

Ramesh Peri

Principal Engineer & Engineering Manager,
Performance and Threading Tools Lab
Intel® Corporation, Austin, TX 78738



Multi-Core Processors – Are they Here Yet ?

- My shopping basket at Fry's electronics on BlackFriday

| Item | Cost |
|-------------------------------------|------------|
| Motherboard+Intel® Quad core 2.4Ghz | 200 |
| 4GB Memory | 70 |
| 0.5TB Disk | 80 |
| Case | 10 |
| Graphics | 10 |
| CD/DVD | 10 |
| Total | 380 |



Outline

- Parallelism in Intel® processors
- Intel® tools for parallel programming
- Some thoughts



A New Era...

THE OLD

**Performance
Equals Frequency**

Unconstrained Power

Voltage Scaling

THE NEW

**Performance
Equals IPC**

Multi-Core

Power Efficiency

**Microarchitecture
Advancements**



Performance Comes From Parallelism

Intel® Wide Dynamic Execution

EACH CORE

*EFFICIENT
14 STAGE
PIPELINE*

*DEEPER
BUFFERS*

*4 WIDE -
DECODE TO
EXECUTE*

*4 WIDE -
MICRO-OP
EXECUTE*

*MICRO
and
MACRO
FUSION*

*ENHANCED
ALUs*

CORE 1

INSTRUCTION FETCH
AND PRE-DECODE

INSTRUCTION QUEUE

DECODE

RENAME / ALLOC

RETIREMENT UNIT
(REORDER BUFFER)

SCHEDULERS

EXECUTE

CORE 2

INSTRUCTION FETCH
AND PRE-DECODE

INSTRUCTION QUEUE

DECODE

RENAME / ALLOC

RETIREMENT UNIT
(REORDER BUFFER)

SCHEDULERS

EXECUTE

Perf ↑

Energy ↓

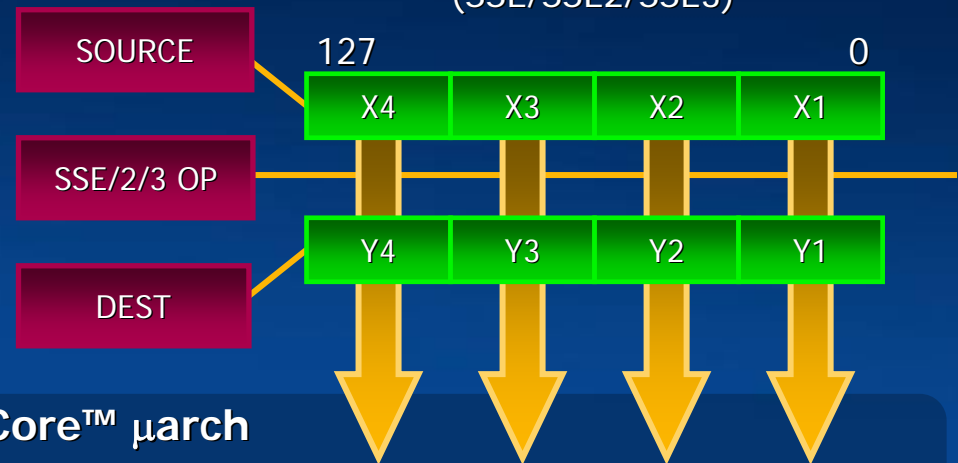
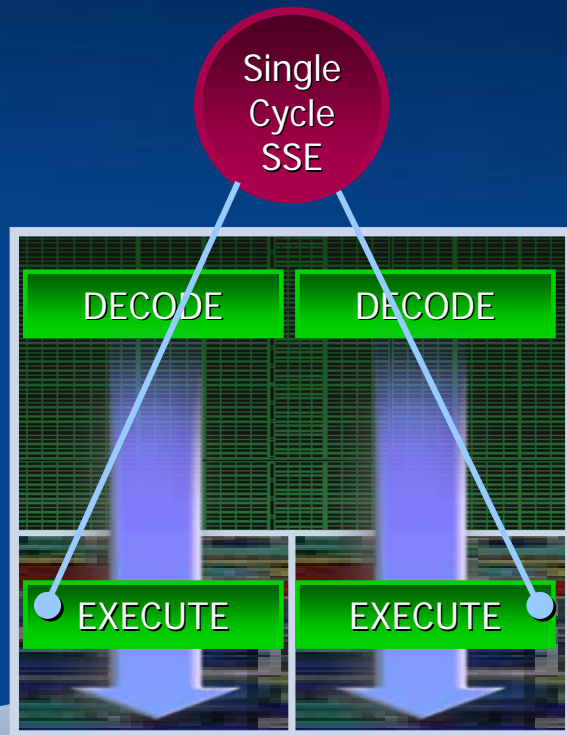
Multiple cores. Multiple instruction issue.

Intel® Advanced Digital Media Boost

Single Cycle SSE

In Each Core

SSE Operation
(SSE/SSE2/SSE3)



Core™ μ arch

CLOCK
CYCLE 1

X4opY4 X3opY3 X2opY2 X1opY1

Previous

CLOCK
CYCLE 1

X2opY2 X1opY1

CLOCK
CYCLE 2

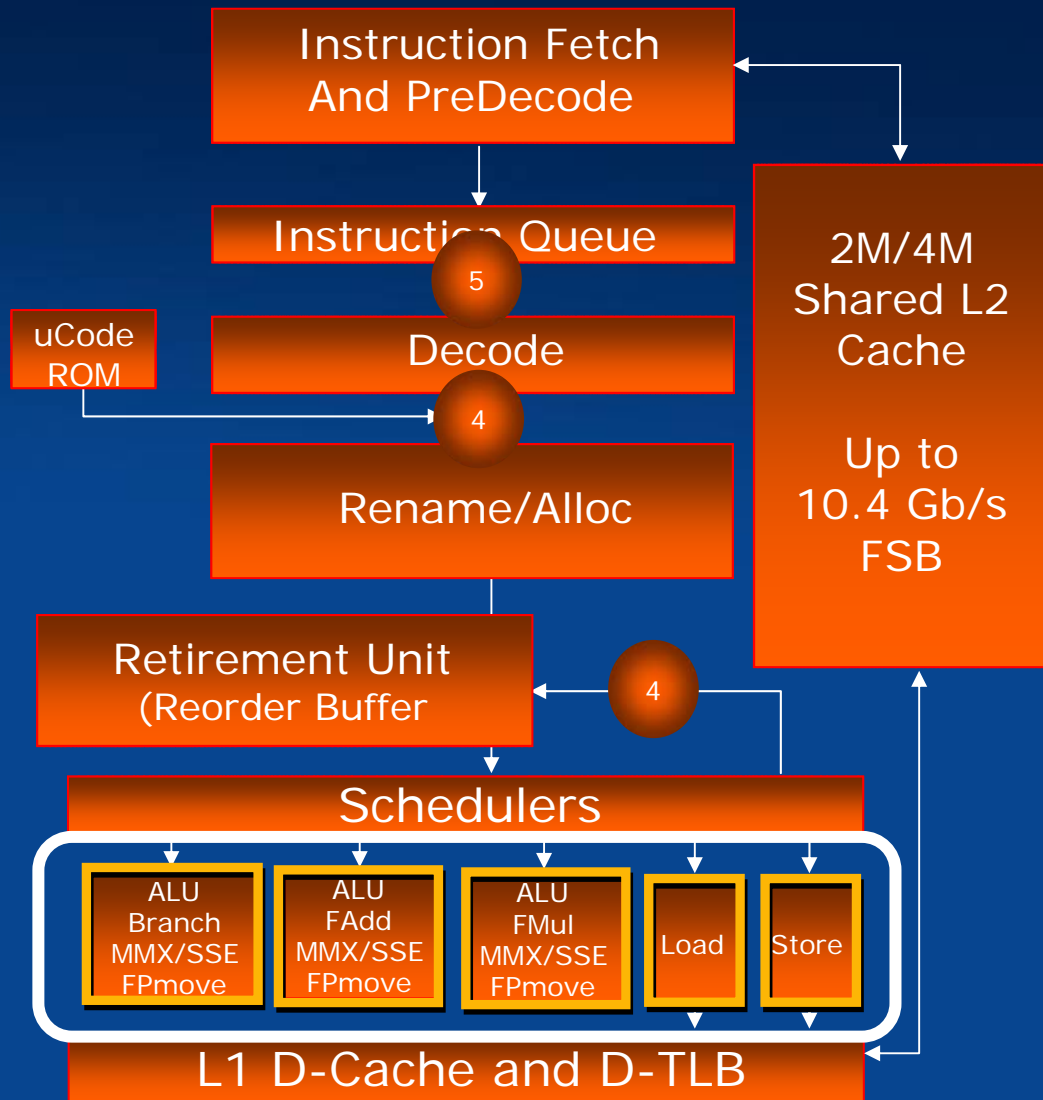
X4opY4 X3opY3

Perf ↑

Energy ↓

SIMD instructions compute multiple operations per instruction

*Graphics not representative of actual die photo or relative size



Advanced Digital Media Boost

128-bit Packed Multiply

plus

128-bit Packed Add

plus

128-bit packed Load

plus

128-bit packed Store

plus

Compare and jump

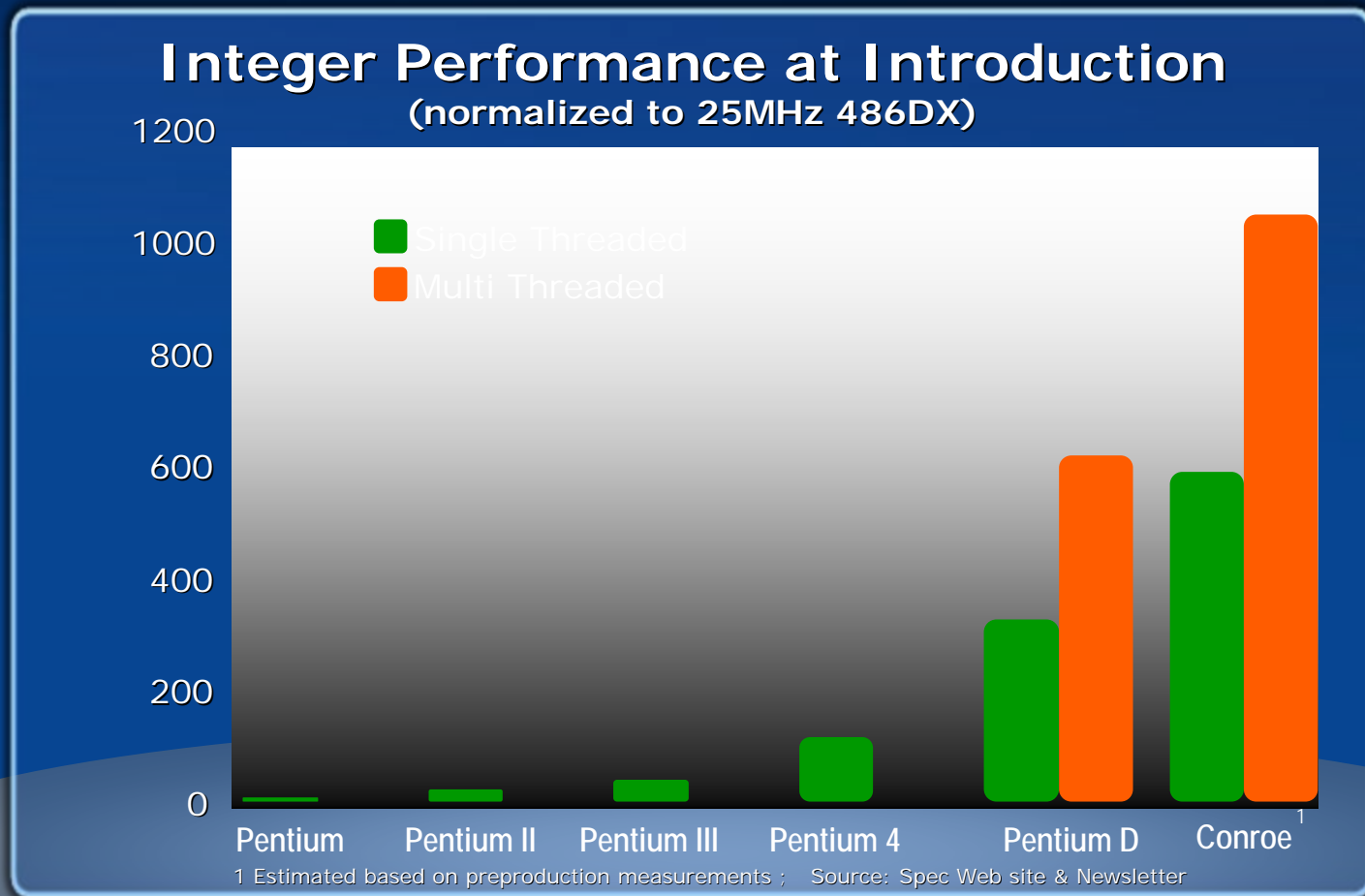
*SIMD Operations Combined
with Multiple Instruction Issue*

Many Levels of Parallelism

- Instruction-level parallelism
- SIMD (vector) parallelism
- Multi-core parallelism



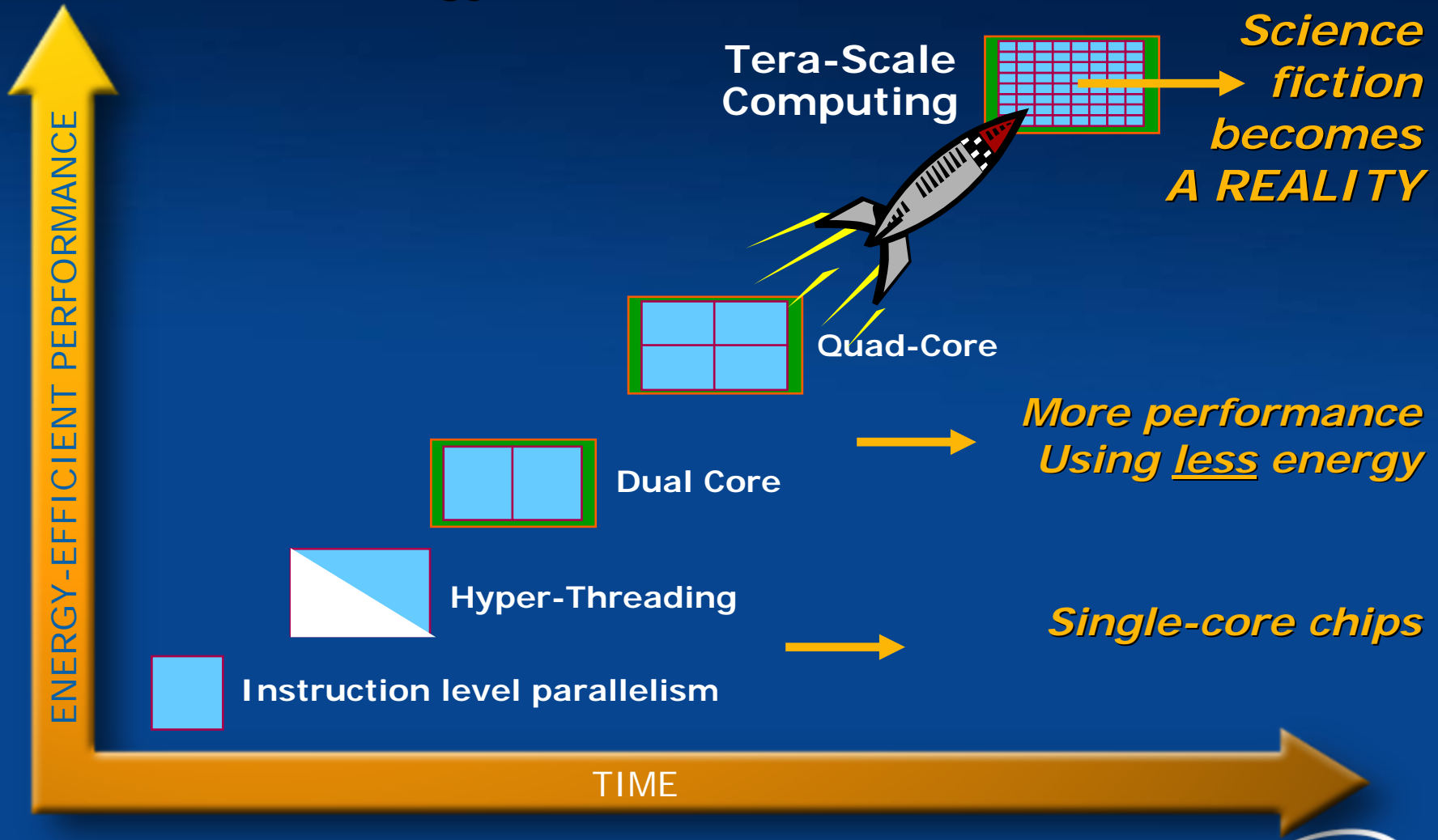
Biggest Performance Leap Since Out-of-Order Execution



*Parallel Processor: Instruction-level Parallelism,
SIMD Parallelism, Multi-core Parallelism*

Tera-Leap to Parallelism:

Energy Efficient Performance



Many core requires a high degree of scaling in an application.



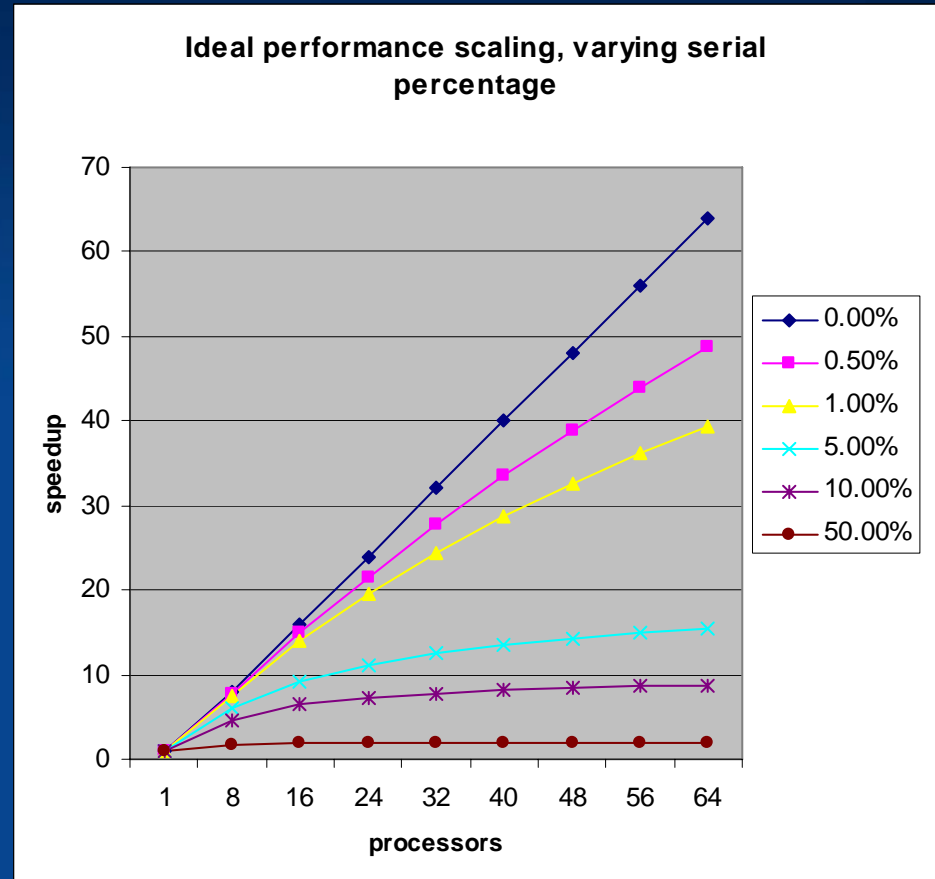
Amdahl's Law

Runtime = serial + parallel

With perfect scaling:

Runtime = serial + parallel/P

To get 50x speedup on 65 processors, serial time must be less than 0.5%



Emerging class of highly parallel workloads: games, media processing, personal data search, and more

Application Example

Sports Video Indexing



Recognize players, objects and events

Mine the video for target scenes

Synthesize a summary of what happened

Outline

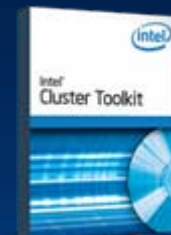
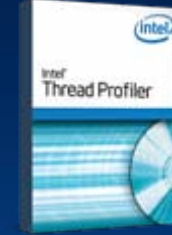
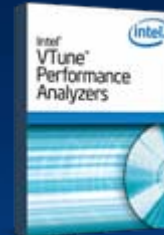
- Parallelism in Intel® processors
- Intel® tools for parallel programming
- Some thoughts



Parallel Programming Deserves Great Tools

Visualization of applications and the system

Architectural Analysis



Highly optimizing compilers delivering scalable solutions

Introduce Parallelism



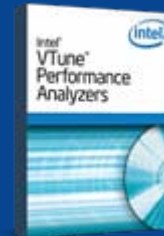
Detect latent Programming bugs

Confidence / Correctness



Tune for performance and scalability

Optimize / Tune



Example: Prime Number Generation

Outer loop
steps through
numbers that
may be prime

Inner loop
tests factors
of **number**

```
#include <stdio.h>
const long N = 18;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    for (long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```



Example: Prime Number Generation

```
#include <stdio.h>
const long N = 18;
long primes[N], number_of_primes = 0;
main()
{
    printf( "Determining primes from 1-%d \n", N );
    primes[ number_of_primes++ ] = 2; // special case
    for (long number = 3; number <= N; number += 2 )
    {
        long factor = 3;
        while ( number % factor ) factor += 2;
        if ( factor == number )
            primes[ number_of_primes++ ] = number;
    }
    printf( "Found %d primes\n", number_of_primes );
}
```

2

3 3

5 3,5

7 3,5,7

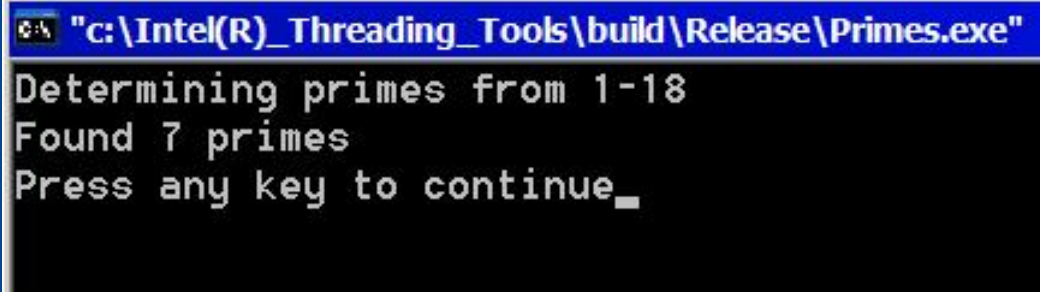
9 3

11 3,5,7,9,11

13 3,5,7,9,11,13

15 3

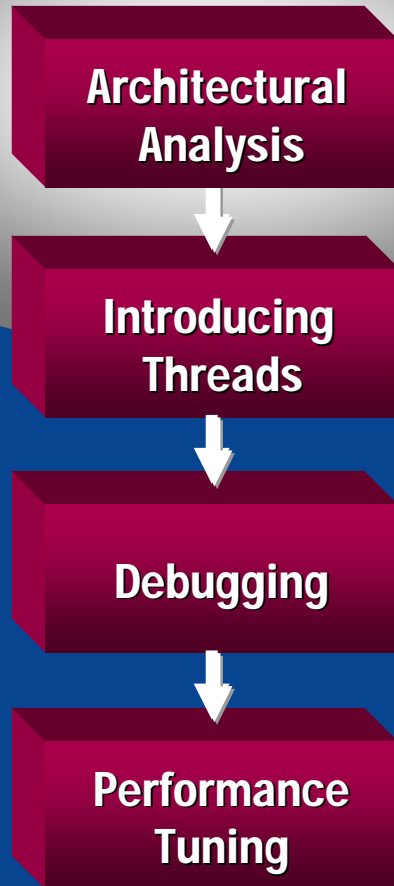
17 3,5,7,9,11,13,15,17



The screenshot shows a command prompt window with the title bar "c:\Intel(R)_Threading_Tools\build\Release\Primes.exe". The window contains the following text: "Determining primes from 1-18", "Found 7 primes", and "Press any key to continue_".

Using the Intel Programming Tools

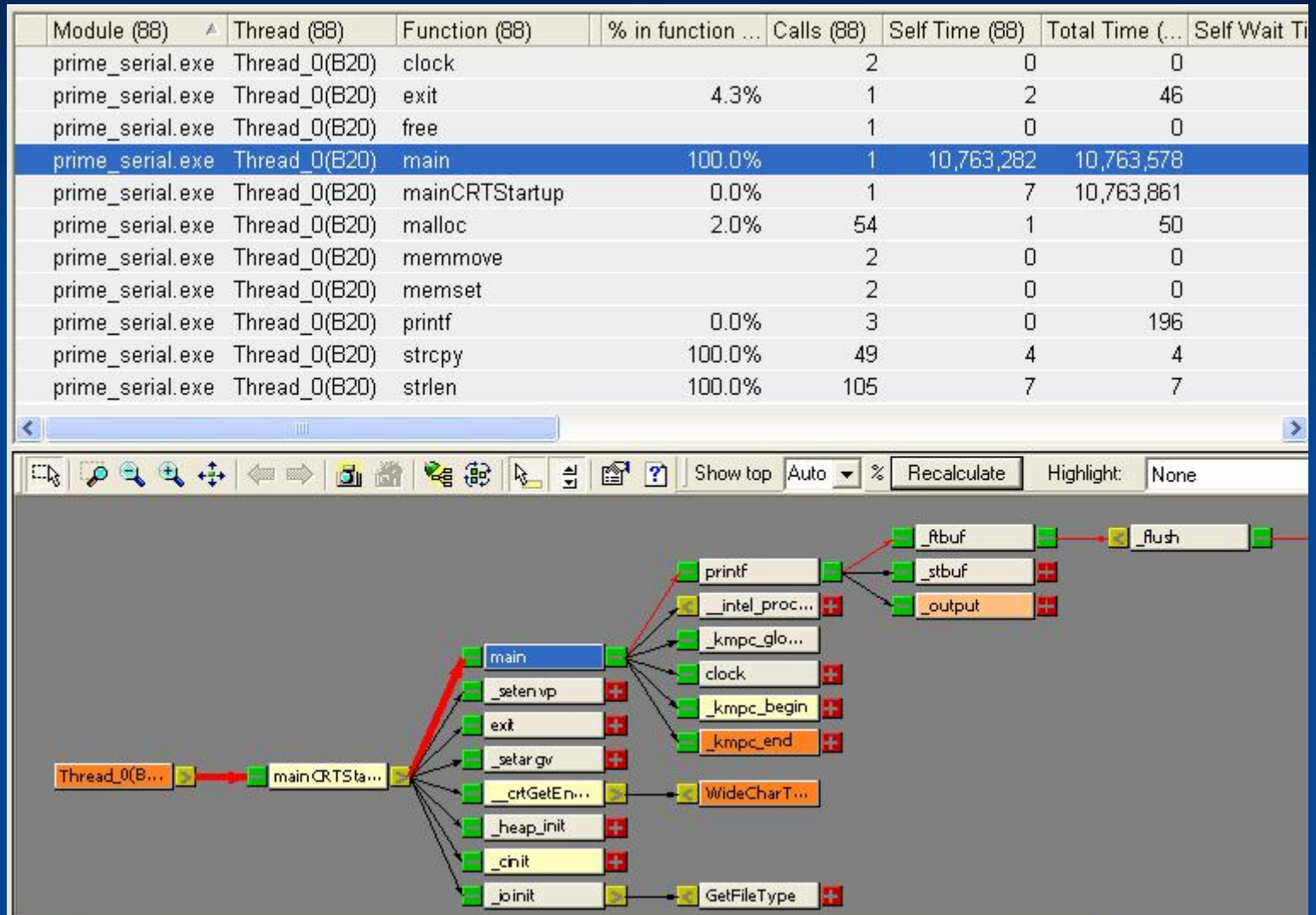
Intel® VTune™ Performance Analyzer



- Call Graph
 - Functional Structure
 - Execution Times
 - Counts



Vtune™ Call Graph Profile



Vtune™ Sampling Profile

| | | | | | | |
|---|--|--|--|--|--|--|
| <div><div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div></div> | | | | | | |
|---|--|--|--|--|--|--|

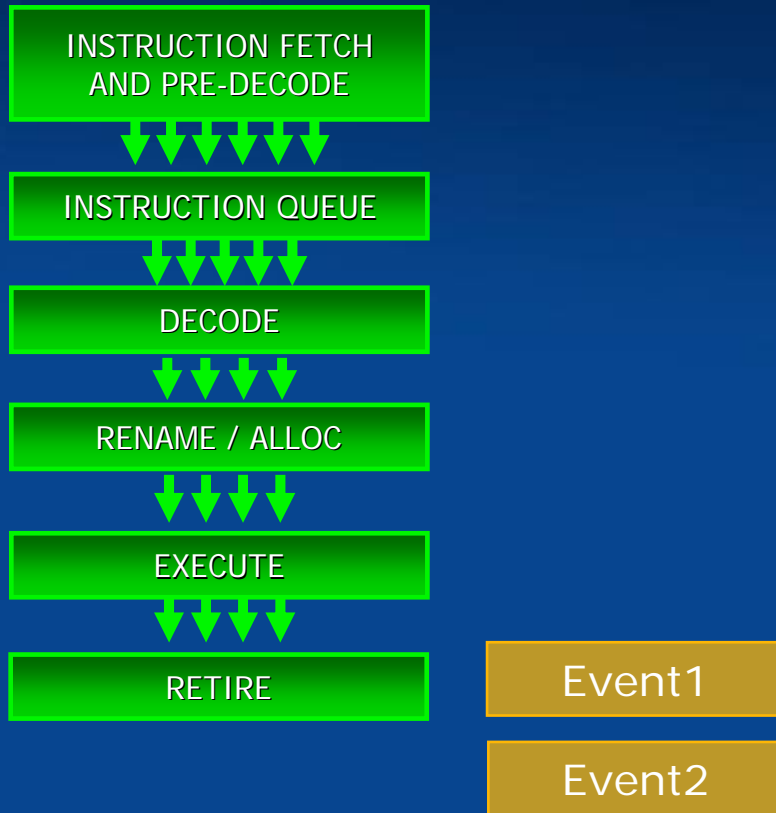
Sampled Cycles: 10,774

Sampled Instructions: 1,608

CPI: 6.7



Hardware Support for Sampling



Event counters count events that occur in pipeline.

Key events:

- Clock_cycle
- Instruction_retired

Counters count down to zero.

- At zero, interrupt and capture instruction pointer.
- Reset counter and continue execution.

Statistically associate events with instructions

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 = 3; number <= N; number += 2 )
{
    long factor = 3;
    while ( number % factor ) factor += 2;
    if ( factor == number )
        primes[ number_of_primes++ ] = number;
}
```

Using the Intel Programming Tools

Architectural
Analysis



Introducing
Threads



Debugging



Performance
Tuning



Intel® Compilers

- OpenMP Loop Construct
 - Creates one thread per core
 - Assigns iterations to threads



Example: Add OpenMP Directive

```
• #include <stdio.h>
• const long N = 100000;
• long primes[N], number_of_primes = 0;
• main()
• {
•     printf( "Determining primes from 1-%d \n", N );
•     primes[ number_of_primes++ ] = 2; // special case
•     #pragma omp parallel for
•     for ( long number = 3; number <= N; number += 2 )
•     {
•         long factor = 3;
•         while ( number % factor ) factor += 2;
•         if ( factor == number )
•             primes[ number_of_primes++ ] = number;
•     }
•     printf( "Found %d primes\n", number_of_primes );
• }
```

Example: Add OpenMP Directive

```
• #include <stdio.h>
• const long N = 100000;
• long primes[N], number_of_primes = 0;
• main()
• {
•     print( "N = %d\n", N );
•     prime_sieve( N );
•     #pragma omp parallel for
•     for ( long number = 3; number <= N; number += 2 )
•     {
•         long factor = 3;
•         while ( number % factor == 0 ) factor += 2;
•         if ( factor == number )
•             primes[ number_of_primes++ ] = number;
•     }
•     printf( "Found %d primes\n", number_of_primes );
• }
```

Thread 1
Iterations 1..50000

Thread 2
Iterations 50000..100000

Example: Add OpenMP Directive

```
• #include <stdio.h>
• const long N = 100000;
• long primes[N], number_of_primes = 0;
• main()
• {
•     printf( "Determining primes from 1-100000\n" );
•     primes[ number_of_primes ] = 1;
•     #pragma omp parallel for
•     for ( long number = 3; number <= N; number += 2 )
•     {
•         long factor = 3;
•         while ( number % factor != 0 )
•             factor += 1;
•         if ( factor == number )
•             primes[ number_of_primes ] = number;
•     }
•     printf( "Found %d primes\n", number_of_primes );
• }
```

Command Prompt

```
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>
```

Data Race

Thread 1

Thread 2



○
○
○

`number_of_primes++`

○
○
○

○
○
○

`number_of_primes++`

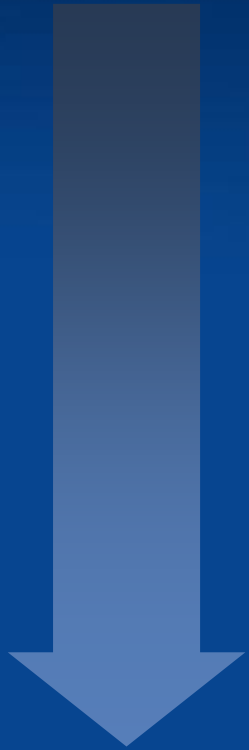
○
○
○

The updates to `number_of_primes` are not atomic!

Data Race

Thread 1

Thread 2



○
○
○

t1 = LOAD n_of_p
t1 ++
STORE n_of_p = t1

○
○
○

○
○
○

t2 = LOAD n_of_p
t2 ++
STORE n_of_p = t2

○
○
○

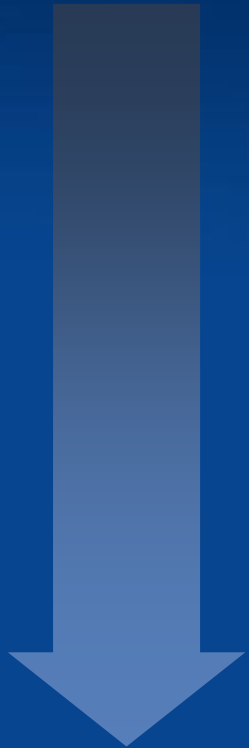
The updates to number_of_primes are not atomic!



Data Race

Thread 1

Thread 2



○
○
○

1: t1 = LOAD n_of_p

3: t1++

6: STORE n_of_p = t1

○
○
○

○
○
○

2: t2 = LOAD n_of_p

4: t2++

5: STORE n_of_p = t2

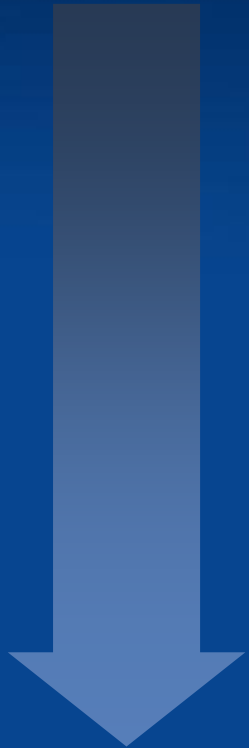
○
○
○

Lose one increment to number of primes!

Data Race

Thread 1

Thread 2



○
○
○

CRITICAL {

t1 = LOAD n_of_p

t1++

STORE n_of_p = t1

}

○
○
○

○
○
○

CRITICAL {

t2 = LOAD n_of_p

t2++

STORE n_of_p = t2

}

○
○
○

Add synchronization to remove data race

Example: Add Synchronization Directive

```
• #include <stdio.h>
• const long N = 100000;
• long primes[N], number_of_primes = 0;
• main()
• {
•     printf( "Determining prime
•     primes[ number_of_primes++
•     #pragma omp parallel for
•     for ( long number = 3; num
•     {
•         long factor = 3;
•         while ( number % factor
•         if ( factor == number )
•     #pragma omp critical
•         primes[ number_of_prim
•     }
•     printf( "Found %d primes\n", number_of_primes );
• }
```

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>_
```

Using the Intel Programming Tools

Architectural
Analysis



Introducing
Threads



Debugging



Performance
Tuning



Intel® Thread Checker

- Thread Safety Issues
 - Data Races
 - Deadlocks



Intel® Thread Checker

VTune(TM) Performance Environment - [Thread Checker - Activity: 03:17 PM, 2005 Feb 13 (TC: primes.exe)]

File Edit View Activity Configure Window Help

| ID | Severity | Count | 1st Access[Source Line] | Short Description | 2nd Access[Source Line] |
|----|----------|-------|-------------------------|--------------------------|-------------------------|
| 0 | | 9590 | "2_omp.cpp":14 | Write -> Read data-race | "2_omp.cpp":14 |
| 1 | | 9590 | "2_omp.cpp":14 | Write -> Write data-race | "2_omp.cpp":14 |
| 2 | | 9590 | "2_omp.cpp":14 | Read -> Write data-race | "2_omp.cpp":14 |
| 3 | | 1 | "2_omp.cpp":5 | Thread termination | "2_omp.cpp":5 |

Memory read of number_of_primes at "2_omp.cpp":14 conflicts with a prior memory write of number_of_primes at "2_omp.cpp":14 (flow dependence)

1st Access Stack: main "2_omp.cpp":14

Source

```

long factor = 3;
while ( number % factor ) factor += 2;
if ( factor == number )
    primes[ number_of_primes++ ] = number;
}
printf( "Found %d primes\n", number_of_primes );

```

2nd Access Stack: main "2_omp.cpp":14

Source

```

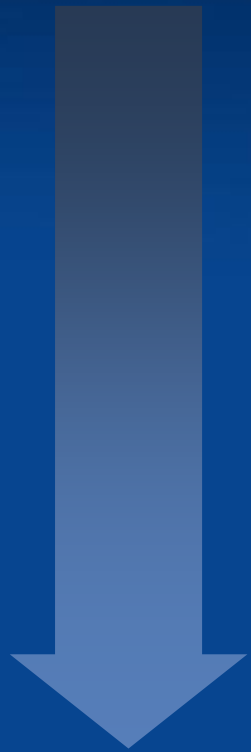
long factor = 3;
while ( number % factor ) factor += 2;
if ( factor == number )
    primes[ number_of_primes++ ] = number;
}
printf( "Found %d primes\n", number_of_primes );

```

Source View Stack Traces

For Help, press F1

How Does Thread Checker Work?



Thread 1

○
○
○

Lock(L);

n_of_p++

Unlock(L);

○
○
○

Thread 2

○
○
○

Lock(L);

n_of_p++;

Unlock(L);



How Does Thread Checker Work?

Use binary instrumentation

Thread 1

○
○
○

record lock(L)

Lock(L);

record read(n_of_p)

record write(n_of_p)

n_of_p++

record unlock(L)

Unlock(L);

○
○
○

Thread 2

○
○
○

record lock(L)

Lock(L);

record read(n_of_p)

record write(n_of_p)

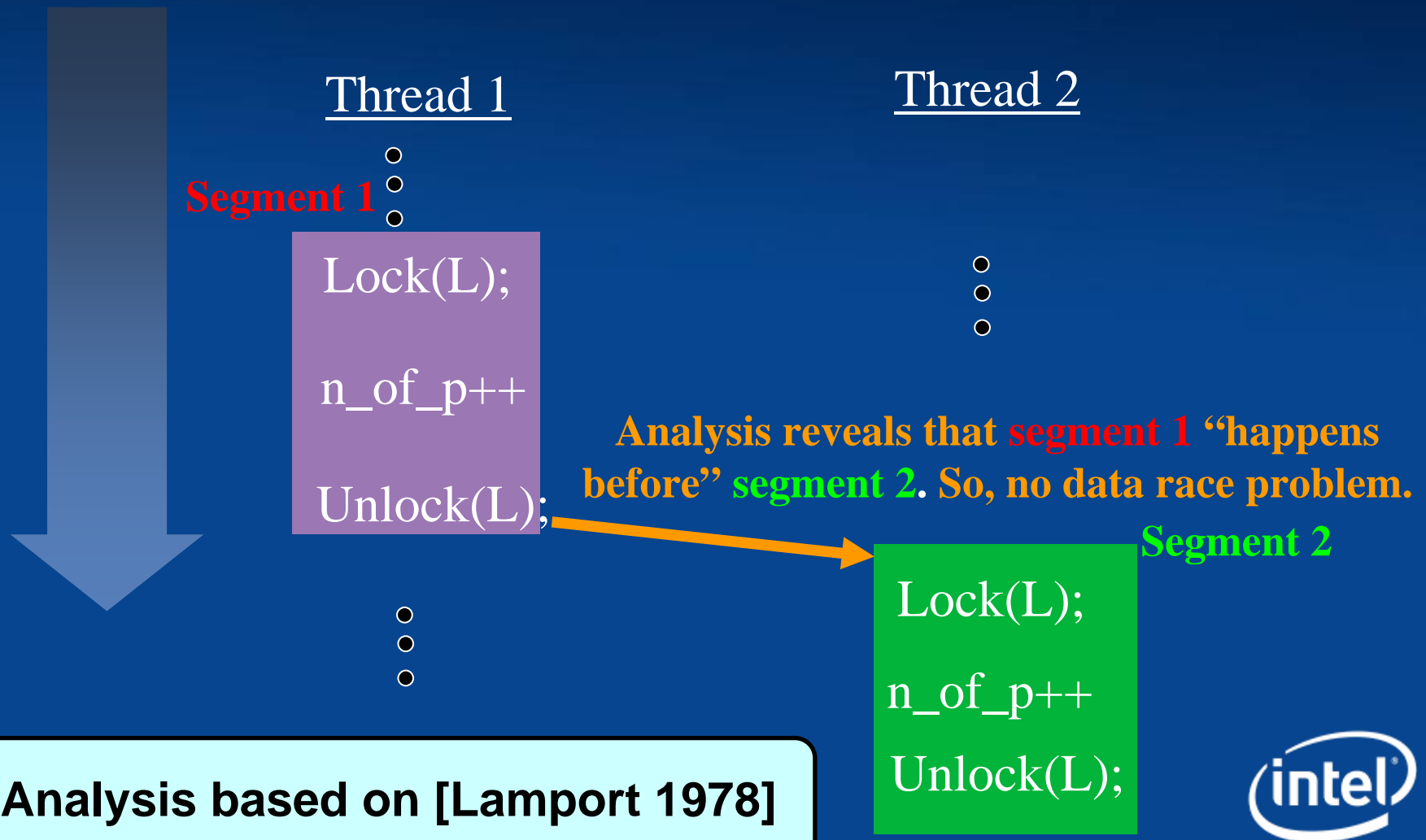
n_of_p++;

record unlock(L)

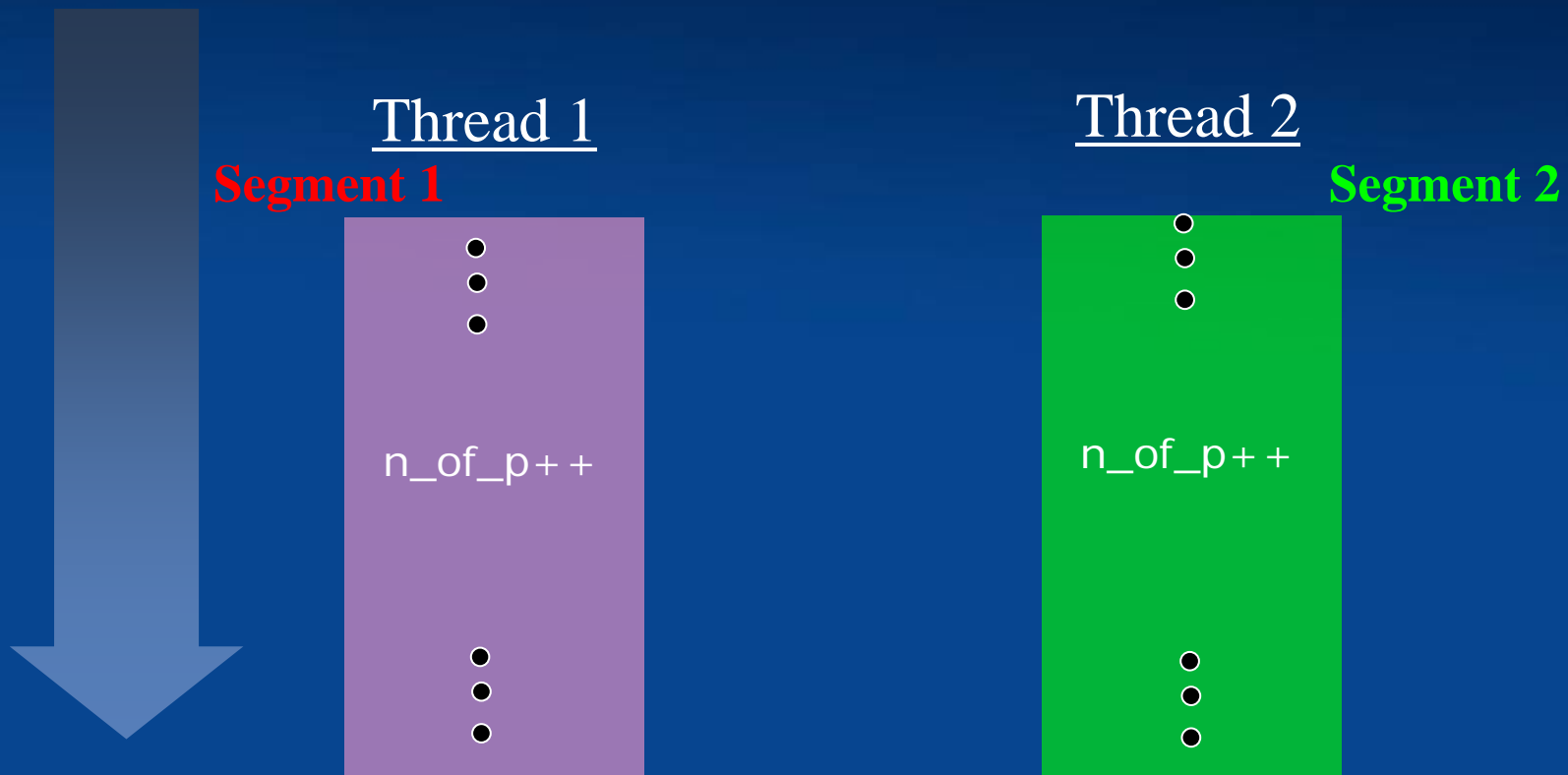
Unlock(L);



How Does Thread Checker Work?



How Does Thread Checker Work?



With no locks, neither segment “happens before” the other. So there is a data race!



Using the Intel Programming Ttools

Architectural
Analysis



Introducing
Threads



Debugging



Performance
Tuning

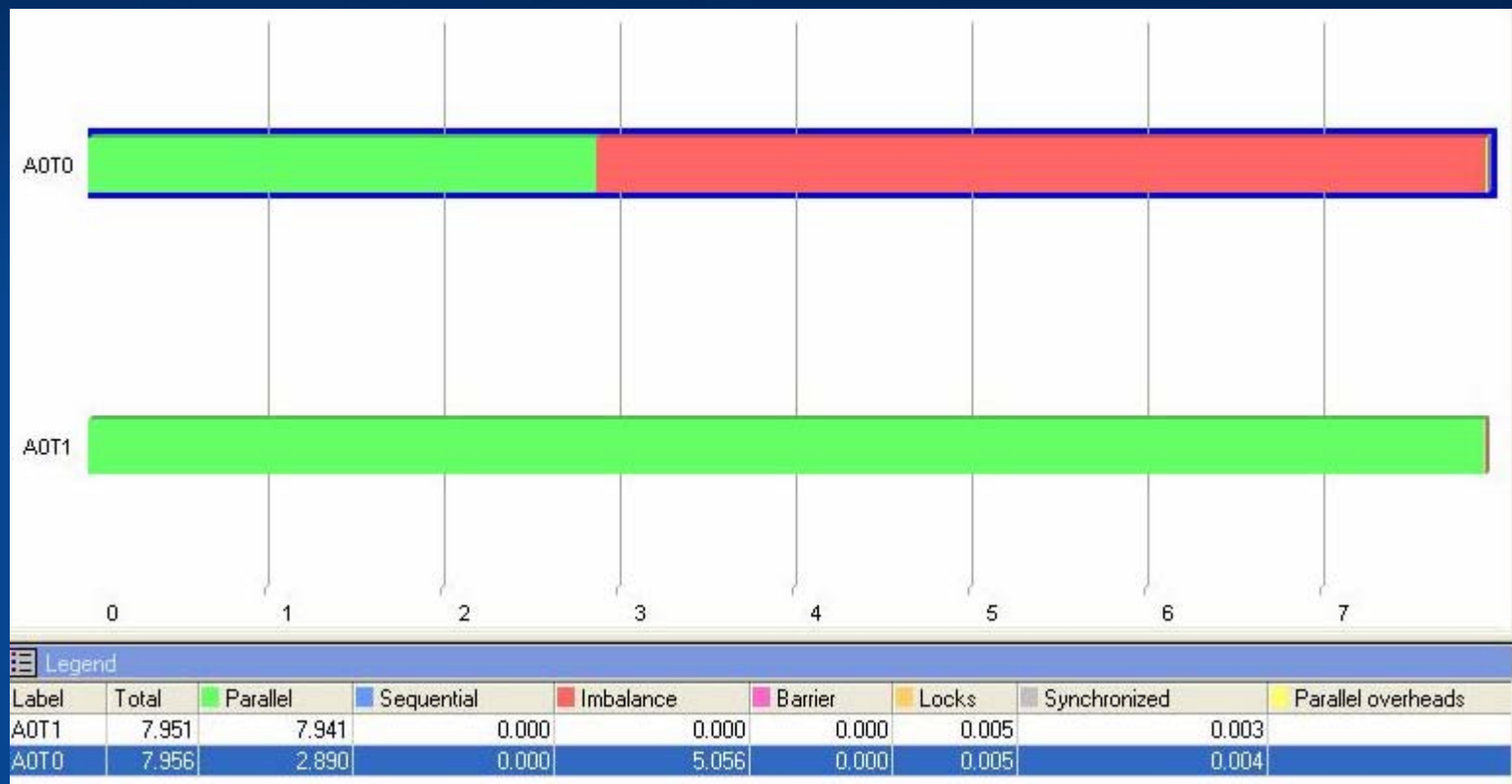


Intel® Thread Profiler

- Find Contended Locks
 - Most Overhead
 - Largest Reduction in Parallelism



Profiling the Two Threads in Primes



Work is not balanced between threads



```
for ( long number = 3; number <= N; number += 2 )  
{  
    long factor = 3;  
    while ( number % factor ) factor += 2;  
    if ( factor == number )  
        primes[ number_of_primes++ ] = number;  
}
```

Thread 1
Iterations 1..50000

Thread 2
Iterations 50000..100000

Thread 2 does 3x more work than thread 1

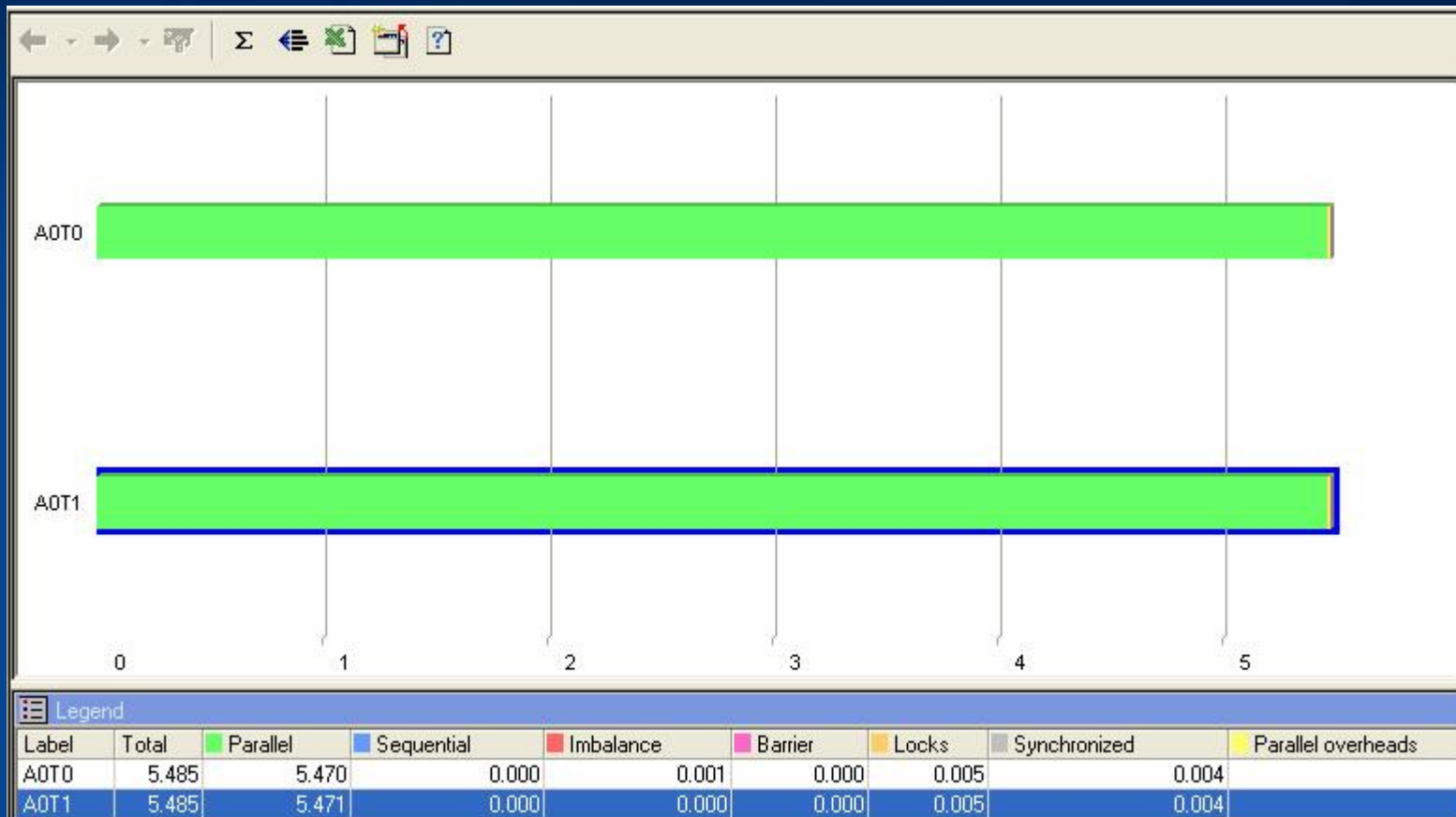


Example: Change to Dynamic Scheduling

```
• #include <stdio.h>
• const long N = 100000;
• long primes[N], number_of_primes = 0;
• main()
• {
•     printf( "Determining primes from 1-%d \n", N );
•     primes[ number_of_primes++ ] = 2; // special case
•     #pragma omp parallel for schedule(dynamic)
•     for ( long number = 3; number <= N; number += 2 )
•     {
•         long factor = 3;
•         while ( number % factor ) factor += 2;
•         if ( factor == number )
•             #pragma omp critical
•             primes[ number_of_primes++ ] = number;
•     }
•     printf( "Found %d primes\n", number_of_primes );
• }
```

Each iteration is assigned dynamically to a worker thread

Profiling the Two Threads with Dynamic Scheduling



Work balanced between the threads



Using the Intel® Programming Tools

Architectural
Analysis

Intel® VTune™ Analyzers

Introducing
Threads

Intel® C++ Compiler

Debugging

Intel® Thread Checker

Performance
Tuning

Intel® Thread Profiler

All these tools are available for free at Intel® website
for academic use



Outline

- Parallelism in Intel® processors
- Intel® tools for parallel programming
- Some thoughts



The landscape of parallel tools

- Tools need to just work out of the box
 - Provide noise less information
 - No false positives
 - Information must be relevant and accurate
- Must be scalable and work in a variety of environments
 - Language, Architecture and OS independent
 - Must work in the presence of VMMs
- Must work on existing binaries
 - No recompilation, special preparation of binaries for tools
- Performance Degradation must be in acceptable range
- Need more capable tools to handle future multi-core processors
 - Non-Uniform memory access
 - Multiple layers of memory hierarchy visible to programmer
 - Very large number of cores



Other Software development tools in our lab

- Data Mining and machine learning techniques for utilization of PMU counters to improve program performance
- Pin – a dynamic instrumentation system
 - **Computer Architecture Research**
 - For emulating new instructions
 - Modeling micro-architectural features
 - **Branch Predictor Models:**
 - Cache Models:
 - Simple Timing Models:
 - **Performance Analysis Tools**
 - Dynamic/Static Instruction Counting Tools
 - CallGraph/CallCount Tools
 - Tool to produce Annotated CFG
 - **Program Analysis Tools**
 - Code Coverage Tools
 - Dynamic Optimization Tools
 - Memory Checking Tools
 - Race Detection Tools
 - Replay Tools
- Dynamic optimization of programs
- Parallel libraries/domain specific languages



Summary

- Parallel processors
 - Instruction level parallelism
 - SIMD parallelism
 - Multi-core parallelism
- Intel® tools for parallel programming
 - Intel® Fortran/C++ compiler
 - Intel® VTune™
 - Intel® Thread Checker
 - Intel® Thread Profiler
- Parallel applications
 - New emerging class of parallel applications
 - Tune for all levels of parallelism
 - Need more research into parallel tools and methodologies



