



Remember when
the sky was the limit?

Multi-core Software Development: Encouraging an Industry Transition

Paul Petersen
and many others ...



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel may make changes to specifications, product descriptions, and plans at any time, without notice.

All dates provided are subject to change without notice.

* Other names and brands may be claimed as the property of others.

Copyright © 2006, Intel Corporation.



Multi-core Software Development: Encouraging an Industry Transition

Motivation

Methodology

Tools

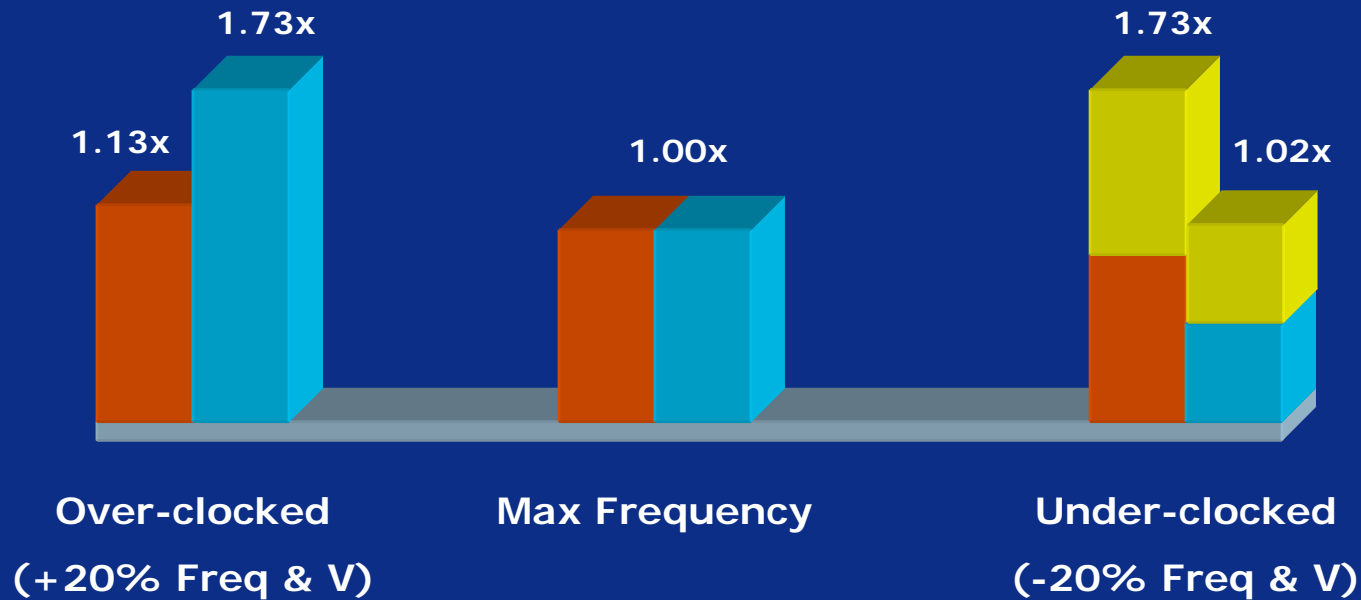
Conclusion



The Future is Now

- Parallel computing was touted as the wave of the future
- But for many reasons we have been able to hold off this eventuality ... until now
- Why is this different now?
 - Physics
 - Transistor densities continue to improve
 - But, voltage scaling slows down
 - Economics
 - Desire for increased performance
 - But, with more energy efficiency

Multi-Core Energy Efficient Performance



Relative single-core frequency and Vcc

Multi-core Processors ... Just the Beginning ...



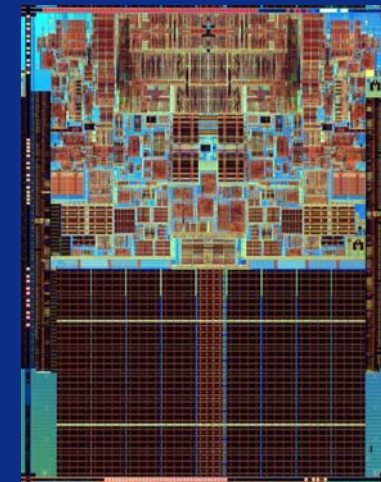
Montecito

1.72 Billion transistors

~580 mm²

90 nanometer design

(Not Actual Size)



Intel® Core™2 Duo

291 Million transistors

~143 mm²

65 nanometer design



For the Short Term ... or the Long Haul

- Multi-core is taking what we learned from SMPs
 - ... and moving it onto a single chip
 - Parallel supercomputers in your pocket!
- For the long haul the industry is going back to rethink the CPU and the whole platform to make every facet of the system as scalable as you can
- So don't just think of 2, 4 or 8 cores
- We need to get people to start thinking about 16, 32 and beyond

Multi-core Software Development: Encouraging an Industry Transition

Motivation

Methodology

Tools

Conclusion



But what about software?

- We have learned from the introduction of Hyper-Threading
 - Successful ISV enabling effort and industry adoption
 - But, many of the applications were hard-coded for two functional threads
- The number of cores is a parameter that is expected to increase
 - The products today may only have 2 cores on a chip
 - In a few generations they may have 8 or more cores
- Software is designed to allow different data sizes
- The applications also need to adapt to different machine sizes
- We must transform the software industry
 - And that is very hard to do
 - ... especially if you are a hardware company such as Intel

Intel's multi-core ramp accelerates in 2006

Intel® dual-core volume exiting 2006**

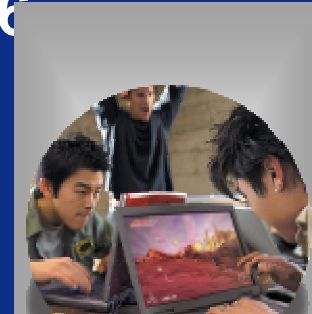
Desktop Performance >70% Dual-Core

Server >85% Dual-Core

Mobile Performance >70% Dual-Core

**Data is run-rate exiting the year. All products and dates are preliminary and subject to change without notice.

Intel single core to multi-core shipments



Desktop Client



Server & Workstation



Mobile client

By 2007, the majority of Intel® platforms will be multi-core

Standard Practice Today

- Applications that exploit multi-core hardware are typically written with
 - Operating System Threads (Win32* or POSIX*)
 - Managed Environment Threads (.NET* or Java*)
 - OpenMP*
- Implicit parallel approaches not practical
 - Automatic Parallelization is available but not widely used
 - Vectorization usually limited to improving ILP (instruction level parallelism)
- Programmers must “re-invent the wheel” for widely used parallel programming utilities
 - These utilities are difficult to write correctly and efficiently
 - Code often becomes very dependent on a particular OS’s threading facilities
- Parallelism is approached as a performance optimization



Multi-core Software Development: Encouraging an Industry Transition

Motivation

Methodology

Tools

Conclusion



Intel® Thread Checker 3.0 for Windows* and Linux*

Create Threads Faster

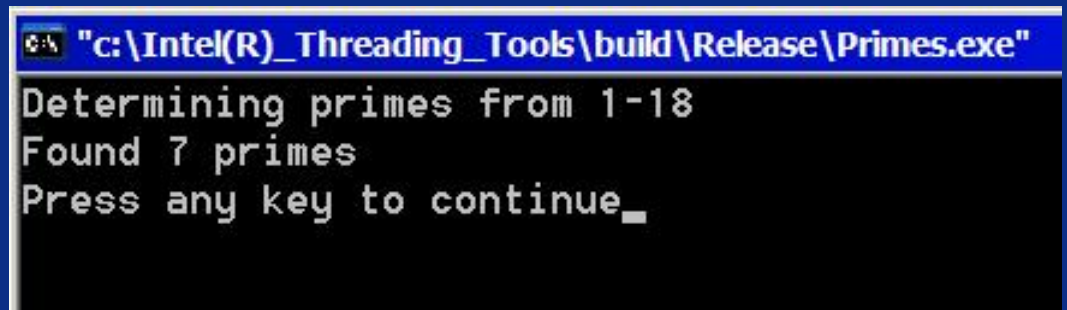
- Detects challenging data races and deadlocks
- Pinpoints errors to the source code line
- Works on standard debug builds without recompiling
- Supports 32-bit and 64-bit applications **New**
- Batch scripts integration for regression test runs **New**
- Recommends modules to instrument by usage **New**
 - Minimize instrumentation overhead
- Linux* **New**
 - Introduction of native Linux* support through command line views
- Windows
 - Supports Microsoft Visual Studio 2005 **New**



Example: Prime Number Kernel

2
3 3
5 3,5
7 3,5,7
9 3
11 3,5,7,9,11
13 3,5,7,9,11,13
15 3
17 3,5,7,9,11,13,15,17

```
for (long number = start; number < end; number += stride ) {  
    long factor = 3;  
    while ( (number % factor) != 0 ) factor += 2;  
    if ( factor == number ) {  
        Primes[ PrimeCount ] = number;  
        PrimeCount++;  
    }  
}
```



```
"c:\Intel(R)_Threading_Tools\build\Release\Primes.exe"  
Determining primes from 1-18  
Found 7 primes  
Press any key to continue_
```

Example: Not Quite Right

```
#pragma omp parallel for
for (long number = start; number < end; number += stride ) {
    long factor = 3;
    while ( (number % factor) != 0 ) factor += 2;
    if ( factor == number ) {
        Primes[ PrimeCount ] = number;
        PrimeCount++;
    }
}
```

```
C:\Primes\Release>Primes.exe
Determining primes from 1-100000
Found 9592 primes

C:\Primes\Release>Primes.exe
Determining primes from 1-100000
Found 9589 primes

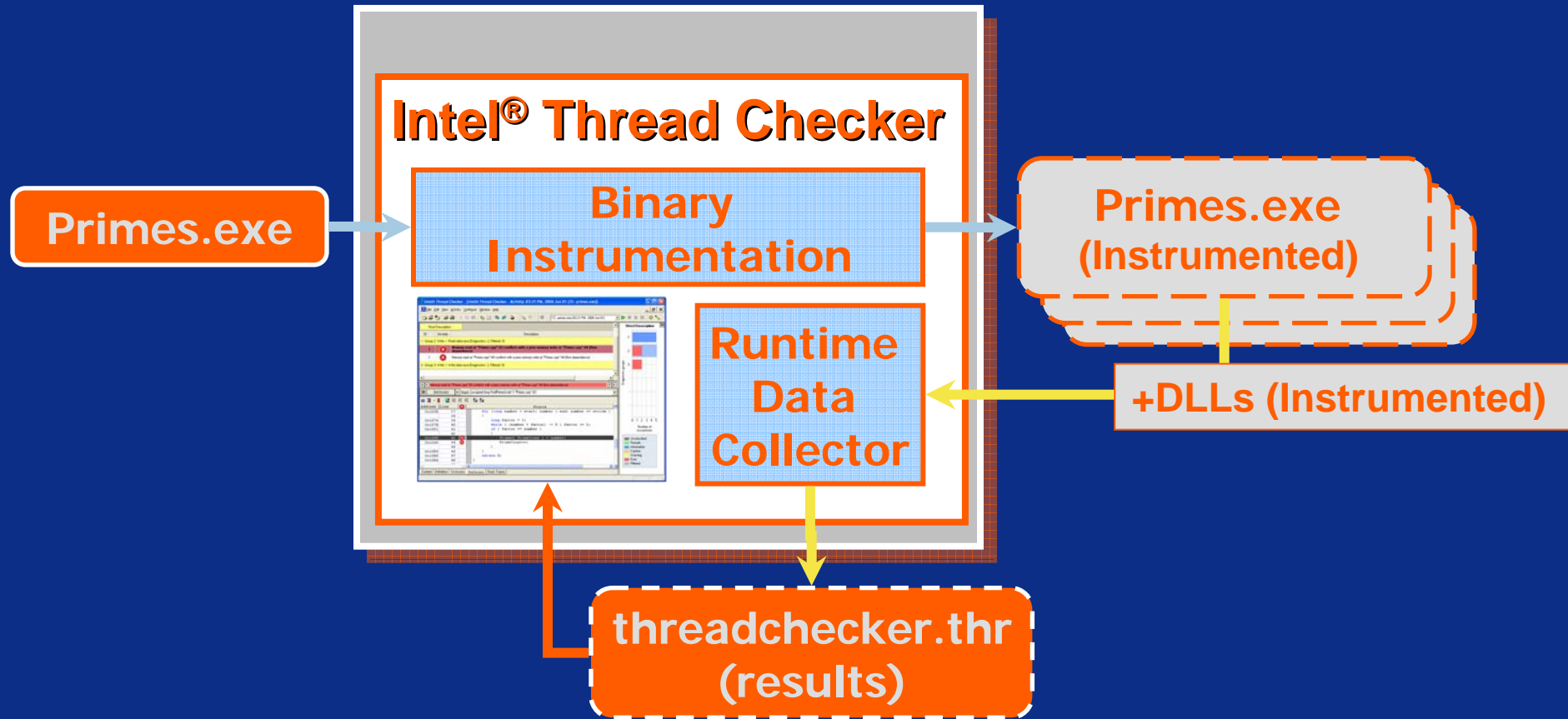
C:\Primes\Release>Primes.exe
Determining primes from 1-100000
Found 9590 primes

C:\Primes\Release>Primes.exe
Determining primes from 1-100000
Found 9588 primes

C:\Primes\Release>Primes.exe
Determining primes from 1-100000
Found 9591 primes

C:\Primes\Release>
```

Thread Checker Phases



Win32* threads, POSIX* threads, OpenMP*

User Interface (Windows)

The screenshot displays the Intel Thread Checker application window. The title bar reads "Intel® Thread Checker - [Intel® Thread Checker - Activity: 03:21 PM, 2006 Jun 01 (TC: primes.exe)]". The menu bar includes File, Edit, View, Activity, Configure, Window, and Help. The toolbar contains various icons for file operations and execution control.

The main area shows a list of diagnostic groups. Group 2, "Write -> Read data-race (Diagnostics: 2; Filtered: 0)", is expanded to show two errors:

- 1. Memory read at "Primes.cpp":43 conflicts with a prior memory write at "Primes.cpp":44 (flow dependence)
- 2. Memory read at "Primes.cpp":44 conflicts with a prior memory write at "Primes.cpp":43 (flow dependence)

Group 3, "Write -> Write data-race (Diagnostics: 2; Filtered: 0)", is also visible.

A yellow callout box with the text "PINPOINTS SOURCE CODE" points to the first error. Below the error list, a detailed view of the selected error is shown. The stack trace indicates the "2nd Access" at "Primes.cpp":43. The source code is displayed in a table with columns for Address, Line, and Source:

Address	Line	Source
0x105B	37	for (long number = start; number < end; number += stride)
	38	{
0x1074	39	long factor = 3;
0x107B	40	while ((number % factor) != 0) factor += 2;
0x1091	41	if (factor == number)
	42	{
0x1099	43	Primes[PrimeCount] = number;
0x10A9	44	PrimeCount++;
	45	}
0x10B6	46	}
0x10B8	47	return 0;
0x10BA	48	}

At the bottom, there are tabs for Context, Definition, 1st Access, 2nd Access, and Stack Traces.

On the right side, there is a "Short Description" panel with a bar chart showing the number of occurrences for different diagnostic groups. The legend indicates the following categories:

- Unclassified (Grey)
- Remark (Green)
- Information (Blue)
- Caution (Yellow)
- Warning (Light Yellow)
- Error (Red)
- Filtered (Light Grey)

Example: A Bit Better

```
#pragma omp parallel for
for (long number = start; number < end; number++)
    long factor = 3;
    while ( (number % factor) != 0 ) factor++;
    if ( factor == number ) {
#pragma omp critical
    {
        Primes[ PrimeCount ] = number;
        PrimeCount++;
    }
}
}
```

```
Command Prompt
C:\Primes\Release>Primes
Determining primes from 1-100000
Found 9592 primes
C:\Primes\Release>Primes
Determining primes from 1-100000
Found 9592 primes
C:\Primes\Release>Primes
Determining primes from 1-100000
Found 9592 primes
C:\Primes\Release>Primes
Determining primes from 1-100000
Found 9592 primes
C:\Primes\Release>Primes
Determining primes from 1-100000
Found 9592 primes
C:\Primes\Release>_
```

Definitions

Thread

- An Independent Sequence of Instructions

Sync Instruction

- Creates a thread
- Destroys a thread
- Posts information about host thread
- Brings information posted by another thread to host thread

Segments

- Parts of a thread separated by sync instructions

Data race: Exists between two segments S and S' if

- The segments are parallel
- There is a location x in shared memory that both segments access, and at least one of the accesses is a “write” access

Precedence Relation between Segments

Partial Order

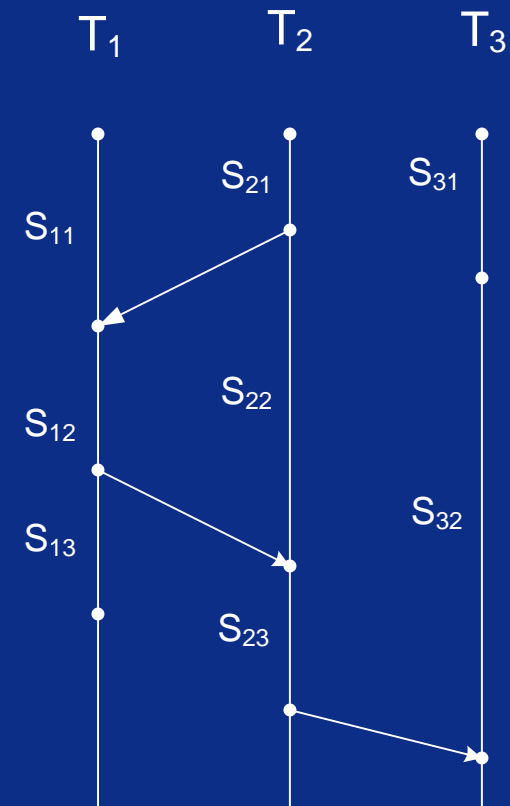
$$S_{11} < S_{12} < S_{13}$$

$$S_{21} < S_{22} < S_{23}$$

$$S_{31} < S_{32}$$

$$S_{21} < S_{12} < S_{23}$$

$$S_{11} < S_{23}$$



Parallel Segments

Two segments S and S' are parallel, if $S < S'$ and $S' < S$ are both false

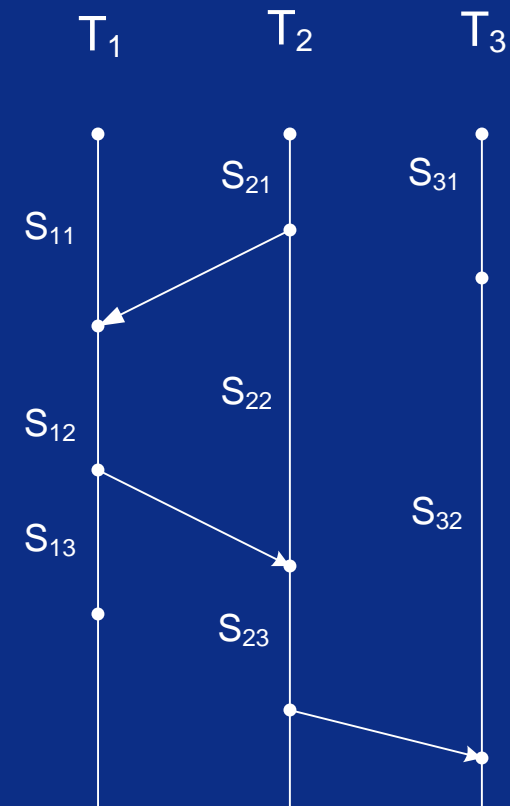
Examples:

S_{11} and S_{21}

S_{11} and S_{22}

S_{11} and S_{32}

S_{22} and S_{31} .



Vector Clock of a Segment

Vector clocks represent numerically the precedence relation between segments.

Vector Clock V_S of a segment S is a function on threads with values in $\{0, 1, 2, \dots\}$.

Take a segment S on a thread T .

Take a thread T' known to S .

Then $V_S(T')$ is one more than the number of segments S' on T' such that $S' < S$.

Vector clock of a segment is computed from the vector clocks of its immediate predecessors.

Conditions in Terms of Vector Clocks

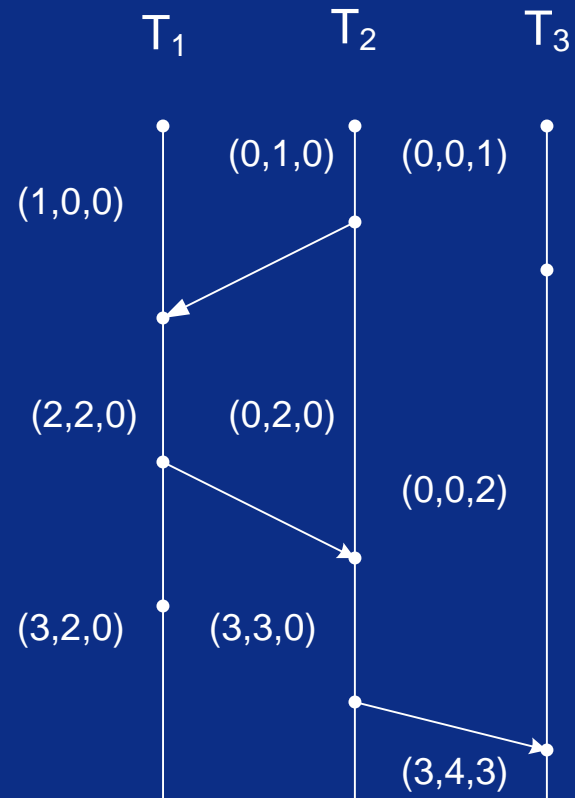
Let S denote a segment on a thread T and S' a segment on a thread T' .

$S < S'$ iff $V_S(T) < V_{S'}(T)$.

S and S' are parallel iff

$$V_S(T) \geq V_{S'}(T)$$

$$V_{S'}(T') \geq V_S(T').$$



Summary : Thread Checker

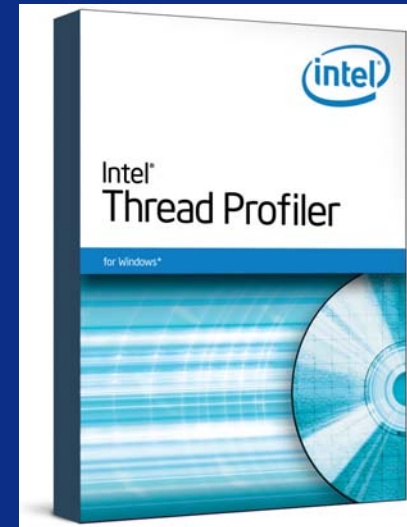
- The Intel® Thread Checker provides a safety net for developers
 - Increases confidence in developers ability to debug threaded software.
 - The techniques used by the Thread Checker allow analysis of “production” applications.
 - Industry direction moving to multi-core solutions makes confidence tools even more critical to success
- But we need more, performance issues in parallel programs are typically treated as bugs too
 - Otherwise you might as well have written a serial application

Intel® Thread Profiler 3.0 for Windows*

Optimize Threads Faster

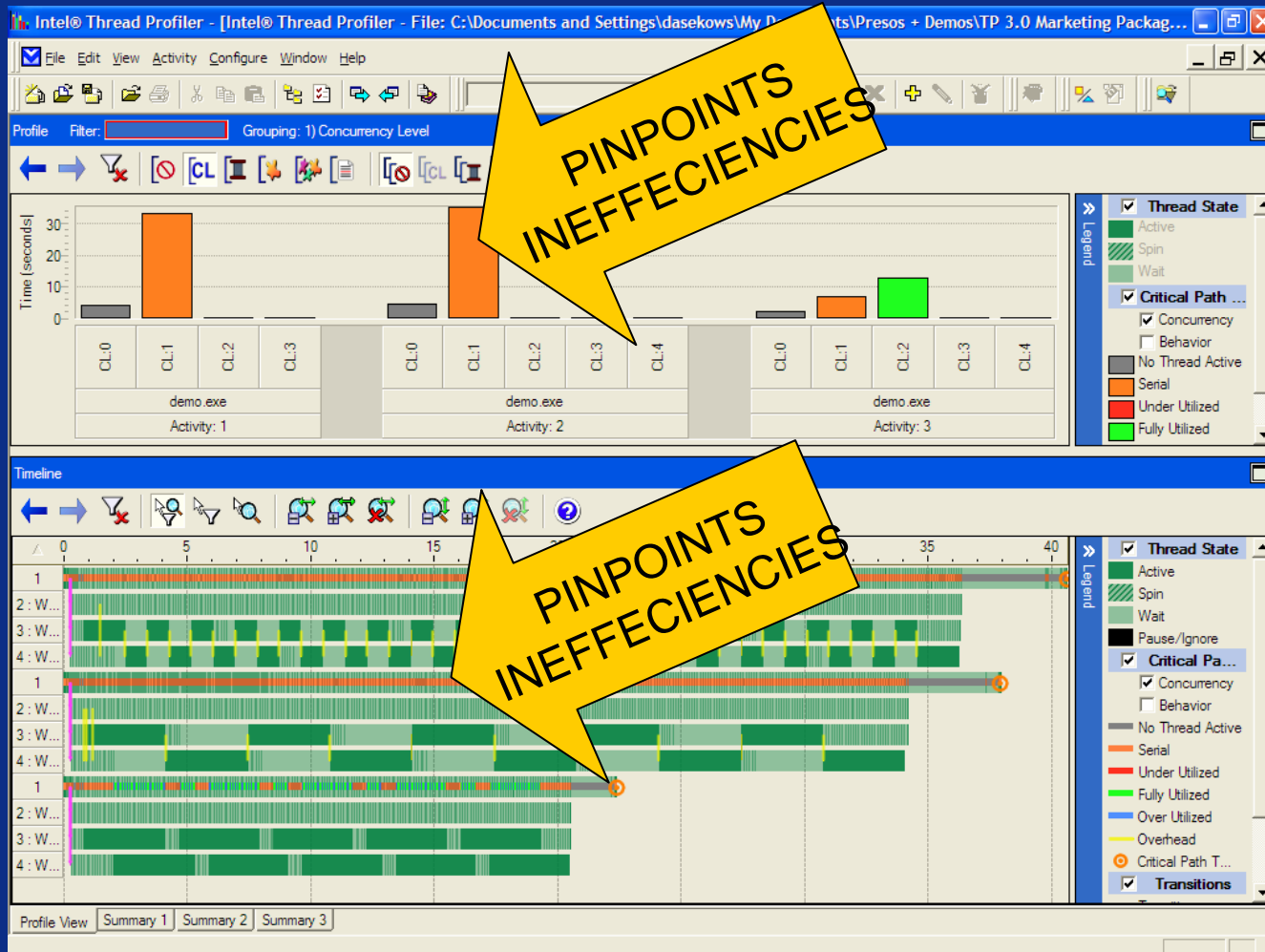
Key Benefits

- Shows how much of your application is not optimally parallel and where
- Identifies where thread specific overhead impacts performance
- Highlights thread workload imbalances and thread activity
- Shows the number of cores utilized
- Pinpoints issues to the source code line
- Maximizes application time spent in parallel regions
- Supports 32 and 64-bit applications **New**
- Supports Microsoft Visual Studio 2005* **New**

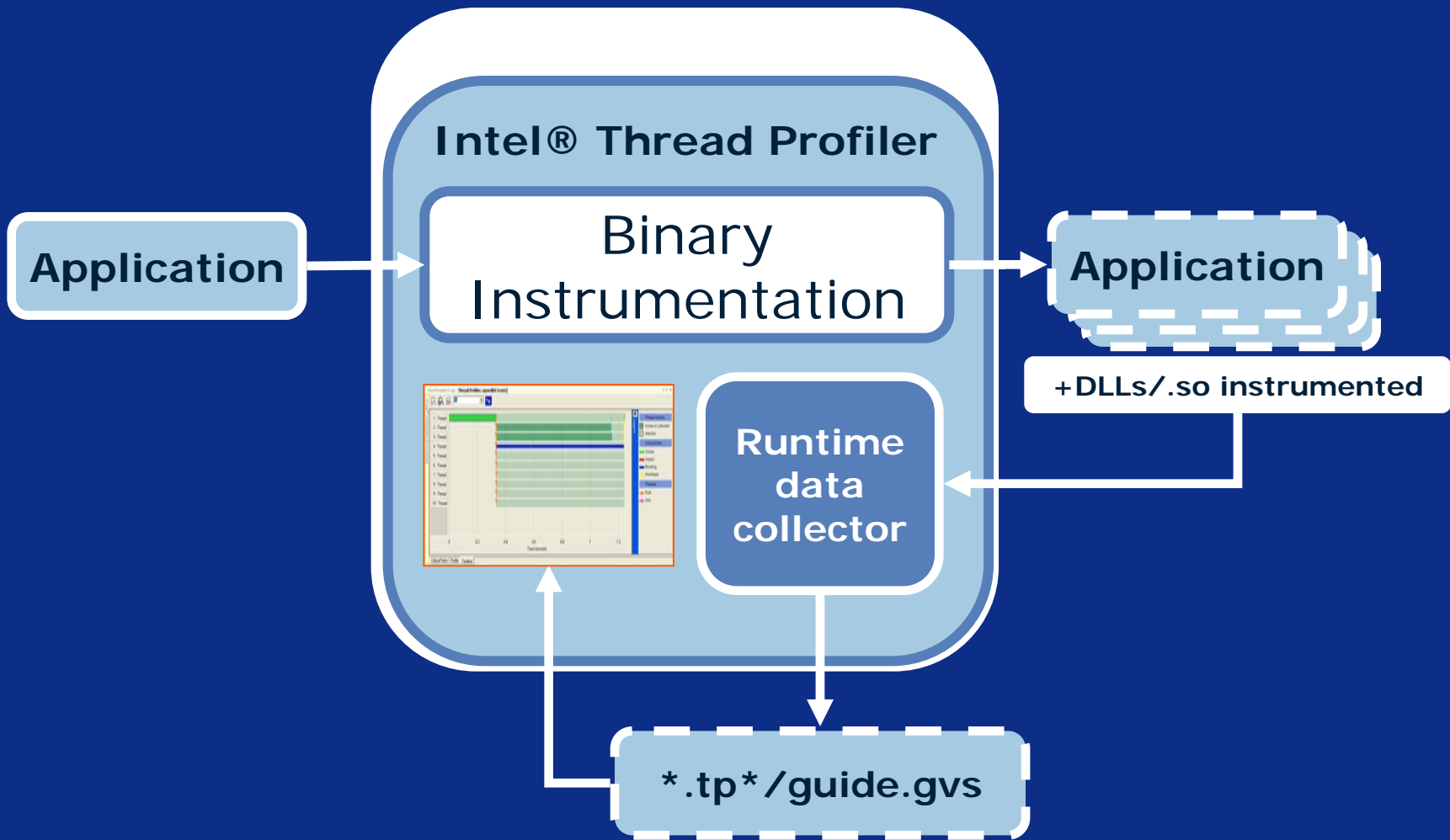


Intel® Thread Profiler

Pinpoints threading inefficiencies



Thread Profiler Phases

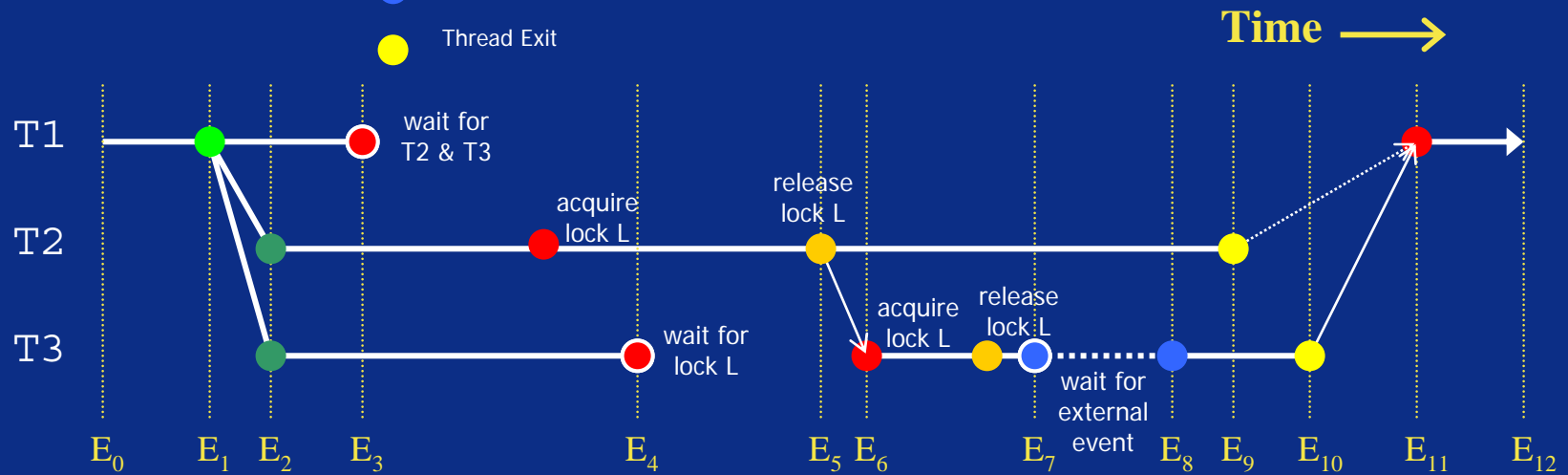


Instrumentation

Usually low run-time overhead because only select events are instrumented
 Target overhead of less than 2x slowdown with reasonable synchronization
 common case is much less

RECORDED EVENTS

- Create Thread (Fork)
- Thread Entry
- Wait for synchronization object or event
- Acquire synchronization object or event
- Release or signal synchronization object or event
- Wait for external event
- Receive external event
- Thread Exit



Critical Path Analysis

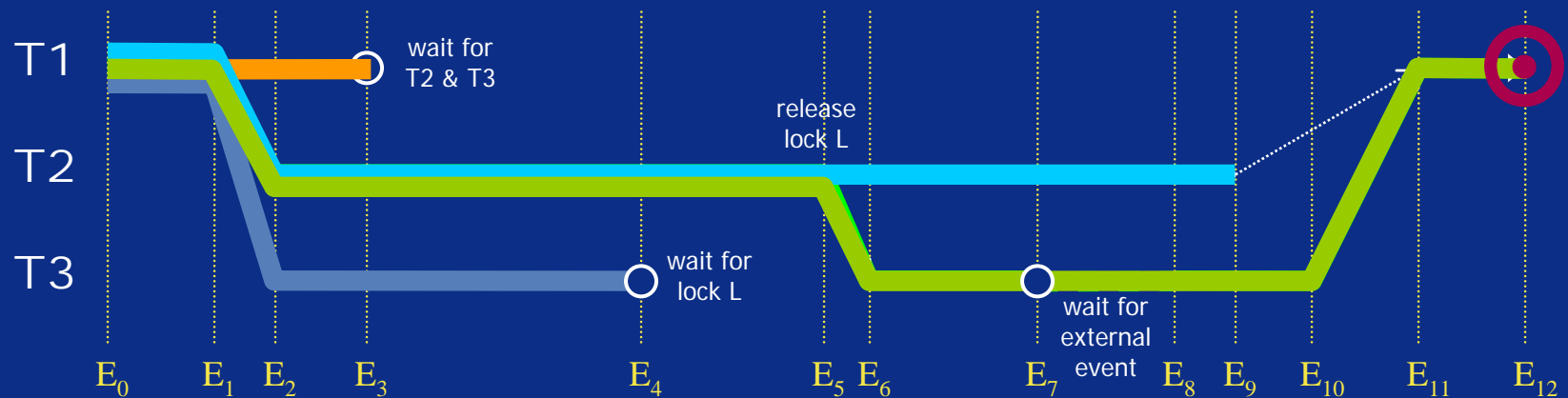
Flow **splits** when a thread **creates** a new thread or **signals (unblocks)** another thread

Flow **ends** when a thread **waits** for another thread or **terminates**

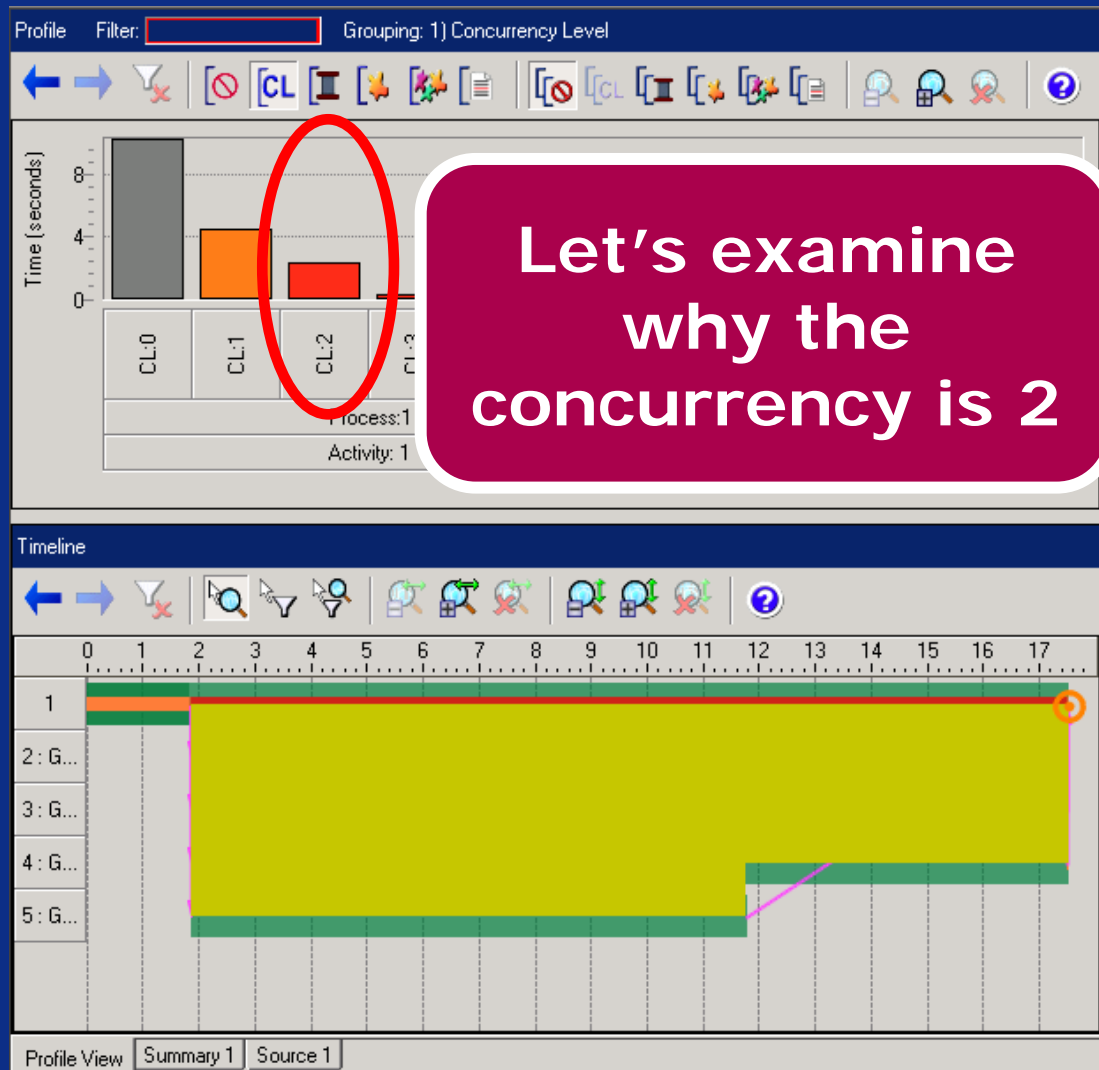
The **continuous flow to target location** is the **critical path**

Default target is the program termination

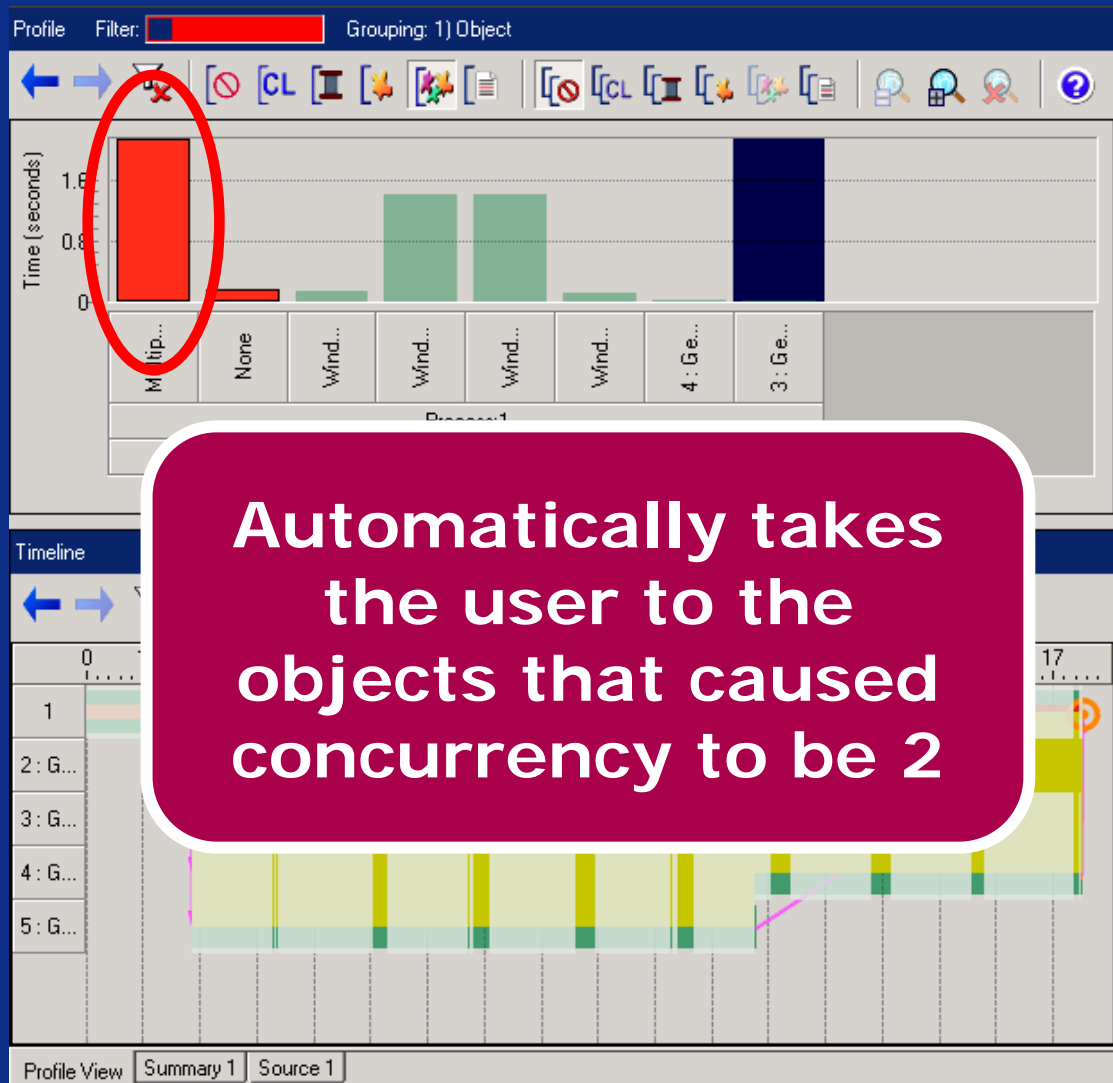
Where to focus optimization energy



Thread Profiler Usage Model (1)

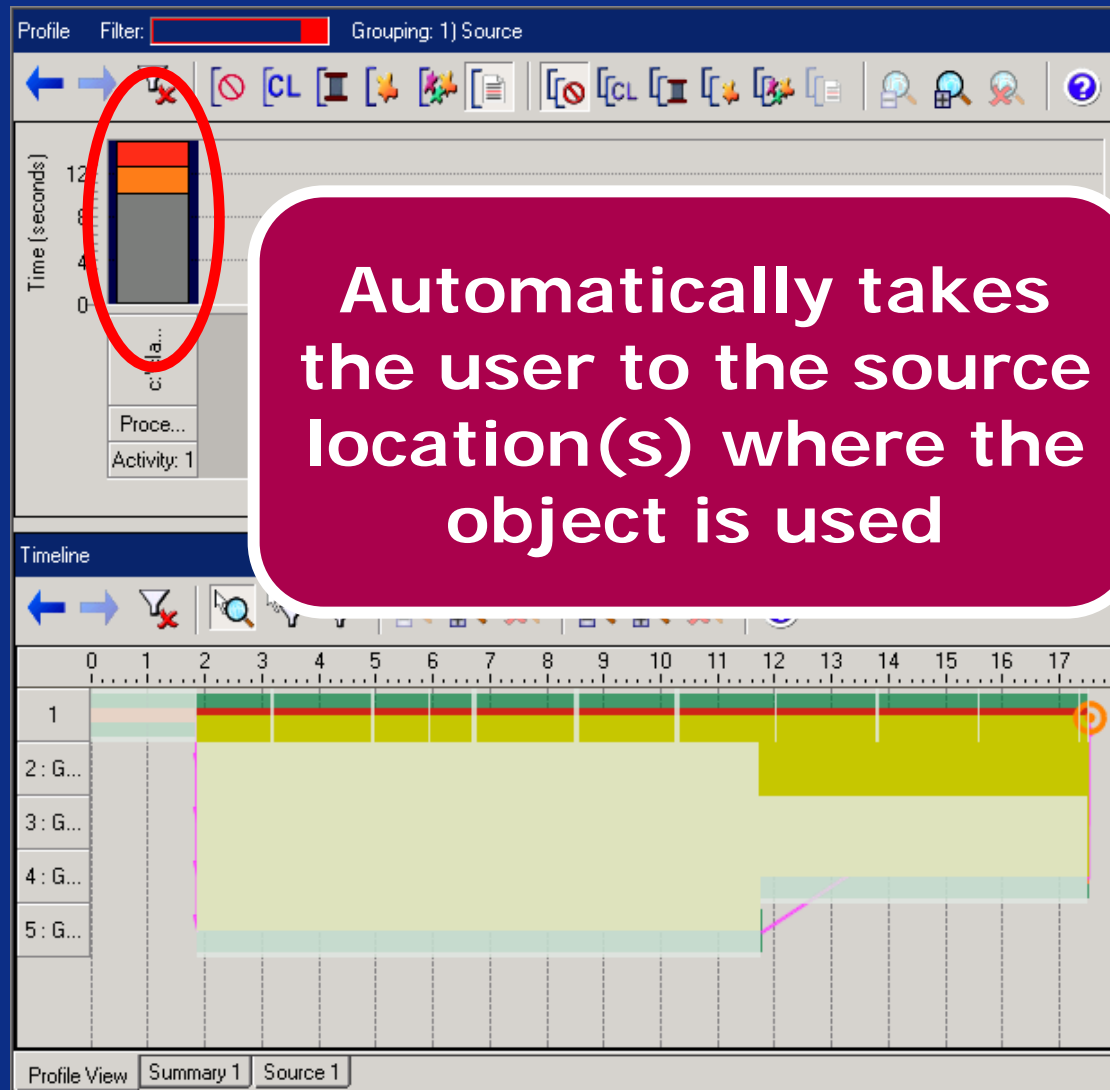


Thread Profiler Usage Model (2)



Automatically takes the user to the objects that caused concurrency to be 2

Thread Profiler Usage Model (3)



Thread Profiler Usage Model (4)

The screenshot shows a 'Source View' window from a thread profiler. On the left, a 'Blocked' pane displays a stack of threads. The top thread is 'GenerateMandelbrot' at address 0x220, with a path of 'c:\classfiles\tmp\itp\lab-3'. Below it are several 'Call to an uninstrumented module on the stack' entries. The current thread is 'Thread 1' at address 0x21F7. The main pane shows the source code for this thread, with line 220 highlighted: `WaitRet = WaitForMultipleObjects(numThreads`. The code includes a `while` loop for waiting on multiple objects, a `for` loop for cleaning up threads, and a `static void Paint` function.

Address	Line	Source
0x21F7	205	SendMessage (pThreadParams [i] .hWndProgressB...
0x2211	206	SendMessage (pThreadParams [i] .hWndProgressB...
	207	
0x2224	208	pThreadHandles [i] = CreateThread (NULL, 0, (...
	209)
	210	
	211	DWORD WaitRet;
	212	MSG msg;
	213	
	214	do {
	215	while (PeekMessage (&msg, hWnd, 0, 0, PM_RE...
	216	{
	217	TranslateMessage (&msg);
	218	DispatchMessage (&msg);
	219	}
0x229D	220	WaitRet = WaitForMultipleObjects (numThread...
0x22AD	221) while (WaitRet == WAIT_TIMEOUT);
	222	
0x22B4	223	for (i=0; i<numThreads; i++)
	224	{
	225	CloseHandle (pThreadHandles [i]);
	226	DestroyWindow (pThreadParams [i] .hWndProgre...
	227	}
	228	
	229	free (pThreadHandles);
	230	free (pThreadParams);
	231	}
	232	
	233	static
	234	void Paint (HWND hWnd)
	235	{

Grouping & Filtering

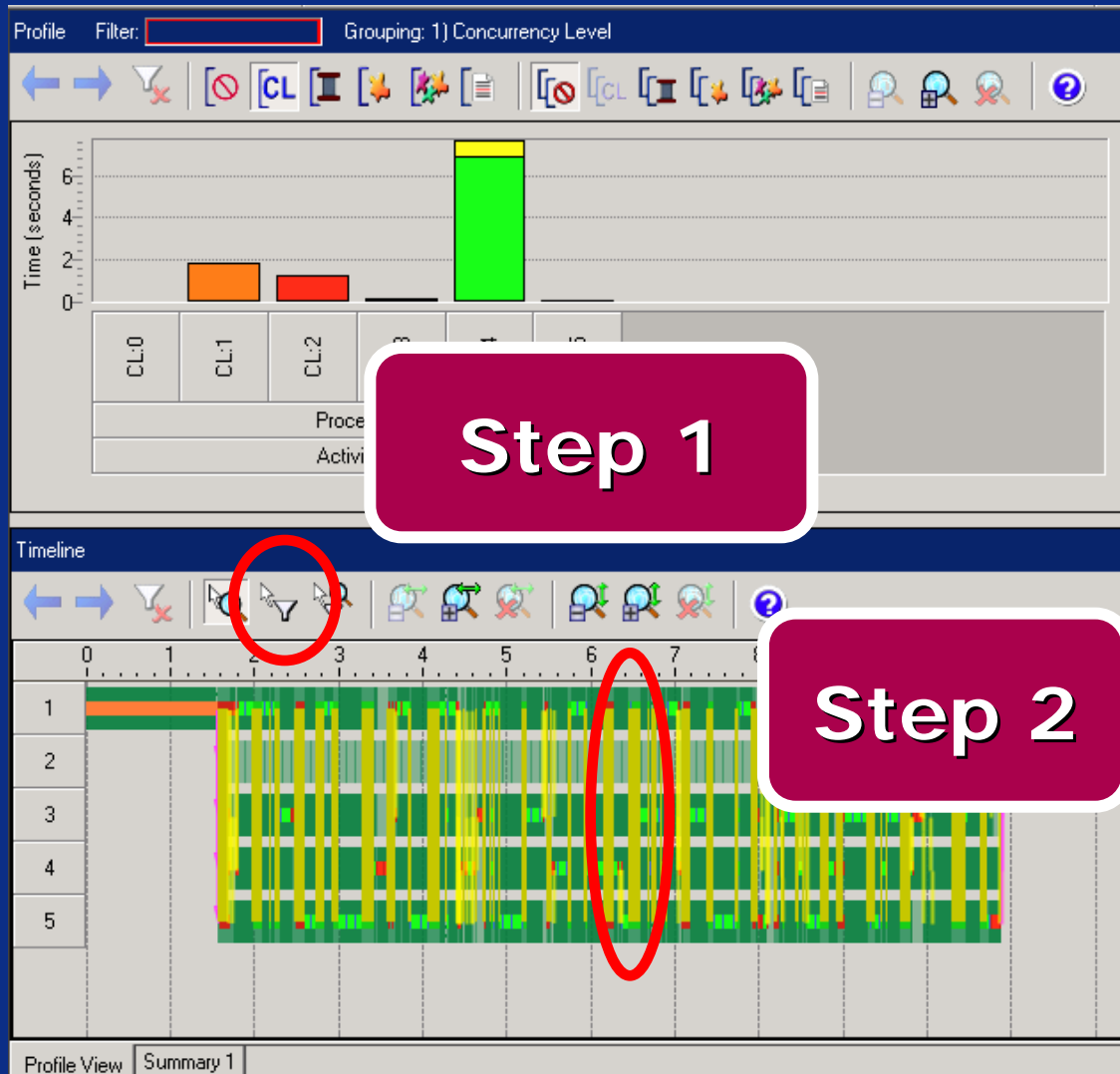
Grouping

- Allows the data to be grouped by a specific “item”
- i.e., concurrency, threads, etc

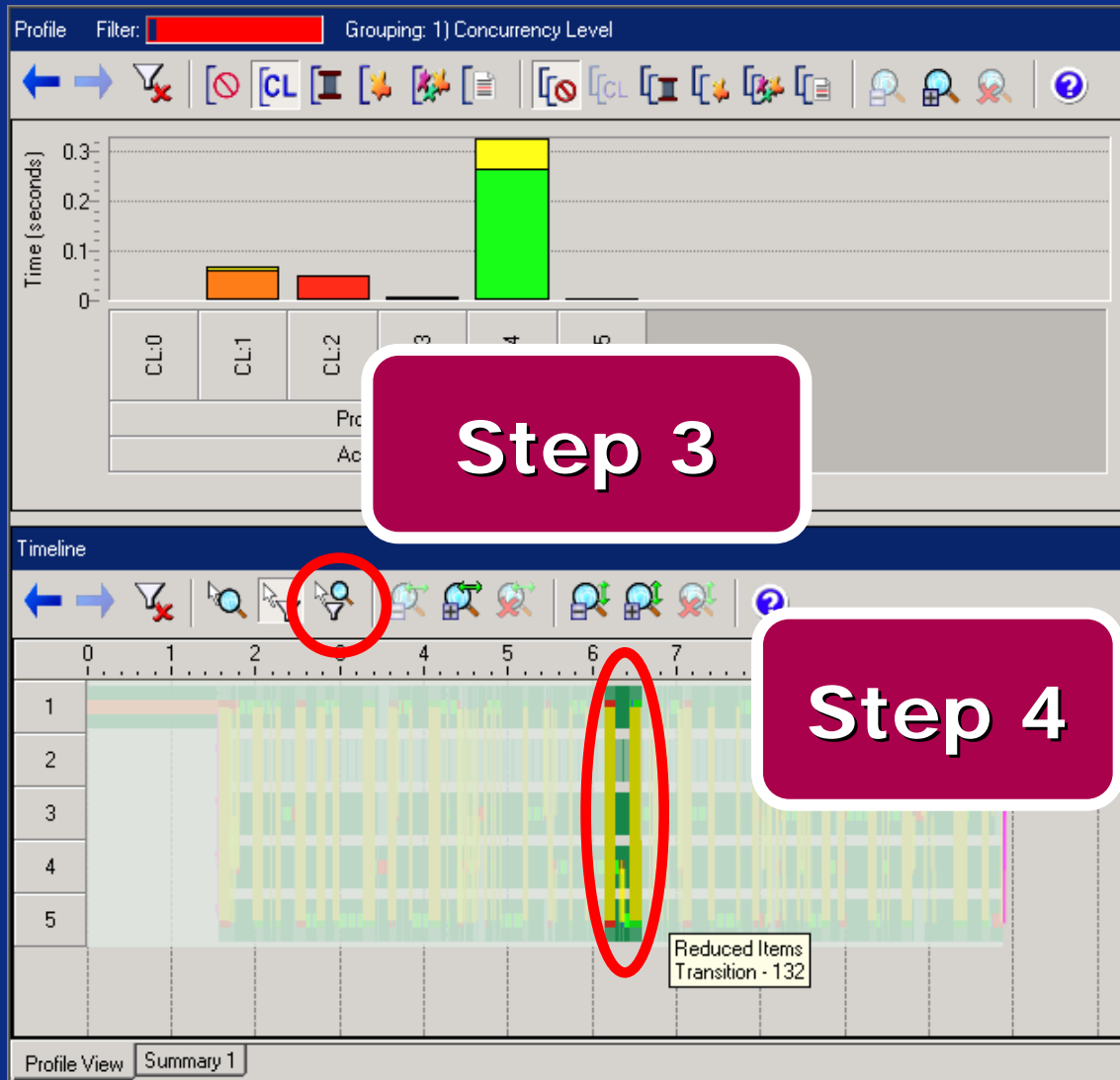
Filtering

- Allows one to filter in or filter out a particular “item”
- i.e., concurrency, object, thread, etc

Filter over Time (1)



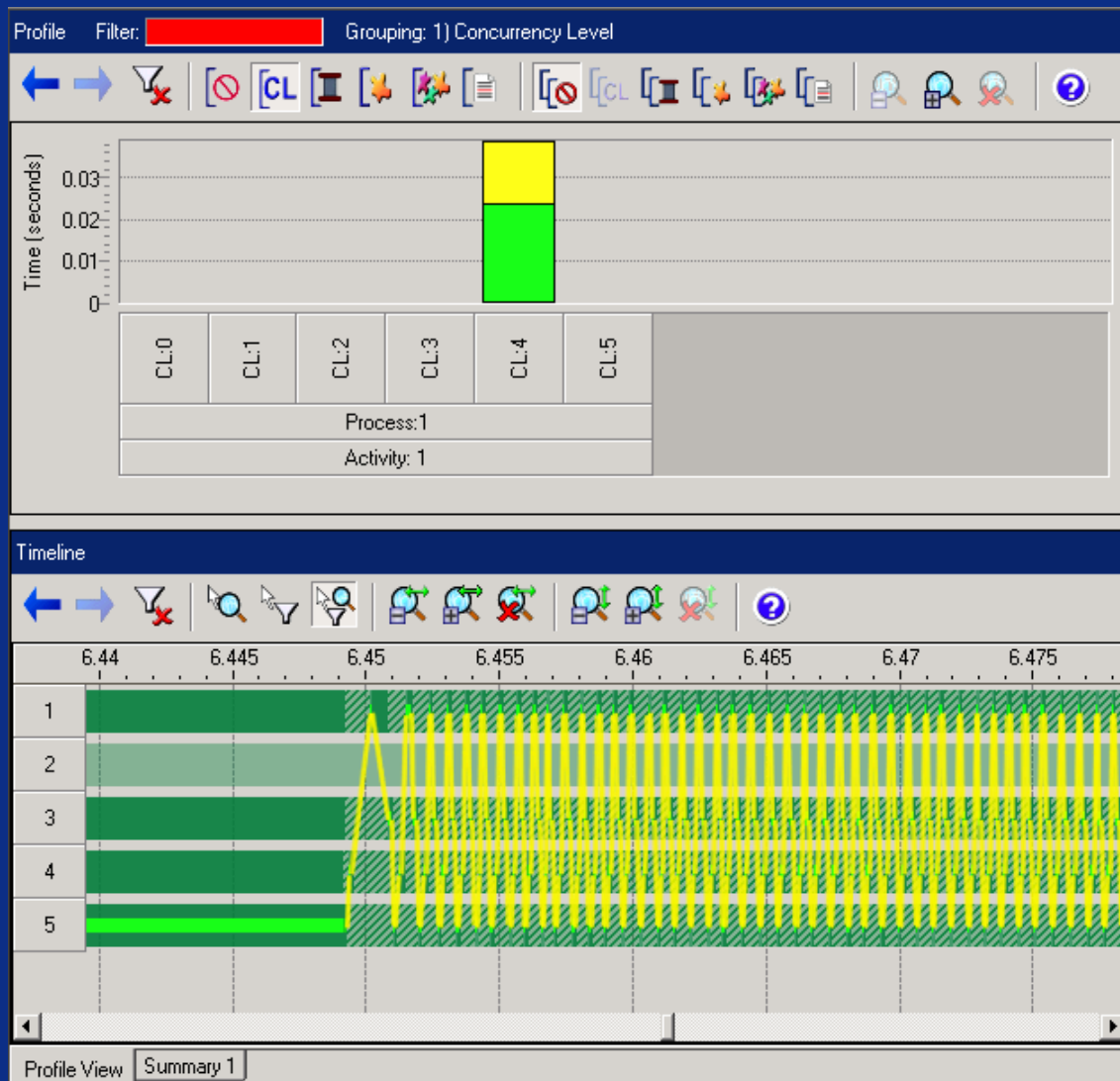
Filter over Time (2)



Step 3

Step 4

Filter over Time (3)



Dealing with User Synchronization

Many ISVs write their own primitives

Thread Checker or Thread Profiler have no knowledge of them

These tools provides an API for to instrument user synchronization.

```
__i t t _n o t i f y _s y n c _p r e p a r e ( & s p i n );  
  
w h i l e ( w a i t f o r s p i n ) {  
    i f ( t i m e o u t ) {  
        __i t t _n o t i f y _s y n c _c a n c e l ( & s p i n );  
        r e t u r n ;  
    }  
}  
__i t t _n o t i f y _s y n c _a c q u i r e d ( & s p i n );  
d o s t u f f ;  
  
__i t t _n o t i f y _s y n c _r e l e a s i n g ( & s p i n );  
r e l e a s e s p i n ;
```

Adding User Events

Many users need to flag specific events in the application

Thread Profiler allows user code to add events that can be displayed in the GUI

Thread Profiler API for user events

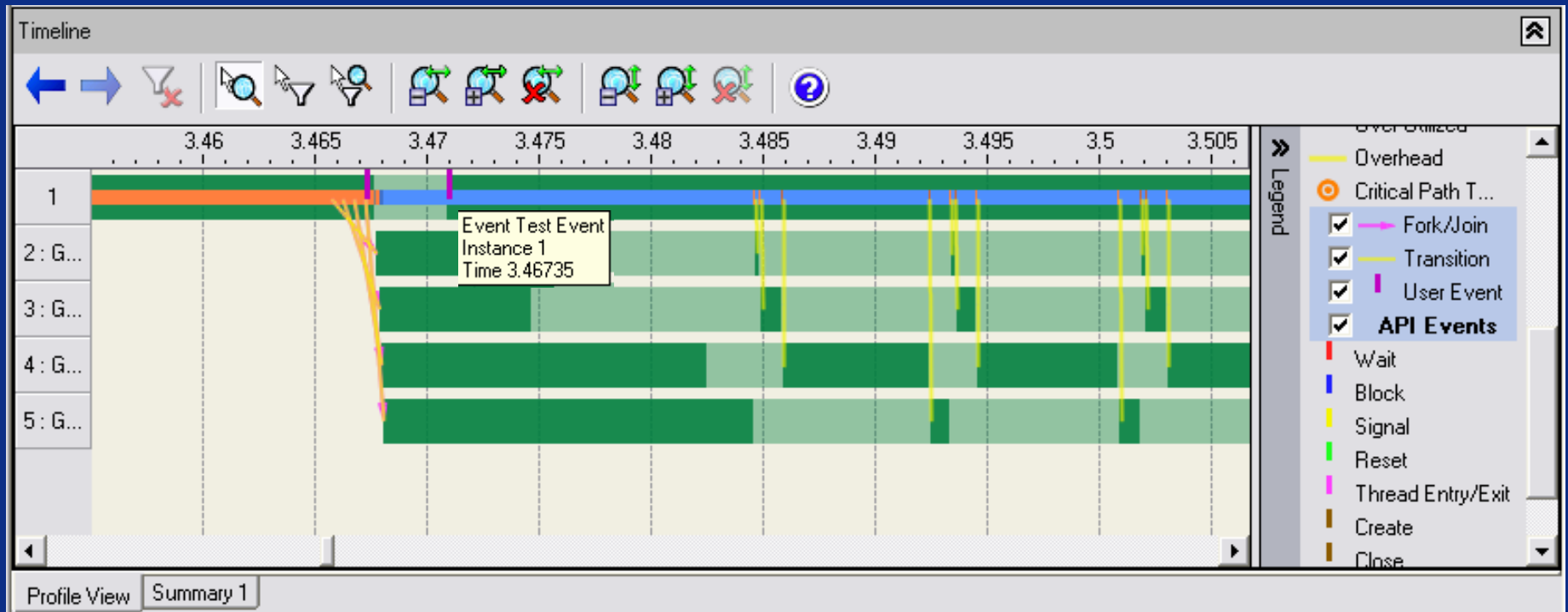
```
#include "libittnotify.h"
...
__itt_event myEvent = __itt_event_create(
    "My condition Event", 18 /* String length */ );

while( myWork ) {
    ...

    if( specificCondition ) {
        __itt_event_start( myEvent );
    }

    // optionally, you can have an end-event
}
...
```

User Events



Summary : Thread Profiler

Identifying performance issues can be time consuming without tools

Tools are required to understand and to optimize parallel efficiency and hardware utilization

Thread Profiler helps you understand your applications' thread activity, system utilization and scaling performance



Multi-core Software Development: Encouraging an Industry Transition

Motivation

Methodology

Tools

Impact

Conclusion



The Future is Now

- Multi-core processors are widely available
- The question is what use will people make of them?
- Existing models are adequate but not sufficient
 - Developers must focus on the specification of concurrent actions
 - Consider the number of cores as a dynamic quantity
- The research in transactional memory is exciting
 - Hold promise of robust parallel computing
 - Needs collaboration between hardware and software communities
- Ultimately we will reach a stage where thinking of an algorithm serially will seem awkward and over constraining

Encouraging an Industry Transition

- Education is crucial to this transition
 - Multi-core software development is still considered a “special topics course”
- Our action plan
 - Engage with the software developers
 - Manage complexity with abstractions
 - Deliver expertise with tools



Command Line (Windows and Linux)

```
C:\Program Files\Intel\VTune\tcheck\Samples\Primes>tcheck_cl Primes.exe
Intel(R) Thread Checker 3.0 command line instrumentation driver
Copyright (c) 2006 Intel Corporation. All rights reserved.
```

```
Running: C:\Program Files\Intel\VTune\tcheck\Samples\Primes\Primes.exe
```

```
Determining primes from 1 - 1000
Found 168 primes
```

```
Application finished
```

ID	Short Description	Severity	Count	Context [Best]	Description	1st Access [Best]	2nd Access [Best]
1	Write -> Read data-race	Error	12	"Primes.cpp":9 :30	Memory read at "Primes.cpp":43 conflicts with a prior memory write at "Primes.cpp":44 (flow dependence)	"Primes.cpp":44	"Primes.cpp":43
2	Write -> Read data-race	Error	12	"Primes.cpp":9 :30	Memory read at "Primes.cpp":44 conflicts with a prior memory write at "Primes.cpp":44 (flow dependence)	"Primes.cpp":44	"Primes.cpp":44
3	Thread termination	Information	1	Whole Program:5	Thread termination at "Primes.cpp":51 - includes stack allocation of 1 MB and use of 8 KB	"Primes.cpp":51	"Primes.cpp":51