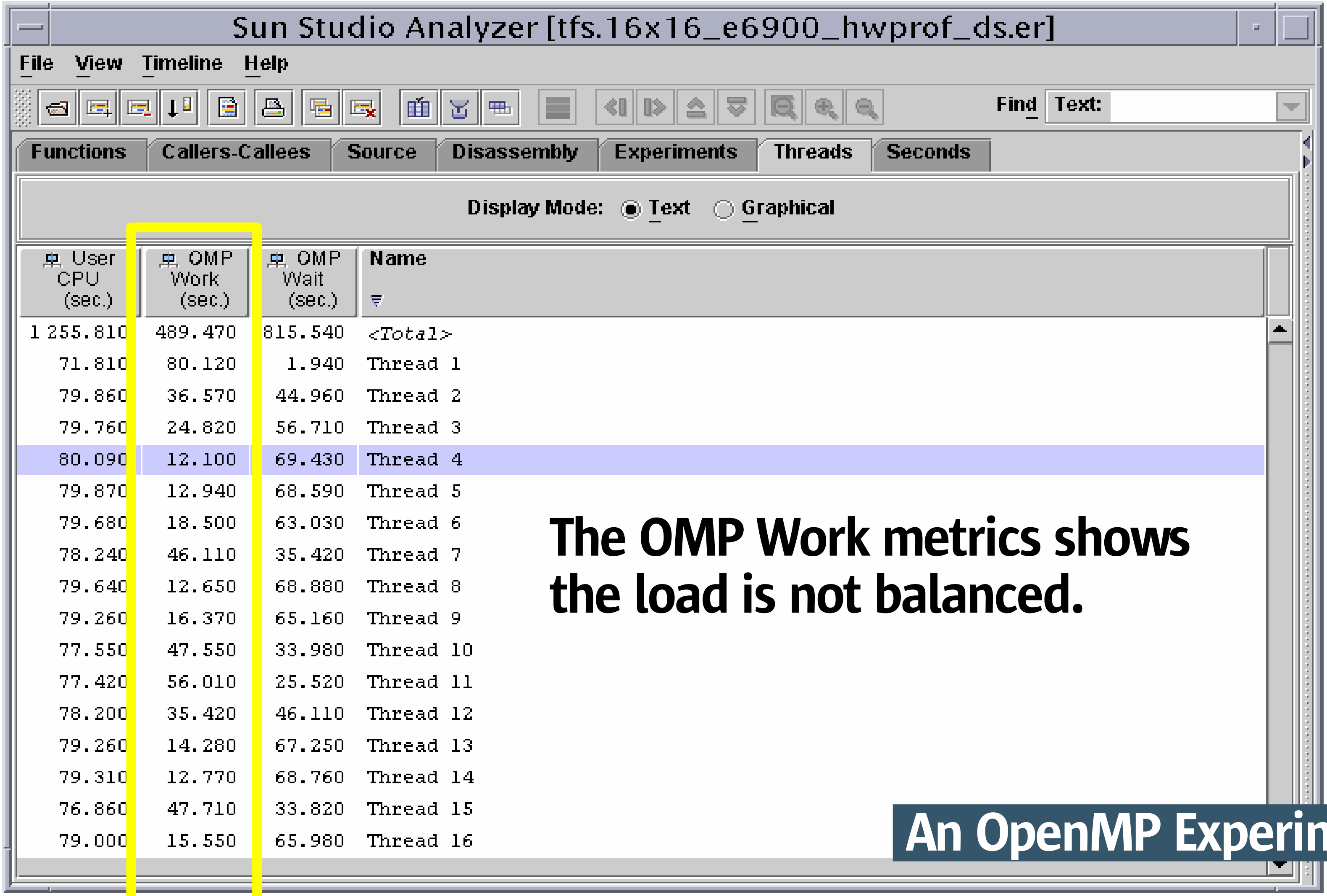
User CPU (sec.)	OMP Work (sec.)	OMP Wait (sec.)	Name
1 255.810	489.470	815.540	*<Total>*
71.810	80.120	1.940	Thread 1
79.860	36.570	44.960	Thread 2
79.760	24.820	56.710	Thread 3
80.090	12.100	69.430	Thread 4
79.870	12.940	68.590	Thread 5
79.680	18.500	63.030	Thread 6
78.240	46.110	35.420	Thread 7
79.640	12.650	68.880	Thread 8
79.260	16.370	65.160	Thread 9
77.550	47.550	33.980	Thread 10
77.420	56.010	25.520	Thread 11
78.200	35.420	46.110	Thread 12
79.260	14.280	67.250	Thread 13
79.310	12.770	68.760	Thread 14
76.860	47.710	33.820	Thread 15
79.000	15.550	65.980	Thread 16

User-CPU time for all threads is the same.

But that does not mean the OpenMP workload for the threads is balanced.

An OpenMP Experiment

Sun Studio Performance Analyzer


Sun Studio Analyzer [tfs.16x16_e6900_hwprof_ds.er]
File View Timeline Help
Find Text:
Functions Callers-Callees Source Disassembly Experiments Threads Seconds

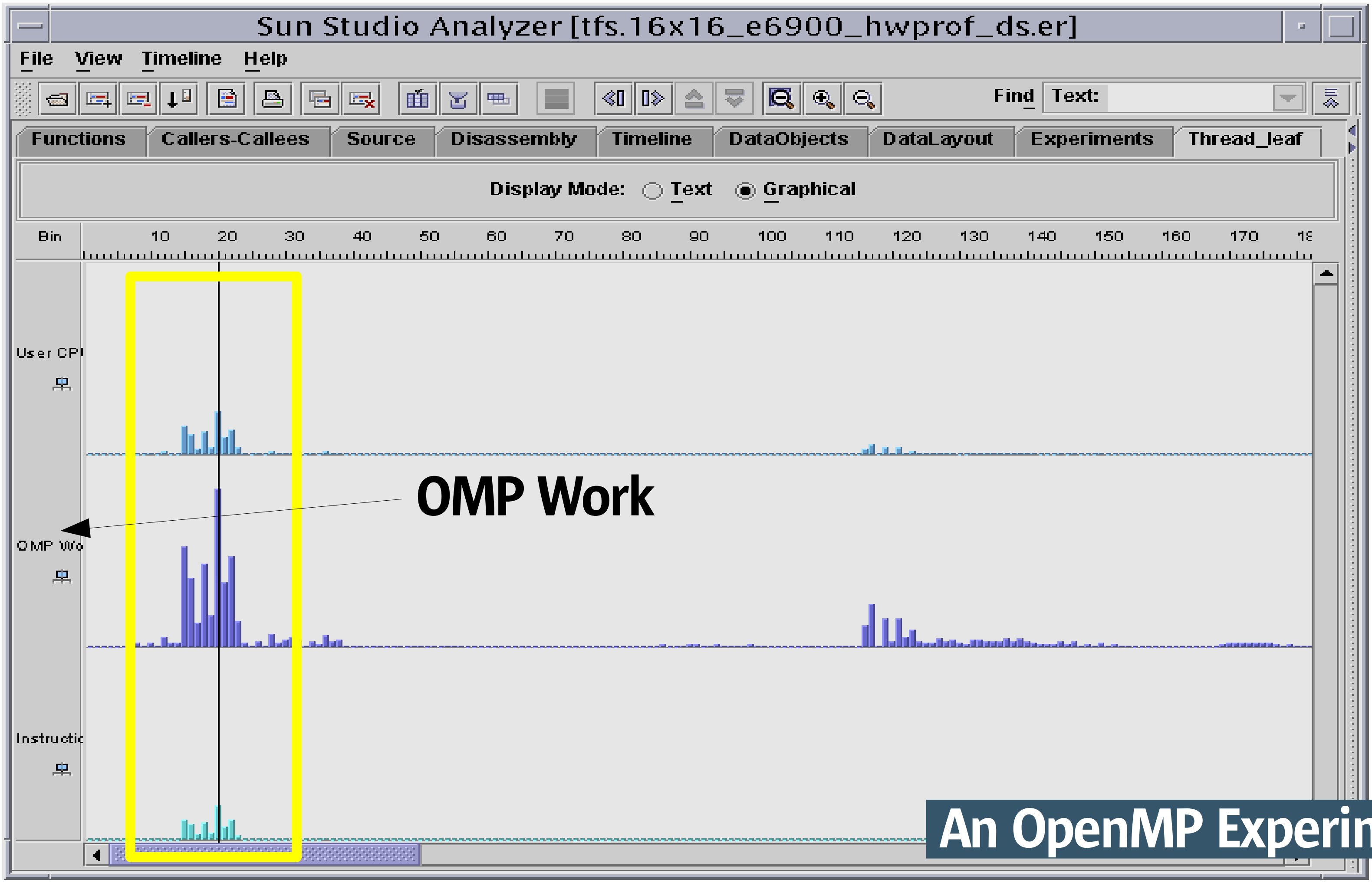
Display Mode: (●) Text ( ) Graphical

User CPU (sec.)	OMP Work (sec.)	OMP Wait (sec.)	Name
1 255.810	489.470	815.540	*<Total>*
71.810	80.120	1.940	Thread 1
79.860	36.570	44.960	Thread 2
79.760	24.820	56.710	Thread 3
80.090	12.100	69.430	Thread 4
79.870	12.940	68.590	Thread 5
79.680	18.500	63.030	Thread 6
78.240	46.110	35.420	Thread 7
79.640	12.650	68.880	Thread 8
79.260	16.370	65.160	Thread 9
77.550	47.550	33.980	Thread 10
77.420	56.010	25.520	Thread 11
78.200	35.420	46.110	Thread 12
79.260	14.280	67.250	Thread 13
79.310	12.770	68.760	Thread 14
76.860	47.710	33.820	Thread 15
79.000	15.550	65.980	Thread 16

The OMP Work metrics shows the load is not balanced.

An OpenMP Experiment

Sun Studio Performance Analyzer



Display Mode: Text [✓ Graphical]

10

Error Recovery

- Catch the errors and restore the application to a stable state.
- Error recovery becomes more difficult in mt applications because it may be an asynchronous problem!
  - Notify other threads
    - Who? How? When?
  - Update global structures
    - Deadlocks or infinite loops in recovering process!
- A common approach: use check points

11

All Single Thread Issues

- “50% chance a bug has nothing to do with multi-threading. It is just 200x difficult to debug in a multi-thread program”.
- Referencing un-initialized memory
- Out-of-bound array accesses
- Dereferencing a stale pointer
- ...
- Tools: Lint, Purify, Valgrind, dbx rtc, ...

MT- capable debuggers

- Basic Features
  - Examing the thread states
  - Examing thread specific data
  - Setting breakpoints for a specific thread
  - Control the execution of the threads
- Advanced Features
  - Integrated with deadlock/data race detection tools
  - Examing synchronization status
  - Be aware of threading model
- Tools
  - Dbx from Sun, TotalView from Etnus, GDB, ...

dbx – Examine Status

- Print a list of all known threads
  - `(dbx) threads`

```
o t@1 a l@1 ?() breakpoint in work()
*>t@2 a l@2 work() breakpoint in  work()
  t@3 a l@3 work() running in _thr_setup()
```

- Switch the viewing context to another thread
  - `(dbx) thread t@3`

dbx – Set Breakpoints

- A break point can be set specifically to a particular thread.

  -
    ```
    (dbx) stop at 430 -thread t@2
    ```

- Breaking is synchronous - “stop the world”
  - When any thread stops, all other threads *sympathetically* stop.

dbx - Control Execution

- Keep the given thread from ever running
  - > `(dbx) thread -suspend t@2`
- Resume a thread
  - > `(dbx) thread -resume t@2`
- Single step a thread
  - > `(dbx) step t@2`
  - > `(dbx) next t@2`

Summary

- Multi-core architecture provides a new thread-rich environment.
- Challenges in multi-threaded programming need to be addressed at all levels.
  - > languages
  - > compilers and tools
  - > libraries and utilities
  - > VM and OS
  - > programming practice

To Learn More ...

- Pthread
  - > "Programming with POSIX Threads", David Butenhof
- OpenMP
  - > www.openmp.org
- Java
  - > "Java Concurrency in Practice", Brian Goetz et al
  - > JSR-133
    - > http://www.cs.umd.edu/~pugh/java/memoryModel/
  - > JSR-166
    - > http://gee.cs.oswego.edu/dl/concurrency-interest/index.html

To Learn More ...

- “Developing and Tuning Applications on UltraSPARC T1 Chip Multithreading Systems”, Denis Sheahan
  - > http://www.sun.com/blueprints/1205/819-5144.html
- Dtrace
  - > http://www.sun.com/bigadmin/content/dtrace/
- Threading Methodolgy: Principles and Practices
  - > http://www.intel.com/cd/software/products/asmo-na/eng/threading/219349.htm
- Using the Sun Studio Data-Race Detection Tool
  - > http://developers.sun.com/prodtech/cc/downloads/drdt/using.html

Thank you!

yuan.lin@sun.com