From Serial to Parallel

Stephen Blair-Chappell
Intel Compiler Labs
www.intel.com

Agenda

- Why Parallel?
- Optimising Applications
- Steps to move from Serial to Parallel

A Message From Steve Lionel
Intel Compiler Labs

Congratulations BCS FIG.wmv

BCS Fortran Interest Group 40th Anniversary

bcs The Chartered Institute for IT
Fortran Specialist Group

Software & Services Group intel Software

Moving to Parallel – a view from some developers

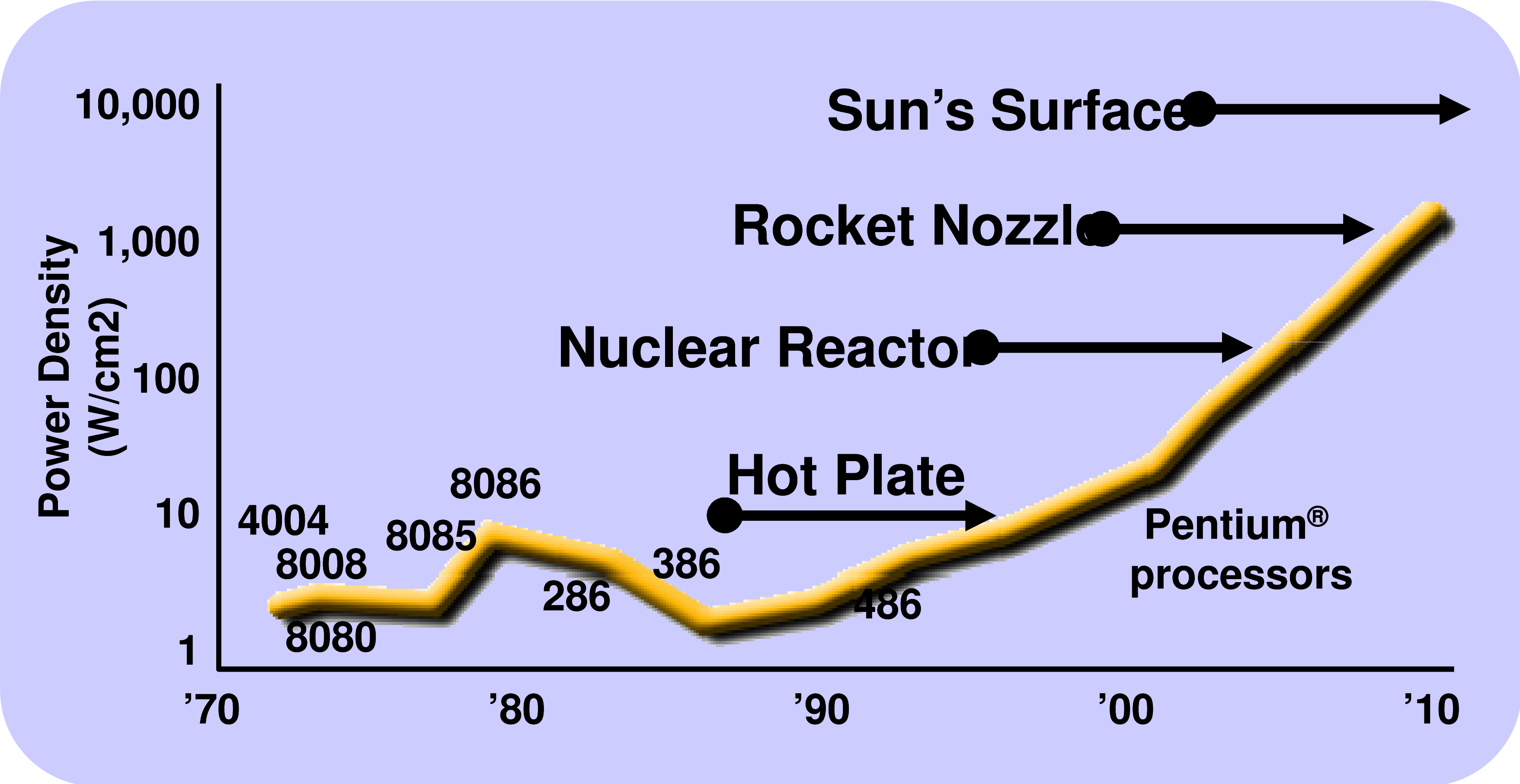
- Top 5 challenges
  - Legacy
  - Education
  - Tools
  - Fear of many cores
  - Maintainability

Why Parallel?

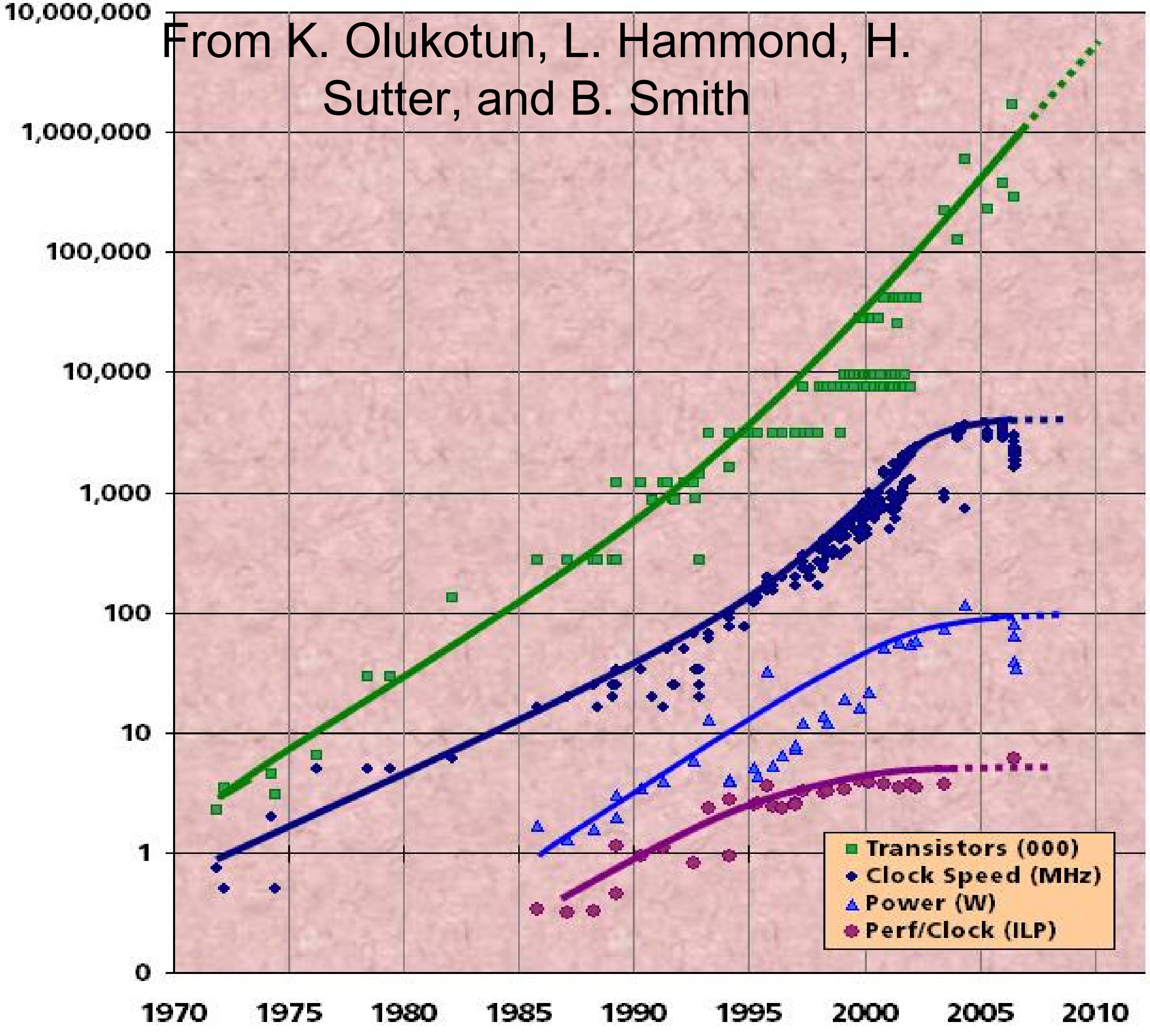
Section 1

Why is *everyone* going multi-core?

Power Density Race

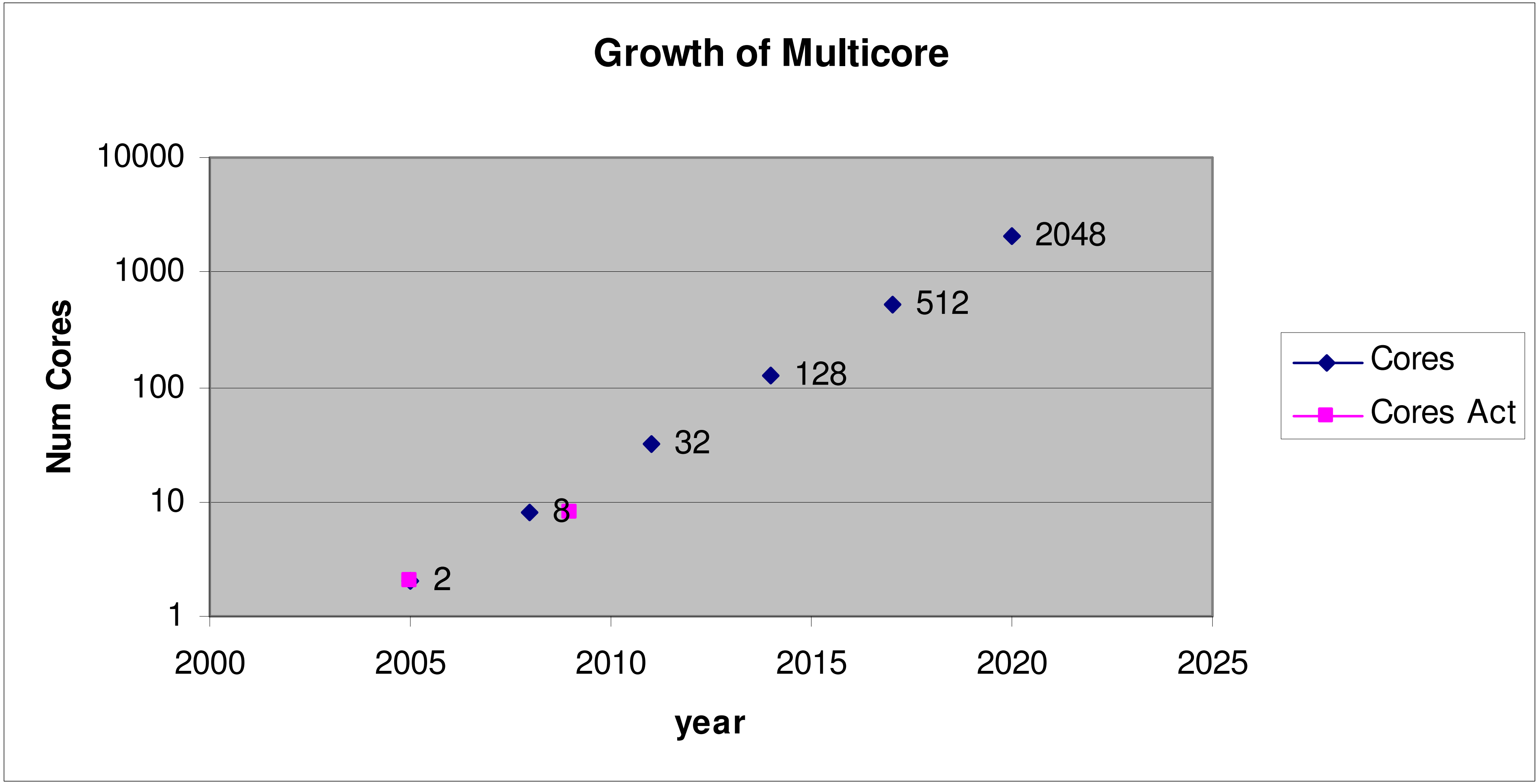


Moore’s Law reinterpreted

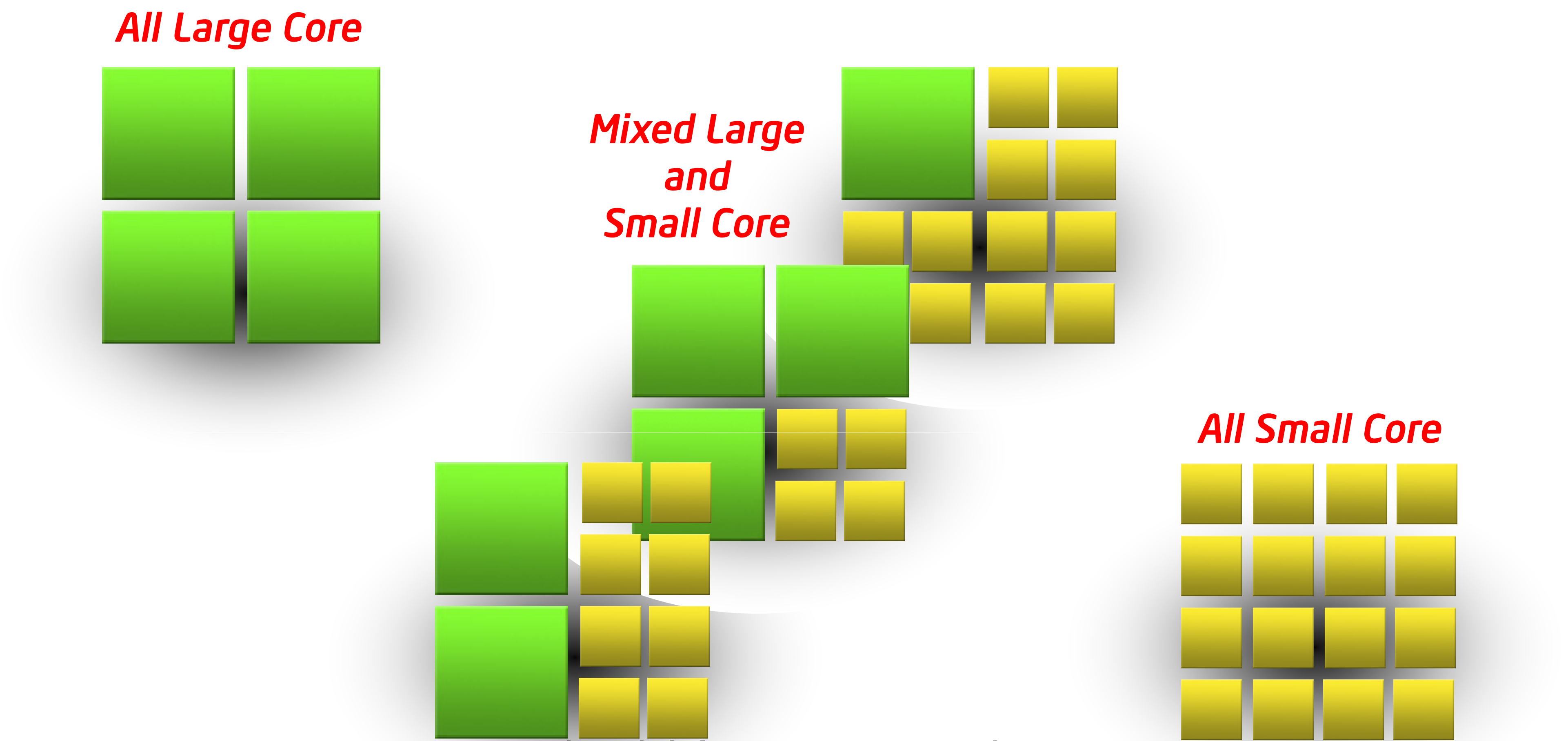


- Speed no longer increasing
- Num transistors still growing
- Num Cores rather than clock speed is doubling every 18 months

Theoretical growth of cores



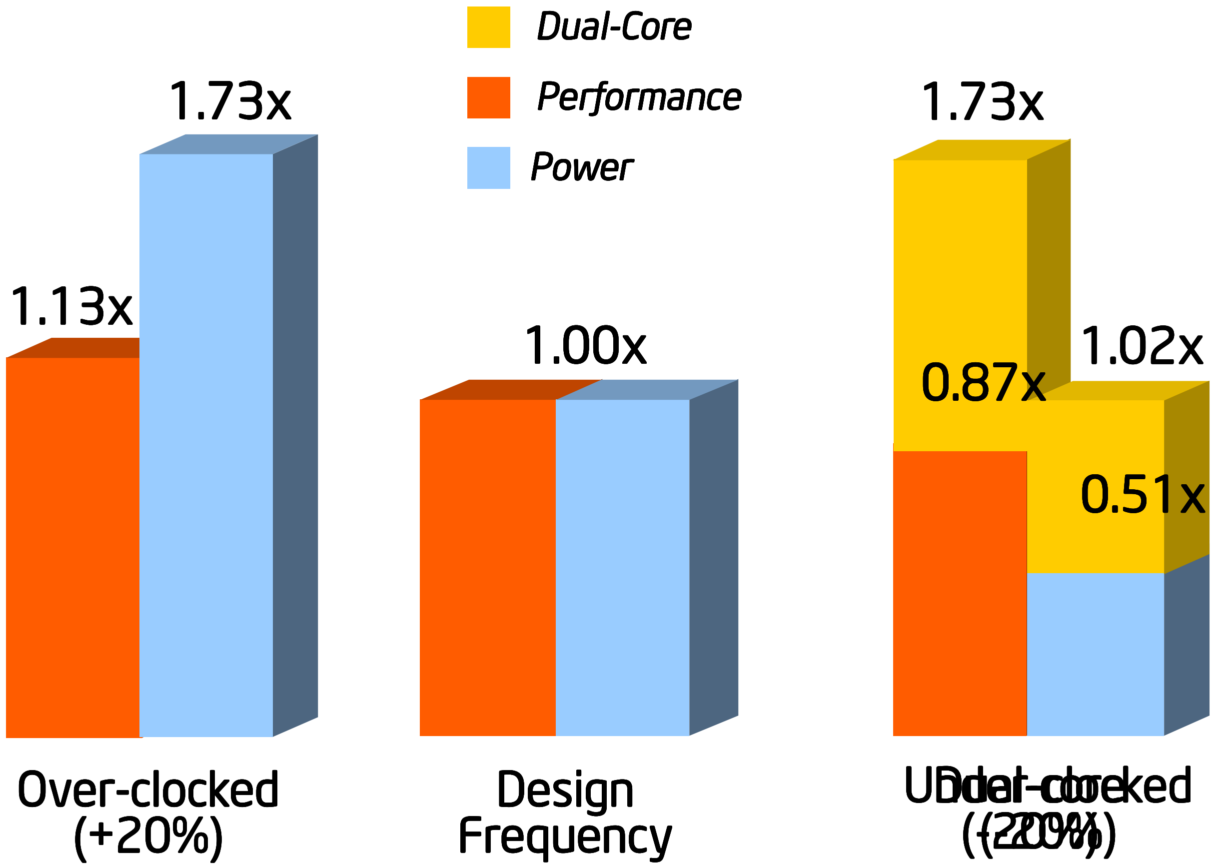
Future: Multicore and Manycore



Connections to memory bank(s), connections between processors, memory coherency models – all come into play. Diversity!

Note: the above pictures don't necessarily represent any current or future Intel products

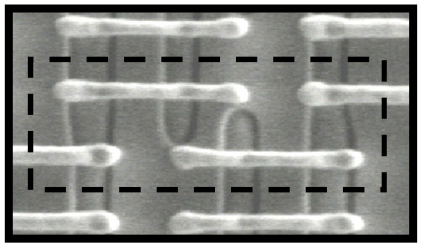
Multi-core : beating the *power\performance* barrier



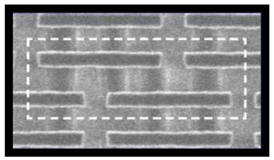
Relative single-core frequency and Vcc

Industry's First 45 nm High-K + Metal Gate Transistor Technology

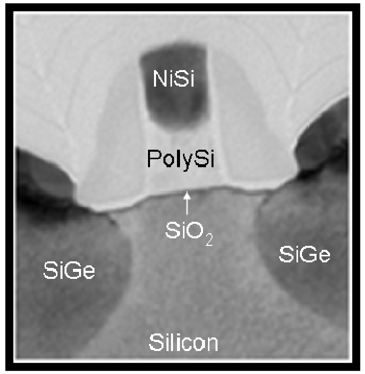
Improved Transistor Density	~2x
Improved Transistor Switching Speed	>20%
Reduced Transistor Switching Power	~30%
Reduction in gate oxide leakage power	>10x



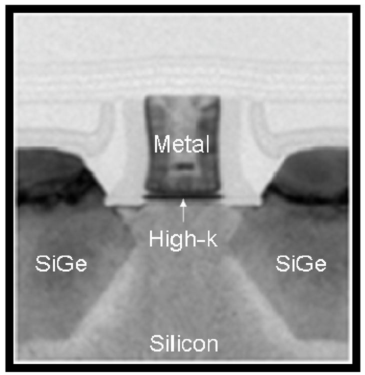
6-transistor 65 nm S-RAM Cell
0.570 µm2



45 nm S-RAM Cell
0.346 µm2



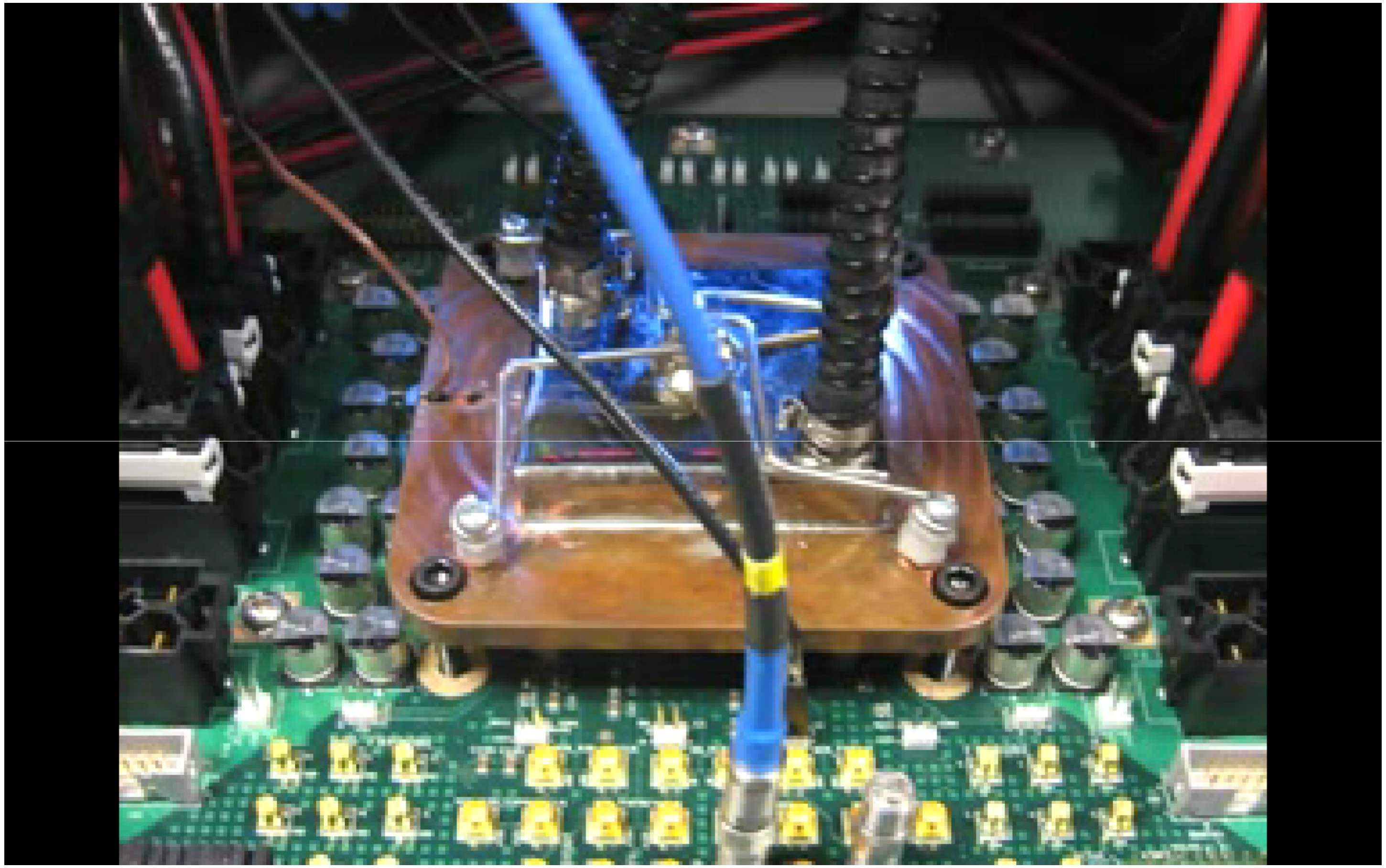
65 nm Transistor



45 nm HK + MG

Enables New Features, Higher Performance, Greater Energy Efficiency

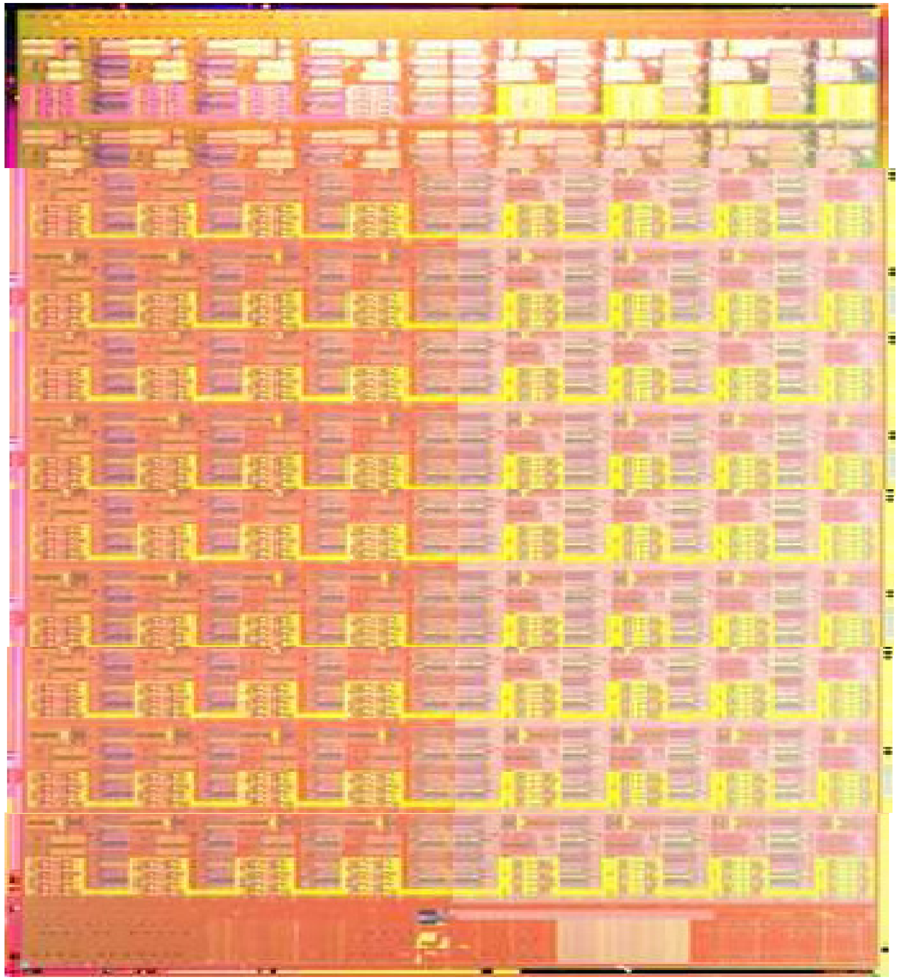
Intel's Teraflops Research Chip



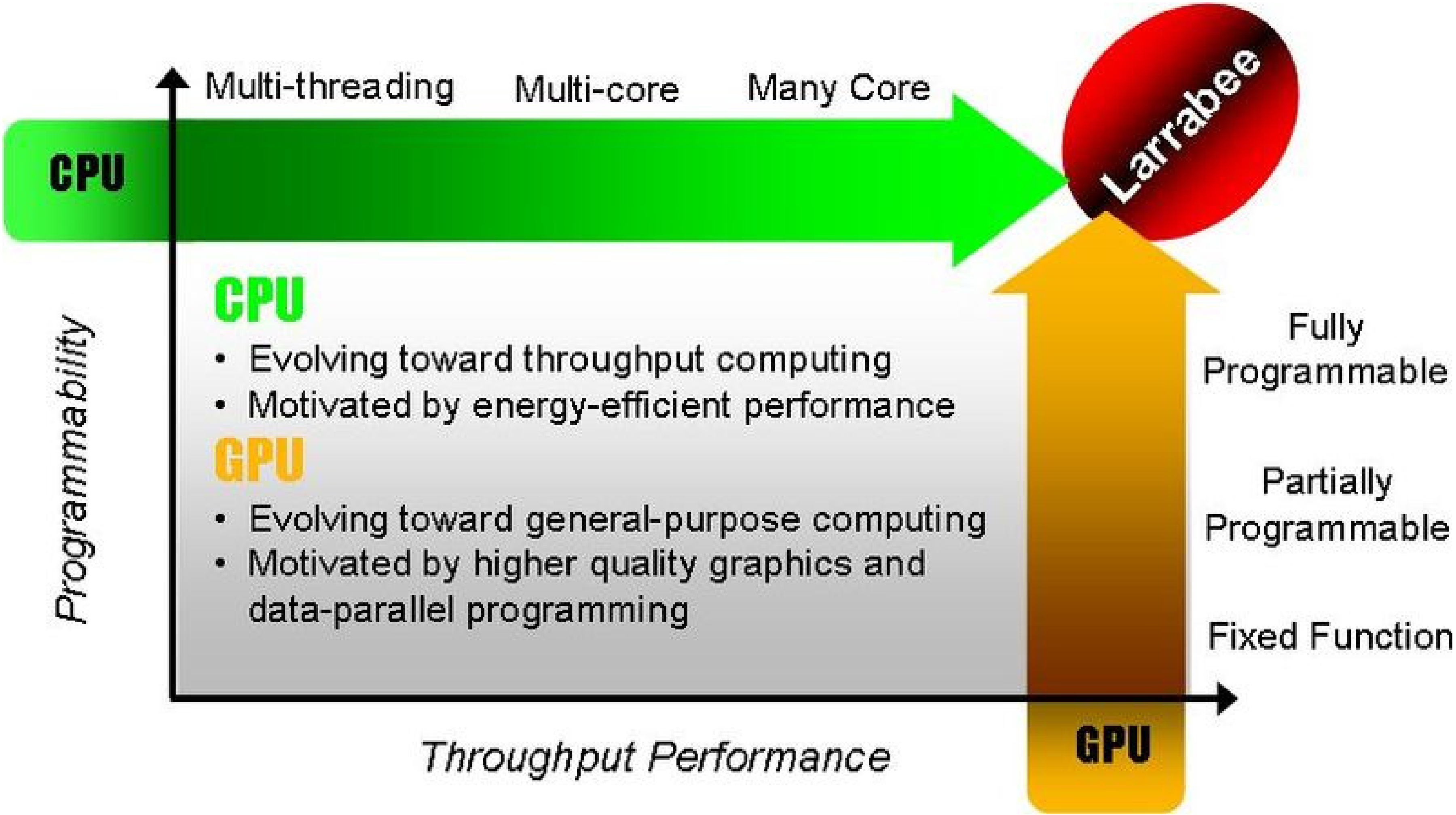
Podtech_Intel_Research_Day_Terascale.flv

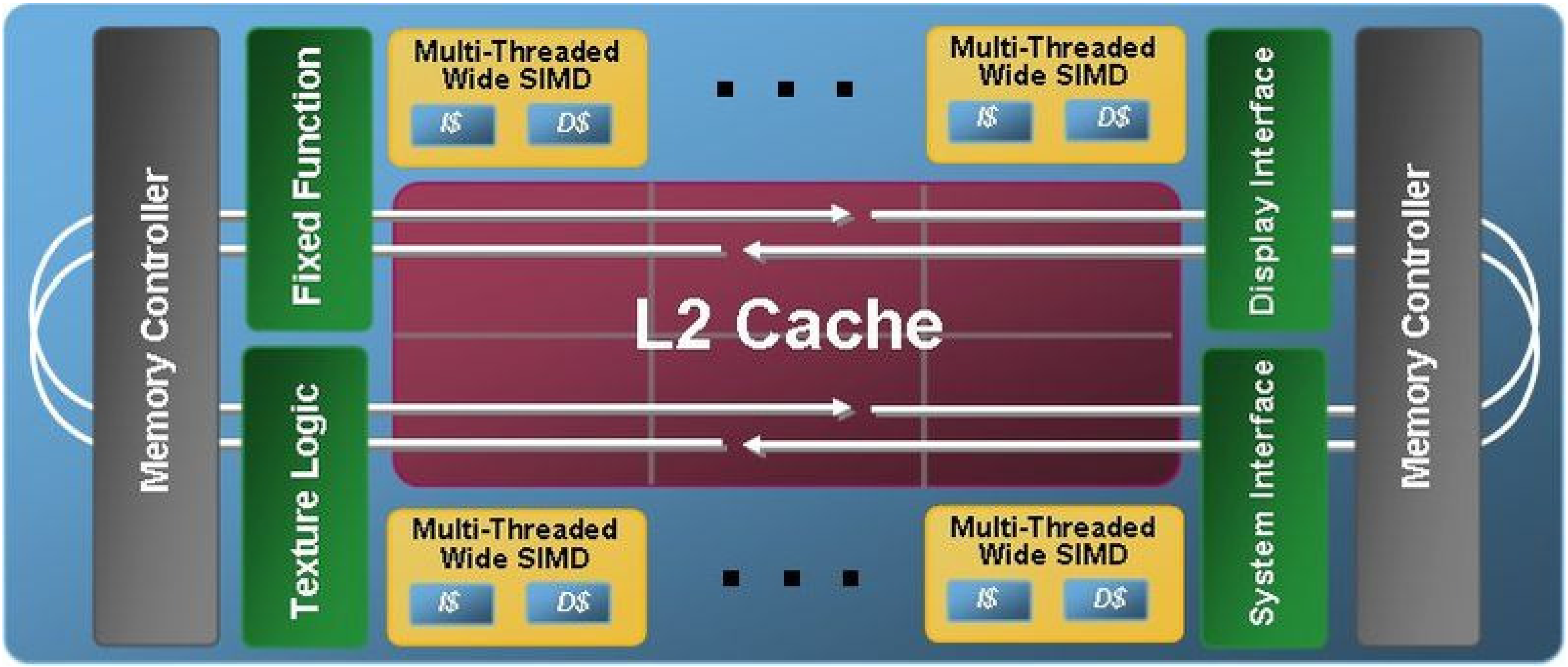
Intel's Teraflops Research Chip

Speed GHz	Power Watts	Perf. Teraflops
3.16	62	1.01
5.1	175	1.63
5.7	265	1.81



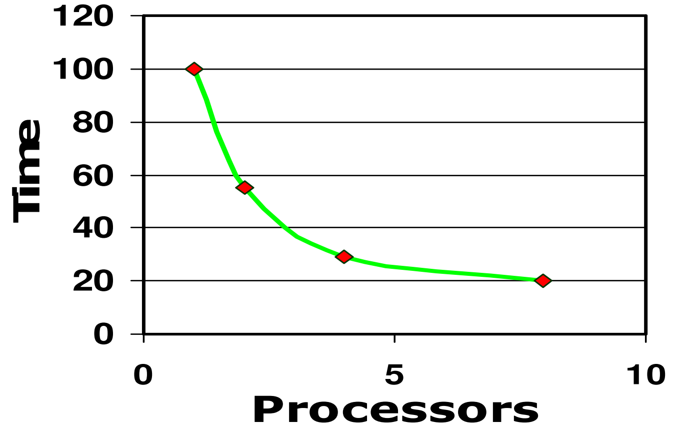
Larrabee


Multi-threading
Multi-core
Many Core
Larrabee
CPU
Programmability
CPU
• Evolving toward throughput computing
• Motivated by energy-efficient performance
GPU
• Evolving toward general-purpose computing
• Motivated by higher quality graphics and data-parallel programming
Fully Programmable
Partially Programmable
Fixed Function
Throughput Performance
GPU

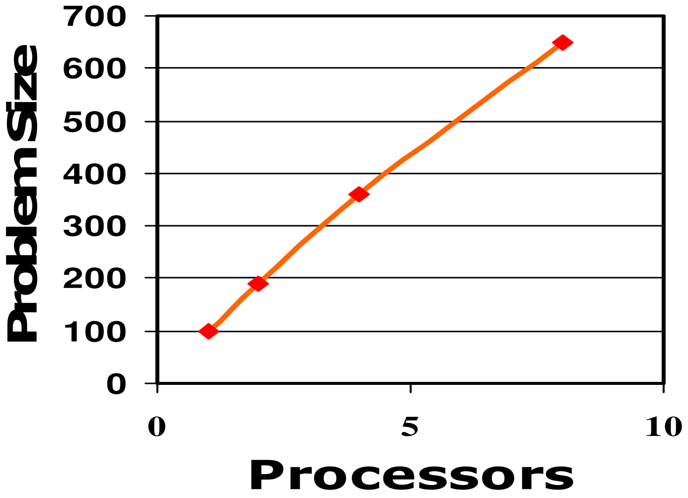

Memory Controller
Fixed Function
Texture Logic
Multi-Threaded Wide SIMD
I$
D$
Multi-Threaded Wide SIMD
I$
D$
L2 Cache
Multi-Threaded Wide SIMD
I$
D$
Multi-Threaded Wide SIMD
I$
D$
Display Interface
System Interface
Memory Controller

What can we do with faster Computers?

- Solve problems faster
  - Reduce turn-around time of big jobs
  - Increase responsiveness of interactive apps



- Get better solutions in the same amount of time
  - Increase resolution of models
  - Make model more sophisticated



Optimising Code

Section 2

Two points to address before you start parallelising

- Will buying a faster computer solve your problem?

Dr Yann Golanski, York

Tip 1

Just buy a faster machine!

First look at how much it will cost you to make your program parallel. If it will take say 2 months of coding, can you just buy a faster machine that will give you the speedup you want? Of course once you reach the limits of a machines speed, you are going to have to then do some parallelisation.

- Maybe Serial Optimisation will be sufficient.

A Three-Tiered Tuning Model

Tuning Level	Question being asked	Examples of issues
System wide	Can my system be 'tuned' to improve the performance of my application	Network, disk and memory performance. Intrusion by 3rd party programs such as virus scanners.
Application Heuristics	Can my application code or heuristics to improve performance?	Code redundancy. Inefficient program algorithms. Poor memory allocation strategies. Bad \ missing threading implementation.
Architectural Bottlenecks	Is the CPU architecture being used at its best?	Stalls in CPU pipeline. Data alignment . Cache misses. Using expensive instructions. Failing to use latest generation optimised instructions.

Compiler generated Optimisations

Global Compiler Options

Inter-procedural Optimisations

Profile Guided Optimisations

Vectorisation

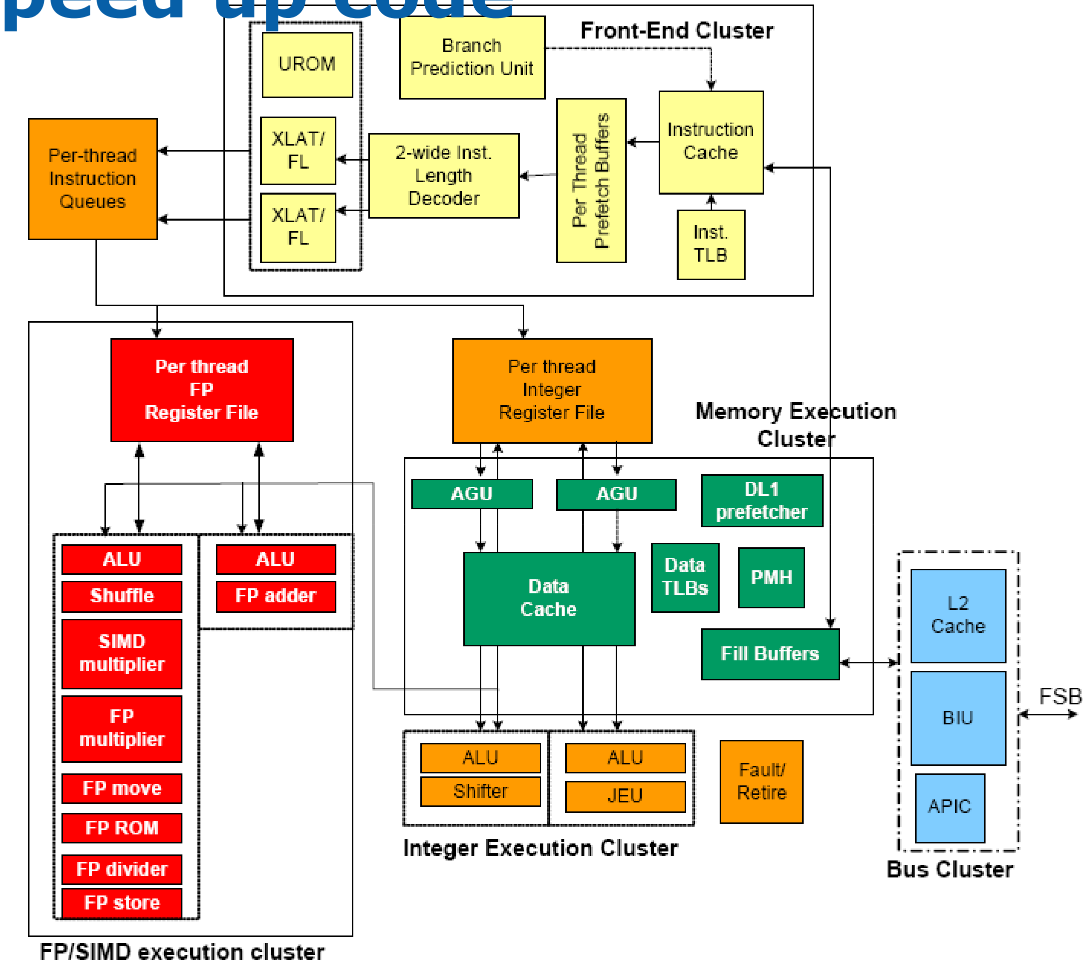
Parallelisation

Example of hand-crafted SSE instructions

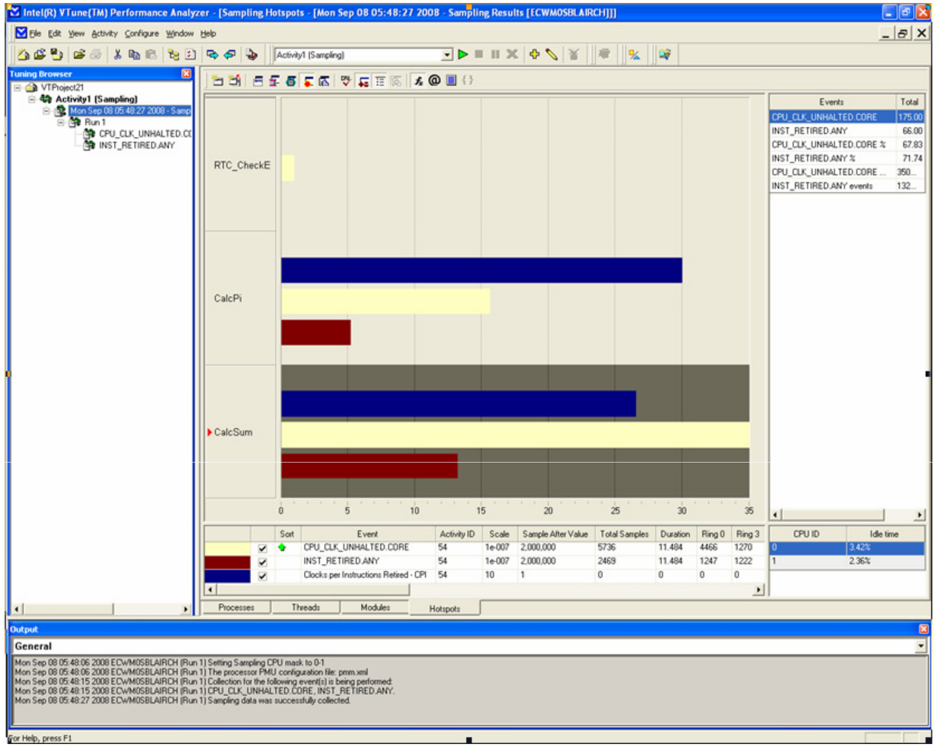
```
bool SSEHasNumber(SUDOKU *pPuzzle,__m128i BinArray[], int
    i, int j)
{
        __m128i Tmp1 = (_mm_and_si128(pPuzzle->BinNum[i-1]

    return false;
}
```

	Time Taken	Speedup
No SSE	4.55 sec	1
With SSE	0.19 sec	24

Modern Architectures have lots of features to help speed up code



Intel® VTune™ Performance Analyzer



Graphical tool

Helps characterise runtime performance

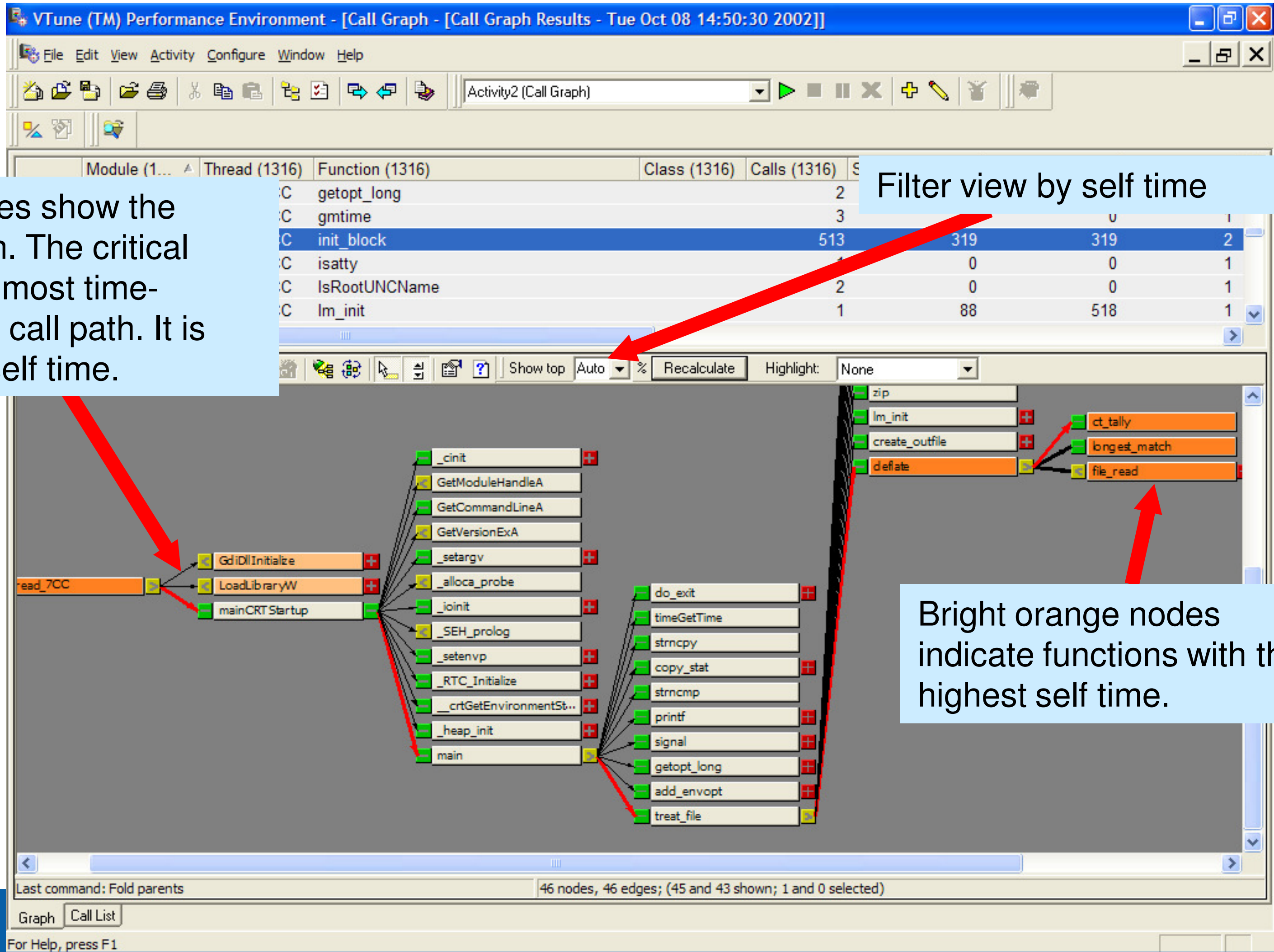
System-wide View of application environment

Use to tune serial and parallel code

Use to identify Hot Spots in Code

Use to generate a call graph

Call graph: Application workflow

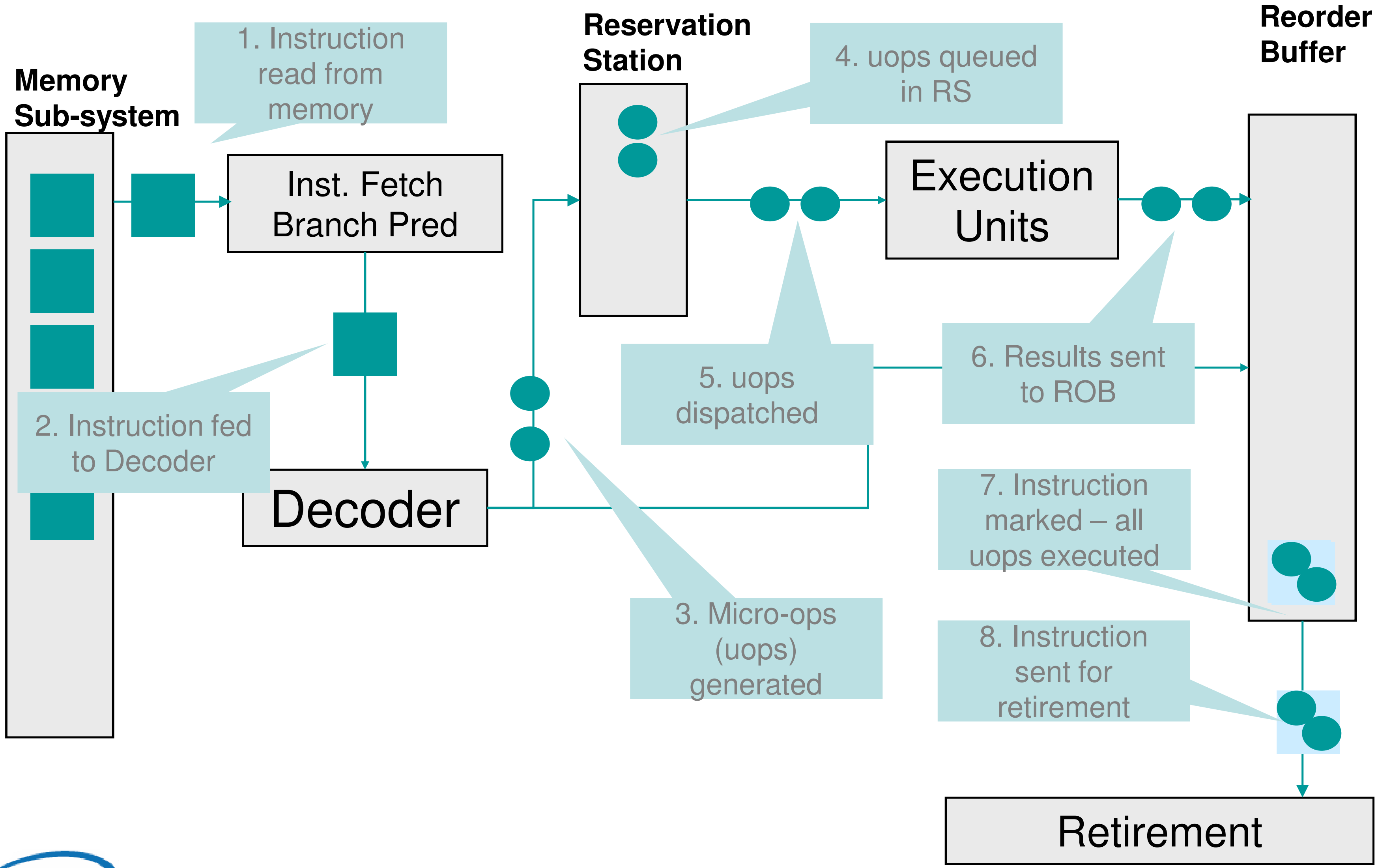


The red lines show the critical path. The critical path is the most time-consuming call path. It is based on self time.

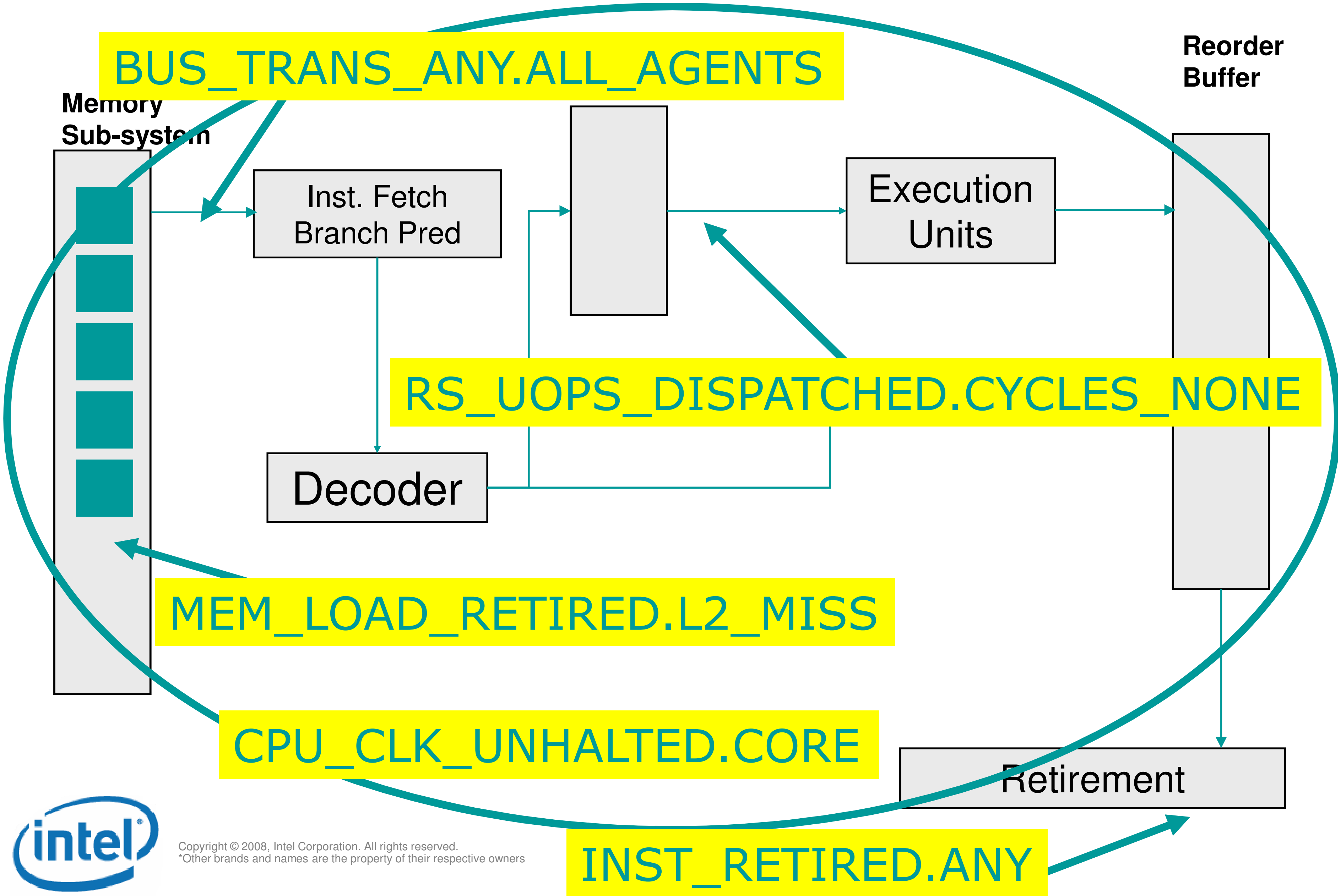
Filter view by self time

Bright orange nodes indicate functions with the highest self time.

The life of a program instruction



Hardware Performance Events



Demo 0 – Using Intel© VTune™ Performance Analyzer

From 1 to 1,000,000

Notice the System Wide View – Can you see any problems?

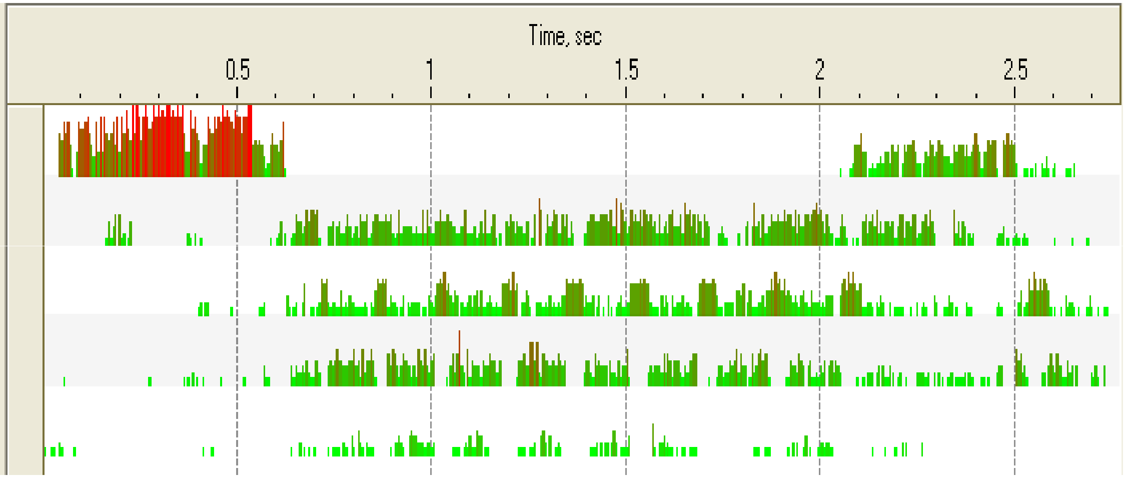
mcshield.exe	1,146
cidaemon.exe	1,067
02_Hotspot Analysis.exe	838
csrss.exe	807
pid_0x0	[illegible]
cisvc.exe	178
rundll32.exe	155
pid_0x4	111
services.exe	97

Process=02_Hotspot Ana

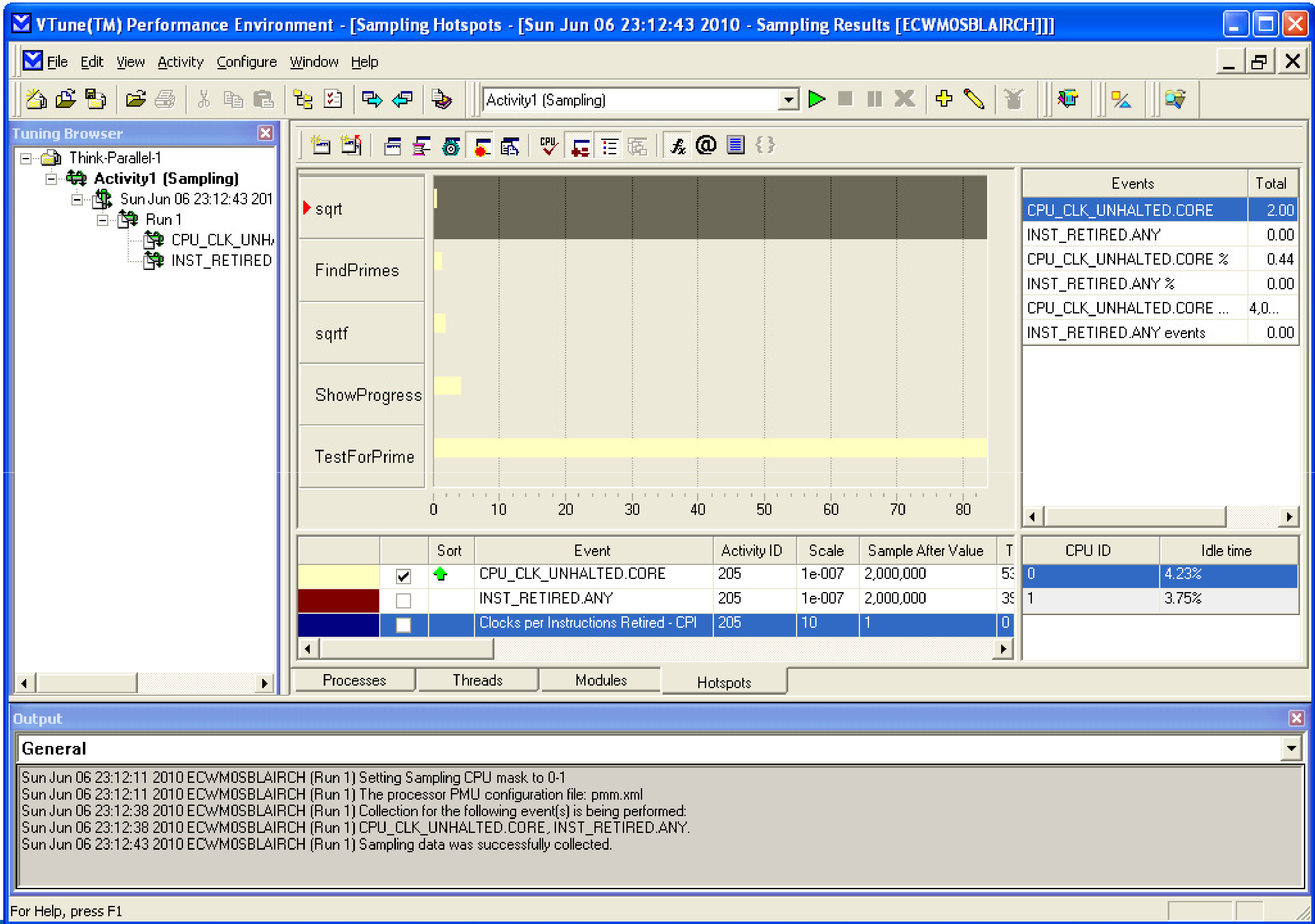
What is the problem here?

- VTune Sample-over-time view

ProcessName	ProcessID	CPU_CLK_UNHALT samples
mcshield.exe	2596	1146
cidaemon.exe	18296	1067
02_Hotspot Analysis.exe	12464	838
csrss.exe	2020	807
pid_0x0	0	214



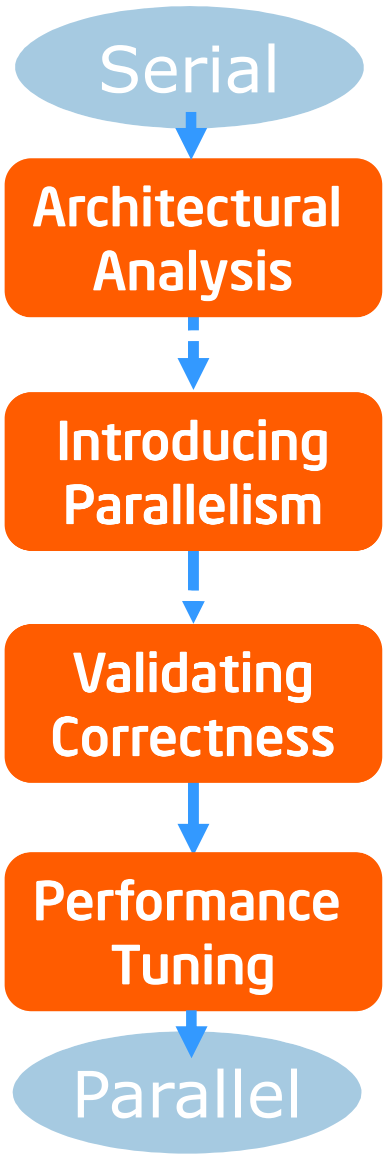
The Hotspot


VTune(TM) Performance Environment - [Sampling Hotspots - [Sun Jun 06 23:12:43 2010 - Sampling Results [ECWMOSBLAIRCH]]]
File Edit View Activity Configure Window Help
Activity1 (Sampling)
Tuning Browser
Think-Parallel-1
Activity1 (Sampling)
Sun Jun 06 23:12:43 201
Run 1
CPU_CLK_UNH/
INST_RETIRED
sqrt
FindPrimes
sqrtf
ShowProgress
TestForPrime
0
10
20
30
40
50
60
70
80
Events
Total
CPU_CLK_UNHALTED.CORE 2.00
INST_RETIRED.ANY 0.00
CPU_CLK_UNHALTED.CORE % 0.44
INST_RETIRED.ANY % 0.00
CPU_CLK_UNHALTED.CORE ... 4,0...
INST_RETIRED.ANY events 0.00
Sort
Event
Activity ID
Scale
Sample After Value
CPU_CLK_UNHALTED.CORE 205 1e-007 2,000,000
INST_RETIRED.ANY 205 1e-007 2,000,000
Clocks per Instructions Retired - CPI 205 10 1
CPU ID
Idle time
0 4.23%
1 3.75%
Processes
Threads
Modules
Hotspots
Output
General
Sun Jun 06 23:12:11 2010 ECWMOSBLAIRCH (Run 1) Setting Sampling CPU mask to 0-1
Sun Jun 06 23:12:11 2010 ECWMOSBLAIRCH (Run 1) The processor PMU configuration file: pmm.xml
Sun Jun 06 23:12:38 2010 ECWMOSBLAIRCH (Run 1) Collection for the following event(s) is being performed:
Sun Jun 06 23:12:38 2010 ECWMOSBLAIRCH (Run 1) CPU_CLK_UNHALTED.CORE, INST_RETIRED.ANY.
Sun Jun 06 23:12:43 2010 ECWMOSBLAIRCH (Run 1) Sampling data was successfully collected.
For Help, press F1

Four Steps in Moving to Parallel

Section 2

Steps in moving from Serial to Parallel



Key Questions

Design

- Is my program parallel?
- Where is the best place to parallelise my program?
- How can I get my program to run faster?
- What's the expected speedup?

Code & Debug

- How?
- How difficult?
- Is my code still working?

Verify

- Is the parallelism correct?
- Do I have deadlocks or data races?
- Do I have memory errors?
- Does my program still work as intended?

Tune

- Do my tasks do equal amounts of work?
- Is my application scalable?
- Is the threading running efficiently?

Intel Software Tools Supporting Parallel Design Cycle

Serial

↓

Architectural Analysis

↓

Introducing Parallelism

↓

Validating Correctness

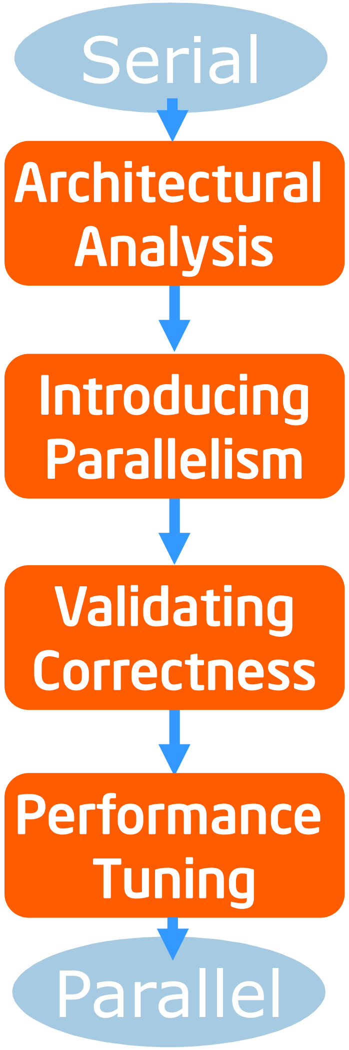
↓

Performance Tuning

↓

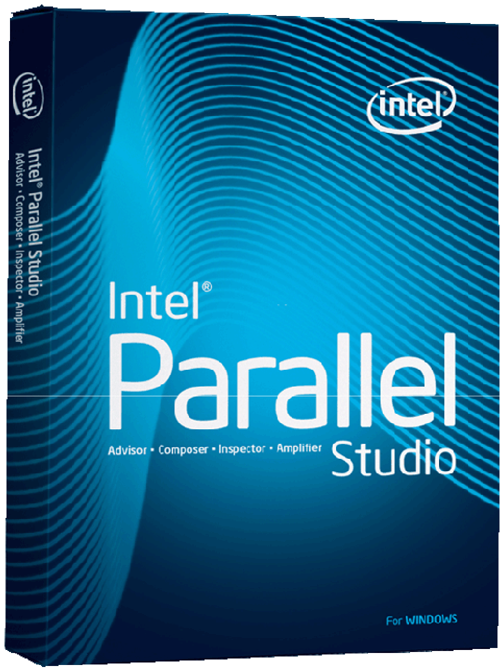
Parallel

Tools



Existing Intel Software	Intel Parallel Studio
Intel® VTune™ Performance Analyzer	Advisor/Amplifier
Intel Compilers Parallel Libraries	Composer
Intel® Thread Checker	Inspector
Intel® Thread Profiler	Amplifier

For Microsoft Visual Studio* C++ architects, developers, and software innovators creating parallel Windows* applications.

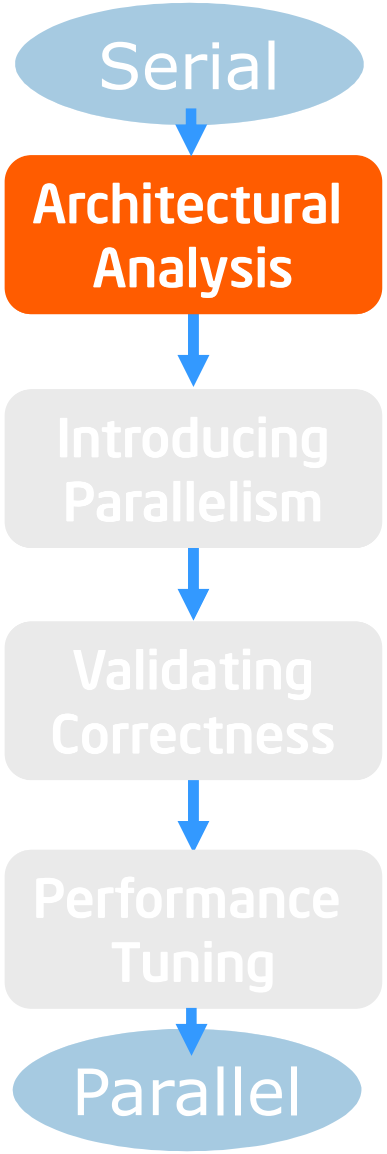


Microsoft Visual Studio* plug-in
End-to-end product suite for parallelism
Forward scaling to many core

Intel® Parallel Studio includes:

- Intel® Parallel **Advisor Lite** **
- Intel® Parallel **Composer**
- Intel® Parallel **Inspector**
- Intel® Parallel **Amplifier**

*** Beta – from whatif.intel.com*



Step 1

Which part of my code should I make Parallel?

Key Questions - Design

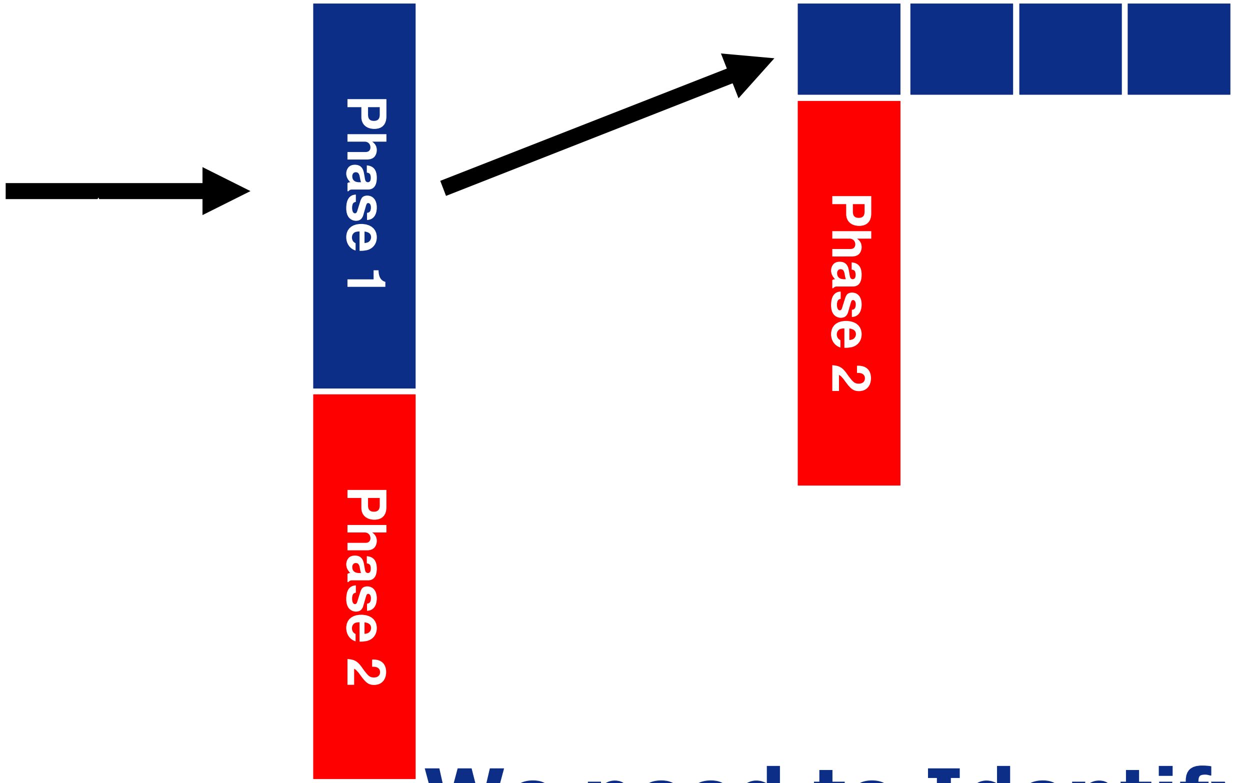
Is my program parallel?

Where is the best place to parallelise my program?

How can I get my program to run faster?

What's the expected speedup?

Identifying best parts to Parallelize



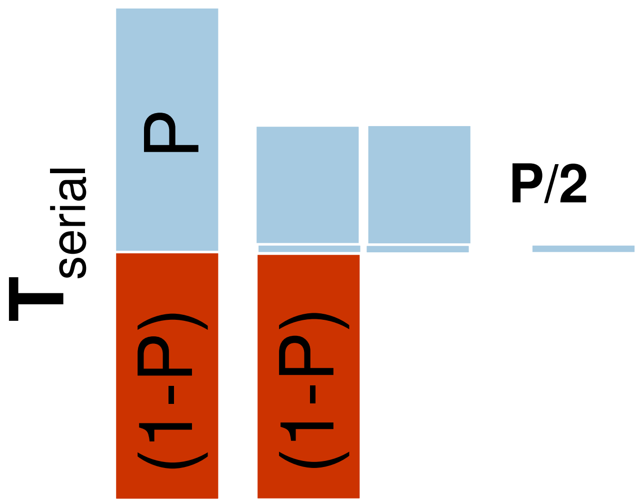
We need to Identify Hotspots

Why use parallelism? Amdahl's Law

Describes the upper bound of parallel execution speedup

Serial code limits speedup

n = 2



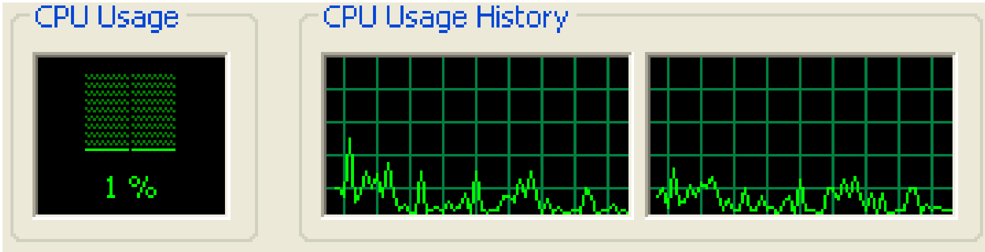
$$\begin{array}{c} 0.5 \quad + \quad 0.25 \\ T_{parallel} = \{(1-P) + P/n\}\, T_{serial} \end{array}$$

n = number of processors

Speedup = T_{serial} / $T_{parallel}$ **1.0/0.75 = 1.33**

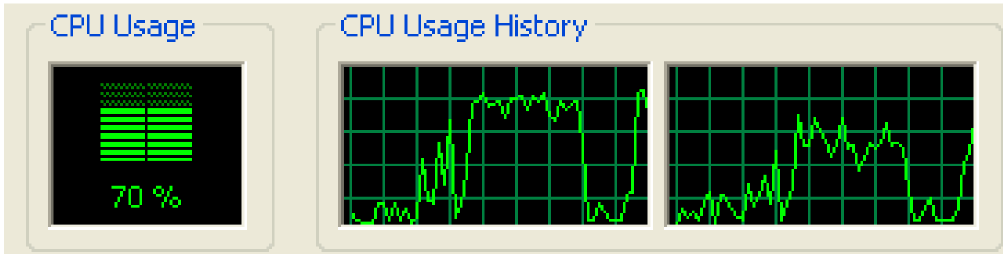
Some code is not worth making parallel…

Don't parallelise code
- just because it's clever
- With low CPU utilisation
- I/O bound



- Do parallelise code that
  - Eats significant CPU cycles

- You need to get visibility of the runtime behaviour



Architectural Extensions can speed up your code

- Always optimise your code
- Even if you don't go parallel, some architectural features can still give significant speed-up
- Example, SSE extensions

Our Application - Prime Number Generator

i	factor
61	3 5 7
63	3
65	3 5
67	3 5 7
69	3
71	3 5 7
73	3 5 7 9
75	3 5
77	3 5 7
79	3 5 7 9

```
bool TestForPrime(int val)
{   // let's start checking from 3
    int limit, factor = 3;
    limit = (long)(sqrtf((float)val)+0.5f);
    while( (factor <= limit) && (val % factor) )
            factor ++;

    return (factor > limit);
}

void FindPrimes(int start, int end)
{
    int range = end - start + 1;
    for( int i = start; i <= end; i += 2 )
    {
        if( TestForPrime(i) )
            globalPrimes[gPrimesFound++] = i;
        ShowProgress(i, range);
    }
}
```

Demo 1 – Getting the Benchmark

From 1 to 1,000,000

Our Application - Prime Number Generator

i	factor
61	3 5 7
63	3
71	3 5 7
73	3 5 7 9
75	3 5
77	3 5 7
79	3 5 7 9

```
bool TestForPrime(int val)
{   // let's start checking from 3
    int limit, factor = 3;
    limit = (long)(sqrtf((float)val)+0.5f);
    while( (factor <= limit) && (val % factor) )
            factor ++;

    return (factor > limit);
```

```
Running 00_Benchmark Serial.exe
100%

   78498 primes found between      0 and 1000000 in     2.48 secs
```

```
    int range = end - start + 1;
    for( int i = start; i <= end; i += 2 )
    {
        if( TestForPrime(i) )
            globalPrimes[gPrimesFound++] = i;
        ShowProgress(i, range);
    }
}
```

Optimise the serial code first

- Using Intel Compiler to automatically generate SSE instructions.
  - Code ran twice as fast
  - No change made to original code

Calculating Pi		
	Time Secs	Speedup
No SSE	1.29	1.00
With SSE	0.66	1.95

```
004018D9  movaps     xmmword ptr [esp],xmm0
004018DD  paddd      xmm5,xmm6
004018E1  addpd      xmm7,xmm3
004018E5  mulpd      xmm7,xmm2
004018E9  add        eax,8
004018EC  mulpd      xmm7,xmm7
004018F0  movaps     xmm0,xmmword ptr ds:[406770h]
004018F7  addpd      xmm7,xmm1
004018FB  divpd      xmm0,xmm7
004018FF  cvtdq2pd   xmm7,xmm5
00401903  paddd      xmm5,xmm6
00401907  addpd      xmm4,xmm0
0040190B  movaps     xmm0,xmmword ptr ds:[406770h]
00401912  addpd      xmm7,xmm3
00401916  mulpd      xmm7,xmm2
0040191A  mulpd      xmm7,xmm7
0040191E  addpd      xmm7,xmm1
00401922  divpd      xmm0,xmm7
00401926  movaps     xmm7,xmmword ptr [esp]
0040192A  addpd      xmm7,xmm0
0040192E  cvtdq2pd   xmm0,xmm5
00401932  movaps     xmmword ptr [esp],xmm7
```

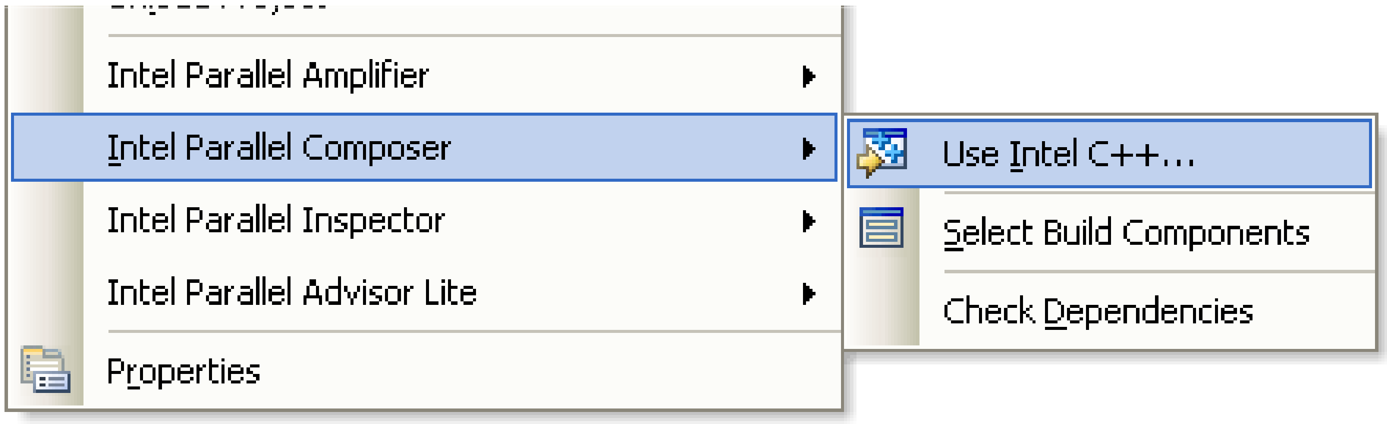
Example of SSE compiler-generated instructions

Demo 2 – Using the Intel Compiler

From 1 to 1,000,000

Swapping compilers.

- From solution drop-down menu
- Action is reversible



Program built and run with Intel compiler

```
Running 00_Benchmark Serial.exe
100%

   78498 primes found between      0 and 1000000 in    2.48 secs
```

```
Running 01_Converted To Intel.exe
100%

   78498 primes found between      0 and 1000000 in    2.28 secs
```

Speedup 1.09

Identifying Hotspots

Pinpointing places where an application could be parallelised

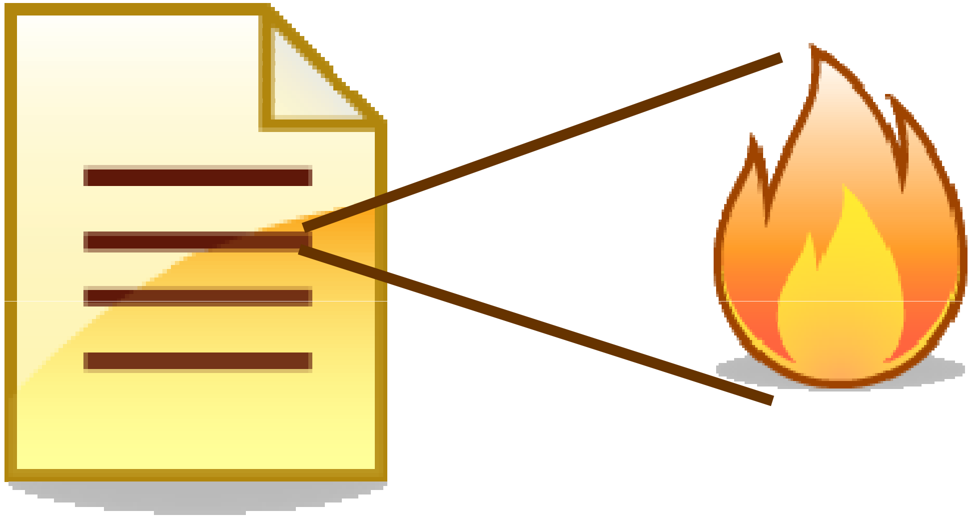
The Big Question

"How can I make my code run faster?"

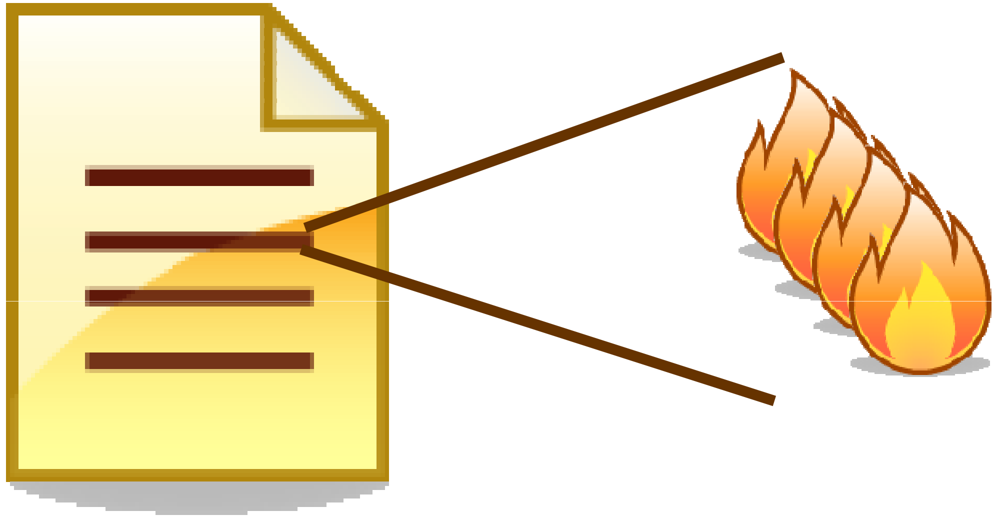
Today's Question

"Where do I split up my code to take advantage of multiple CPU cores?"

The task, Identifying the Hot Spot...

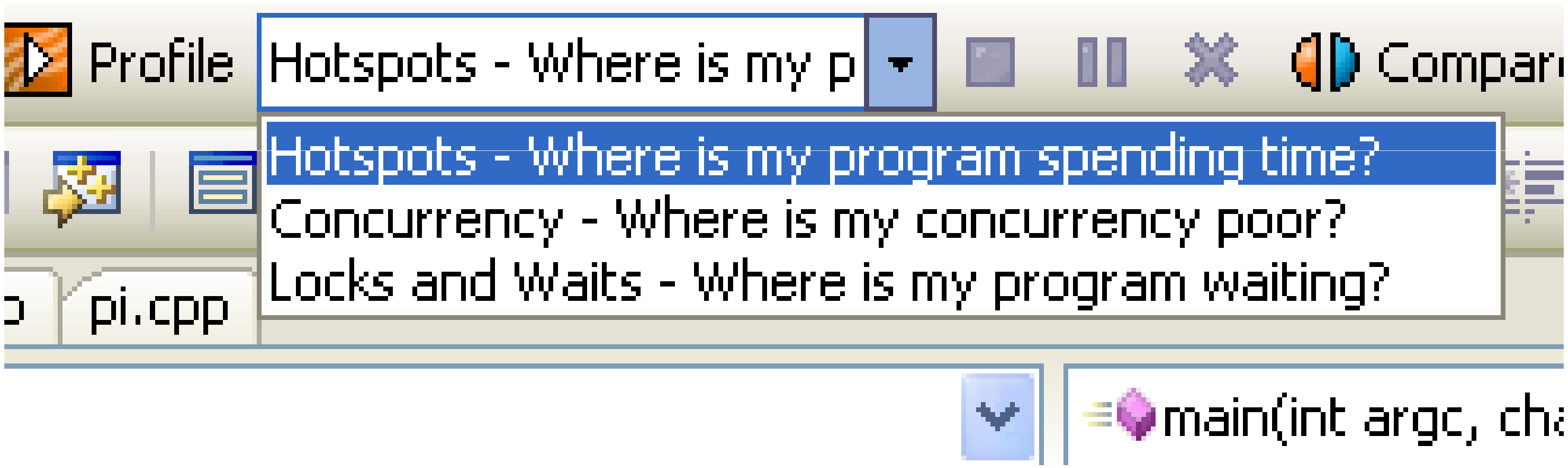


… and Splitting up the Work.



Demo 2 – Finding the Hotspots

Finding a Hot Spot



Where to Parallelise – Amplifier

Call Stack	CPU Time:Total	CPU Time:Self
Total	100.0%	0s
BaseProcessStart	100.0%	0s
_tmainCRTStartup	100.0%	0s
main	100.0%	0s
FindPrimes	100.0%	0s
TestForPrime	24.4%	0.328s
ShowProgress	75.6%	1.016s

Call Stack

Hotspot

Design: What's the expected speedup?

- Use Amdhals Law

Speedup = 1/[s+(1-s)/n + H(n)]
s is serial part (fraction of 1)
H is parallel overhead (ignore)
n is number of cores

main	0s
FindPrimes	0s
TestForPrime	0.328s
ShowProgress	1.016s

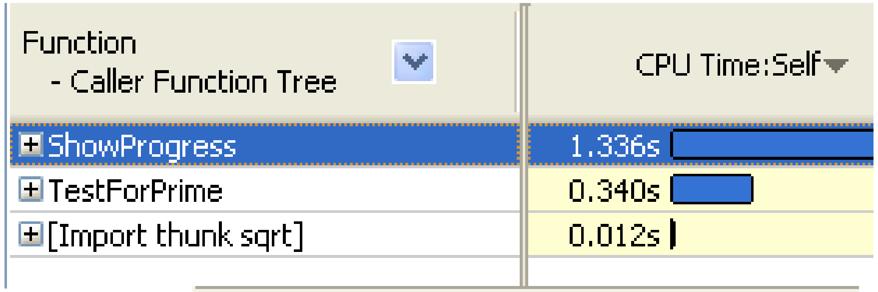
Elapsed Time:	3.347s
CPU Time:	1.344s
Unused CPU Time:	5.351s
Core Count:	2
Threads Created:	1

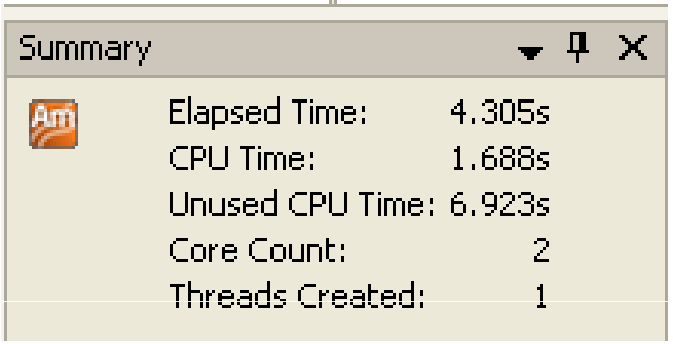
S = 0
Speedup = 1 / [0 + (1 - 0) / 2]
= 1 / [0 + 0.5]

Speedup = 2 (i.e. new speed ~ 0.672 seconds)

Alternate Calculation

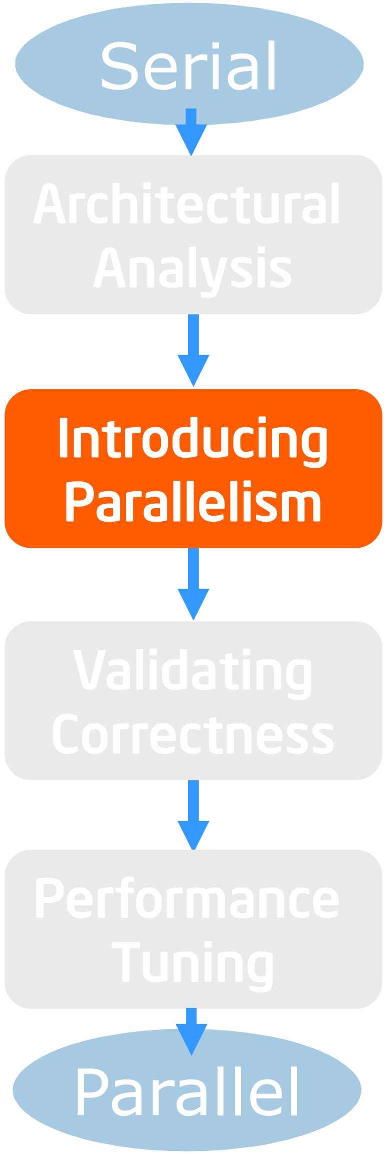
Speedup = 1/[s+(1-s)/n + H(n)]
s is serial part (fraction of 1)
H is parallel overhead (ignore)
n is number of cores





S = 1 - (1.688 – 0.012)/1.688 = .007
Speedup = 1 / [.007 + (1 - .007) / 2]
= 1 / [0007 + 0.4965]

Speedup = 1.986 (i.e. CPU Time ~ 0.850 seconds)



Step 2

Implement Parallelism in code

Key Questions – Code & Debug

S
1
2
3
4
D

How?

How difficult?

Is my code still working?

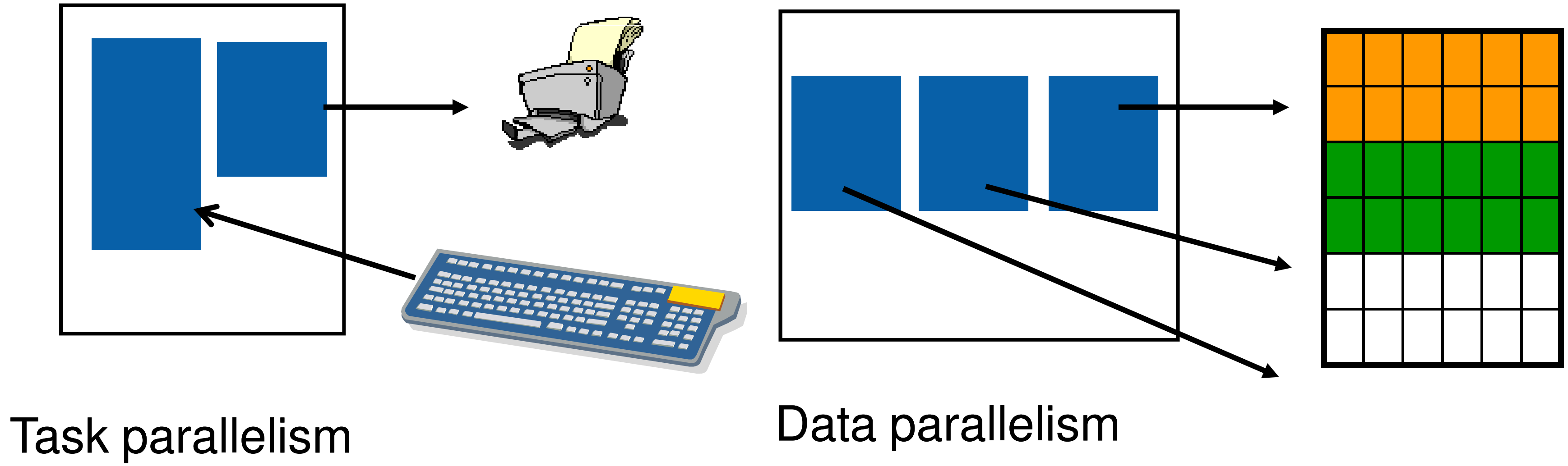
Common types of parallelism

- Functional or Task Parallelism
- Data Parallelism
- Software Pipelining

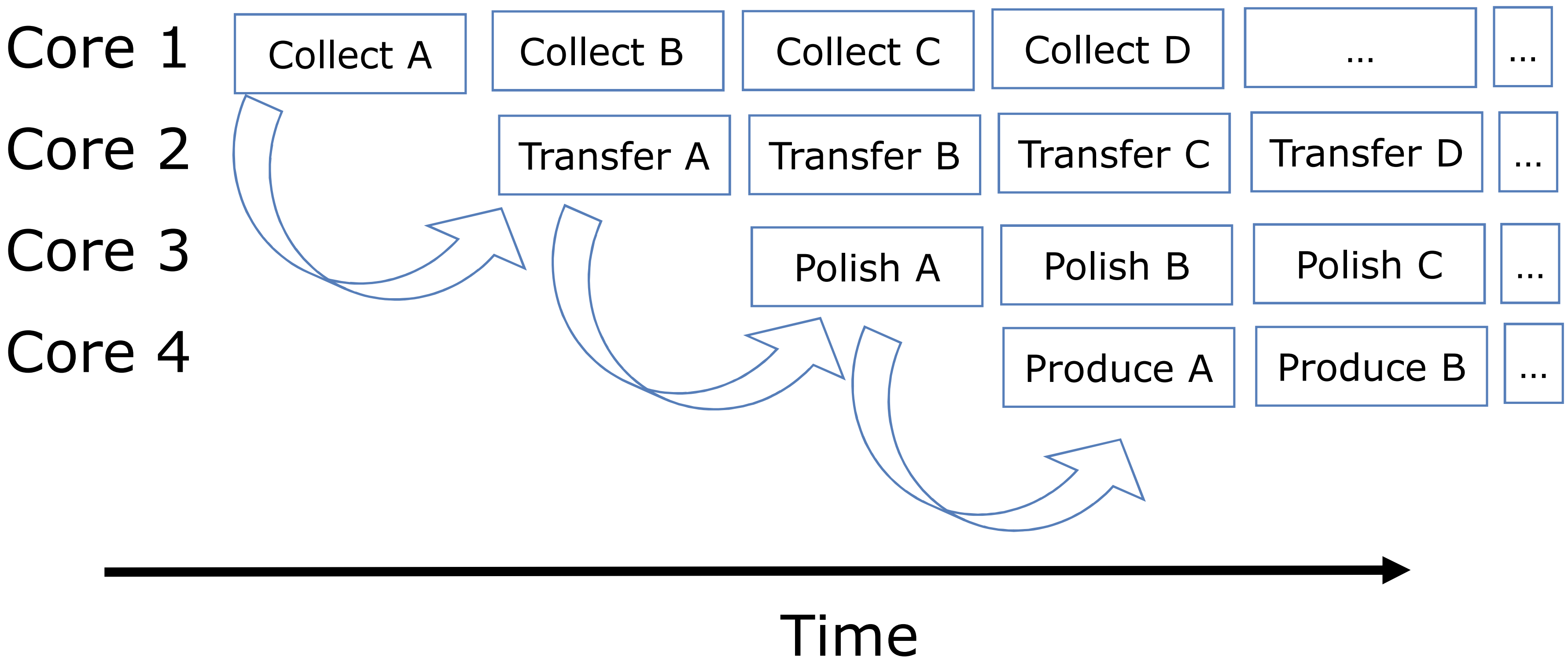
Task and Data Parallelism

- Different job for each thread
- e.g. one thread prints, another reads keyboard

- Splitting workload between multiple identical threads
- e.g. three identical threads perform calculations on data array



Software Pipeline



Question

- How many different ways can you think of to implement parallelism?
  - *E.g OpenMP, …, …*

Intel's Family of Parallel Models

Most Composable Parallel Models

- Intel® Threading Building Blocks (TBB)
- Intel® Cilk
- Intel® Ct Technology
- Array Notations and SIMD pragmas

Fixed Function Libraries

- Intel® Math Kernel Library (MKL)
- Intel® Integrated Performance Primitives (IPP)

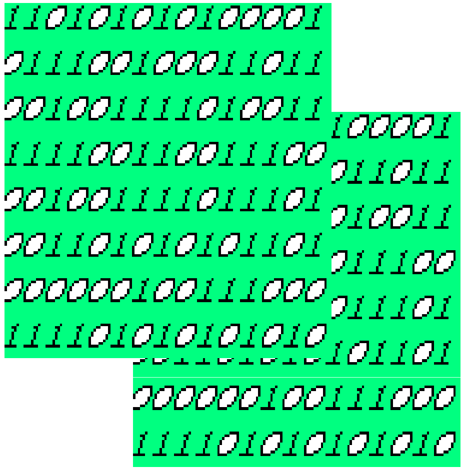
Other Supported Standards

- MPI
- OpenMP*
- Intel® OpenCL SDK

Research Initiatives

- Intel® Concurrent Collections
- Intel® Cluster OpenMP
- Software Transactional Memory

Auto Parallelism



Loop-level parallelism automatically supplied by the compiler

Auto-parallelization

- Auto-parallelization: Automatic threading of loops without having to manually insert OpenMP* directives.

Windows*	Linux*	Mac*
/Qparallel	-parallel	-parallel
/Qpar_report[n]	-par_report[n]	-par_report[n]

- Compiler can identify "easy" candidates for parallelization, but large applications are difficult to analyze.

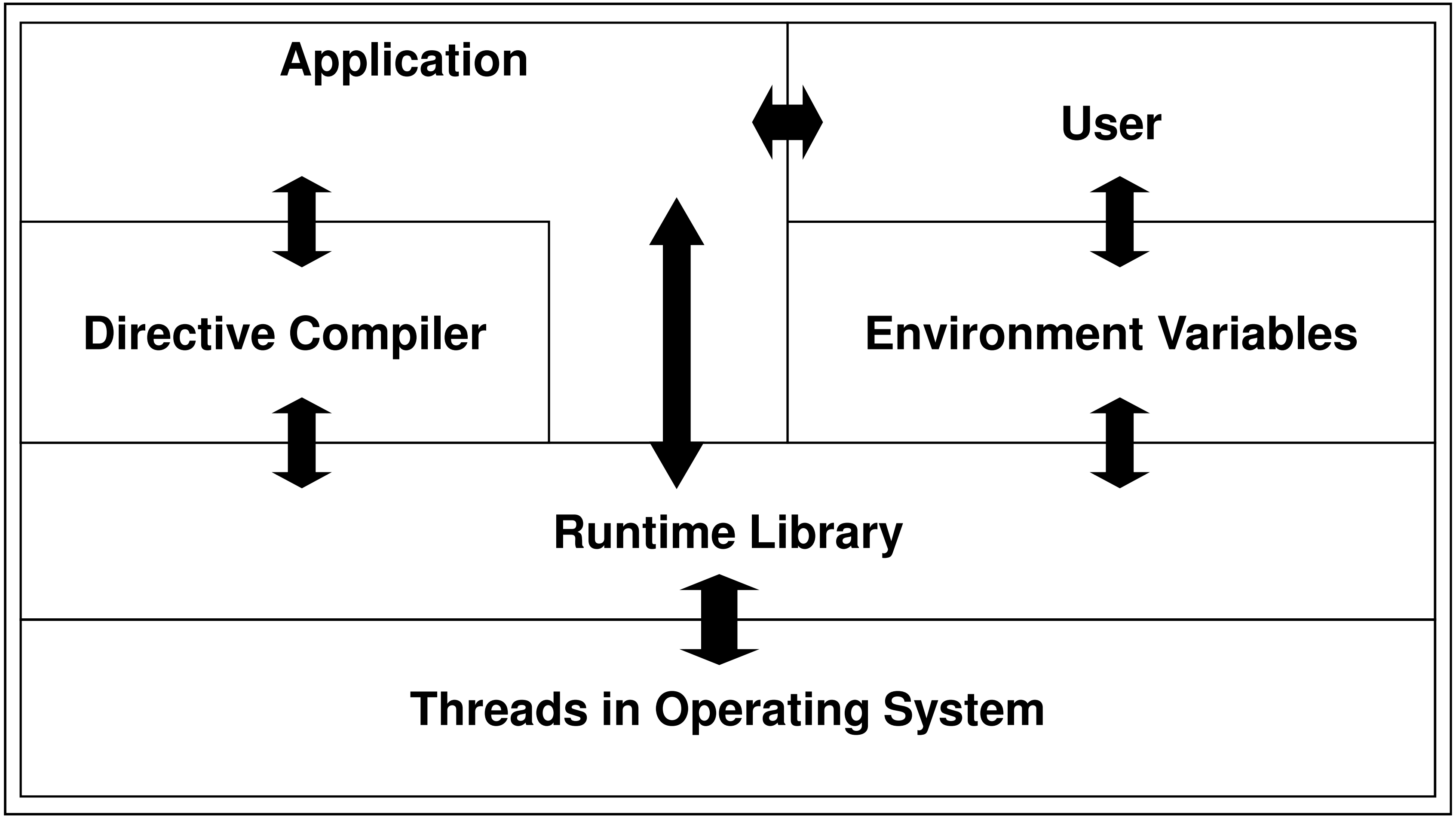
Optimisation Results – pi application

Optimisation	Time Taken (secs)	Speedup
default	0.938	1
auto-vectorisation	0.375	2.5
auto-parallelism	0.516	1.8
auto-vec. & auto-par.	0.203	4.6

OpenMP Architecture

- Fork-Join Model
- Worksharing constructs
- Synchronization constructs
- Directive/pragma-based parallelism
- Extensive API for finer control

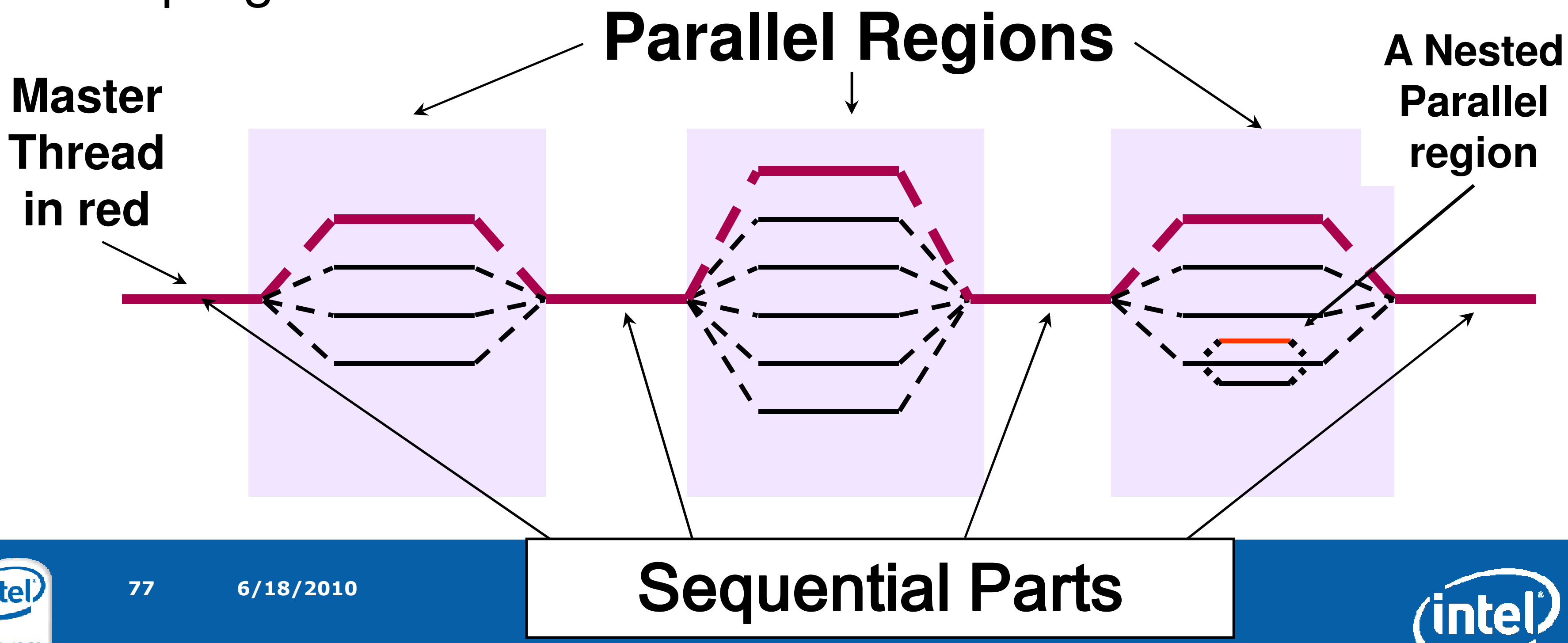
OpenMP Runtime



OpenMP Programming Model:

Fork-Join Parallelism:

- Master thread spawns a team of threads as needed.
- Parallelism added incrementally until performance are met: i.e. the sequential program evolves into a parallel program.



Introducing Parallelism

```
#pragma omp parallel for
     for( int i = start; i <= end; i+= 2 ){
        if( TestForPrime(i) )
           globalPrimes                    = i;
        ShowProgress(i, range);
     }
```

OpenMP

Divide iterations of the for loop

Create threads here for this parallel region

Demo 3 : Adding parallelism using #pragma omp for

Results – Open MP

```
Running 03_Naive OpenMP.exe
 50%

   36964 primes found between       0 and 1000000 in     1.45 secs
```

Amazing!

We Have a speed up!

Code: Is my code still working?

```
Running 01_Converted To Intel.exe
100%

   78498 primes found between        0 and 1000000 in     2.28 secs
```

```
Running 03_Naive OpenMP.exe
 50%

   36964 primes found between        0 and 1000000 in     1.45 secs
```

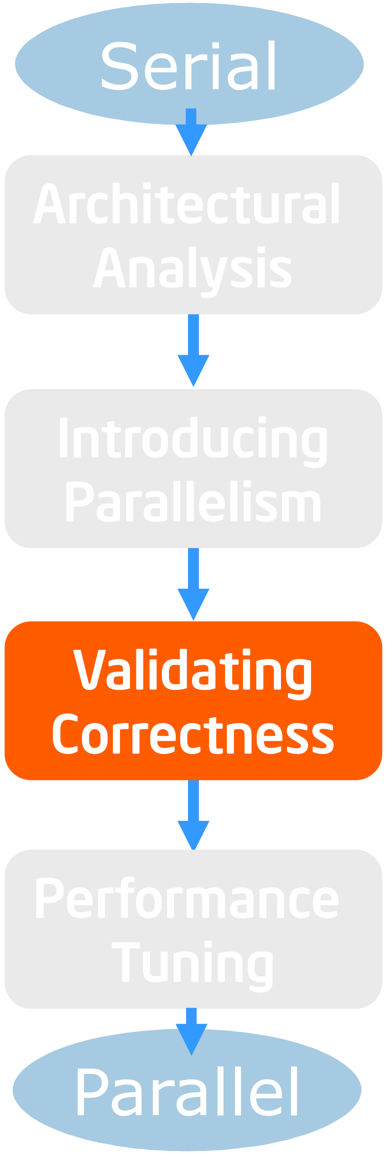
Bother !!!!!!!!!!!!!!!!!!!!!!

Number of primes is wrong

Questions

Are the results right?

Was the run quicker?



Step 3

Check for any problems

Key Questions - Verify

Is the parallelism correct?

Do I have deadlocks or data races?

Do I have memory errors?

Does my program still work as intended?

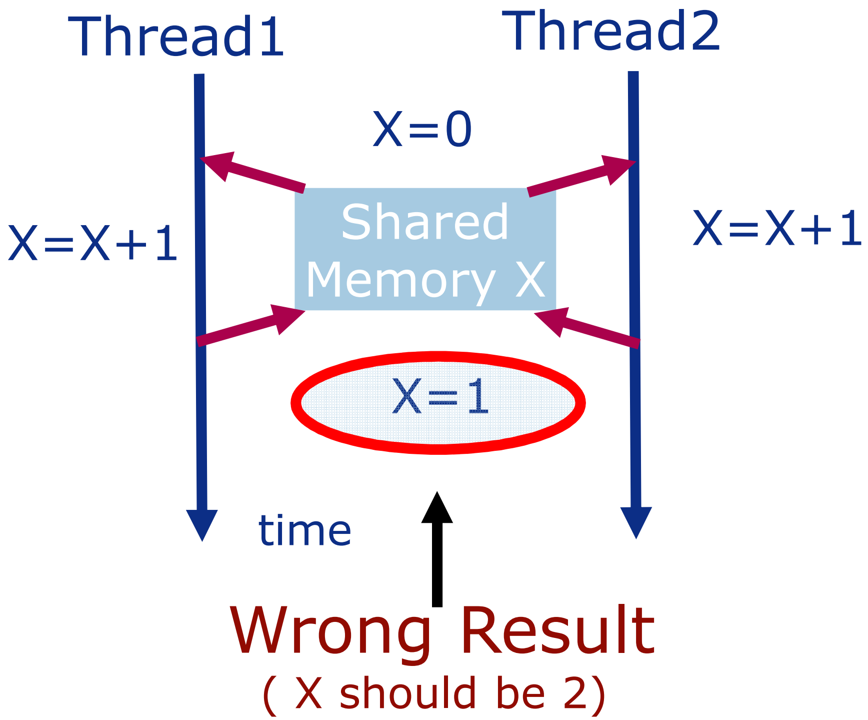
New paradigm requires new tools

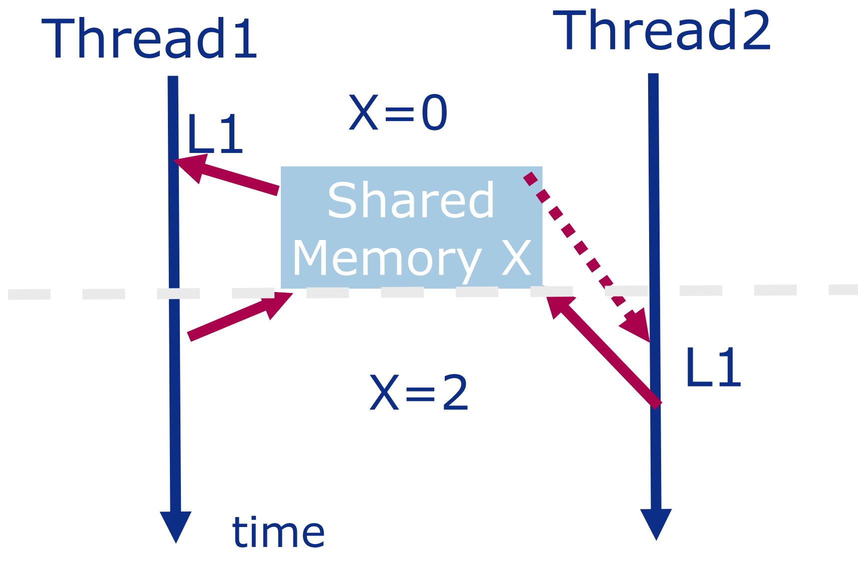
- Using traditional debugging tools is difficult /impossible
  - Printf – not re-entrant
  - Debugging several threads is notoriously hard
  - Many debuggers \ profilers are not multi-core enabled

- Multi-core tools are available

Non deterministic Error Sources in parallel Applications

- Shared Resources require locks
- Locks can
  - 'serialize' a program
  - lead to Deadlocks





Demo 4 – Checking for threading errors

Checking for Errors with Parallel Inspector

Overview → Sources ← Details

Problem Sets

ID		Problem	Sources
P1		Data race	PrimeOpenMP.cpp
P2		Data race	PrimeOpenMP.cpp
P3		Potential privacy infringement	PrimeOpenMP.cpp
P4		Potential privacy infringement	PrimeOpenMP.cpp
P5		Potential privacy infringement	PrimeOpenMP.cpp

The Offending Sources

Focus Observation: PrimeOpenMP.cpp:116 - Read

```
111
112        #pragma omp parallel for
113        for( int i = start; i <= end; i += 2 )
114        {
115            if( TestForPrime(i) )
116                globalPrimes[gPrimesFound++] = i;
117
118            ShowProgress(i, range);
119        }
120    }
121
122    int main(int argc, char **argv)
```

Related Observation: PrimeOpenMP.cpp:116 - Write

```
111
112        #pragma omp parallel for
113        for( int i = start; i <= end; i += 2 )
114        {
115            if( TestForPrime(i) )
116                globalPrimes[gPrimesFound++] = i;
117
118            ShowProgress(i, range);
119        }
120    }
121
122    int main(int argc, char **argv)
```

Protecting shared variables

```
#pragma omp parallel for
     for( int i = start; i <= end; i+= 2 ){
        if( TestForPrime(i) )
  #pragma omp critical
               globalPrimes[gPrimesFound++] = i;
        ShowProgress(i, range);
    }
```

Will create a critical section for this reference

```
#pragma omp critical
  {
    gProgress++;
    percentDone = (int)(gProgress/range *200.0f+0.5f)
  }
```

Will create a critical section for both these references

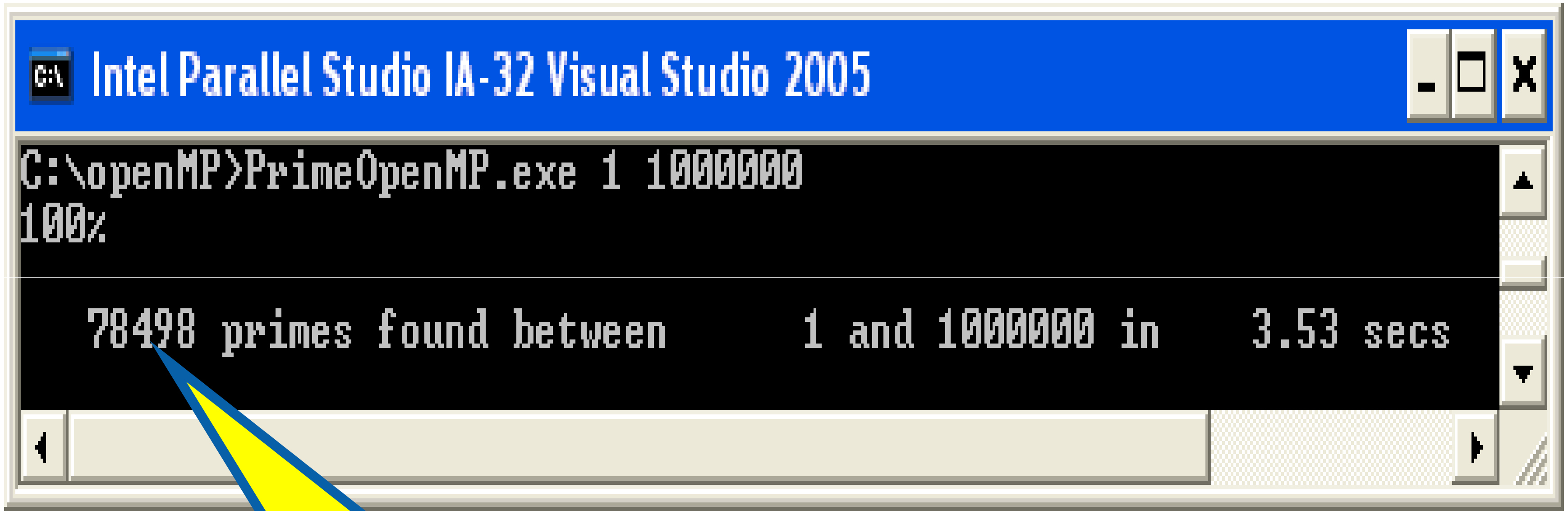
Demo 5 – Fixing the threading errors

Data Races Have Disappeared!

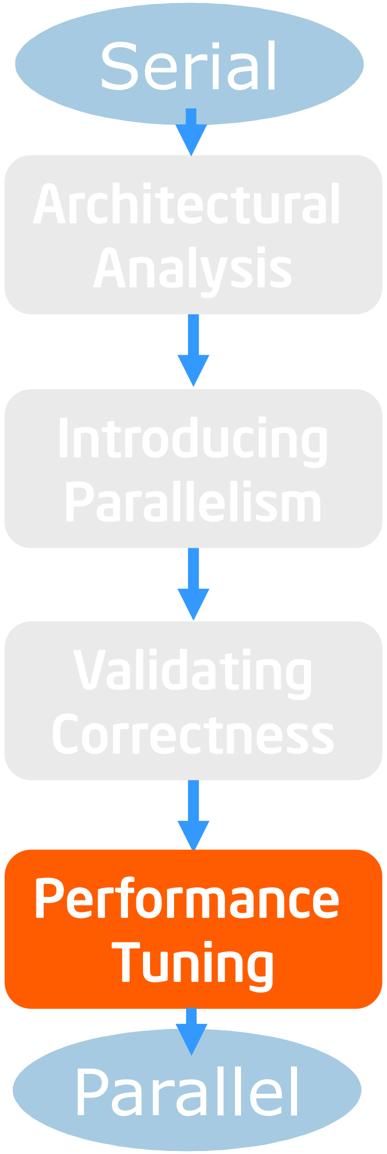
Problem Sets

ID		Problem	Sources	Modules	State
P1		Potential privacy infringement	PrimeOpenMP.cpp	05_Correct Threading Errors.exe	Not fixed
P2		Potential privacy infringement	PrimeOpenMP.cpp	05_Correct Threading Errors.exe	Not fixed
P3		Potential privacy infringement	PrimeOpenMP.cpp	05_Correct Threading Errors.exe	Not fixed

Number of Primes is Correct



Number of primes is correct



Step 4

Tune for best performance

Key Questions -Tune

S
1
2
3
4
D

Is the threading running efficiently?

Do my tasks do equal amounts of work?

Is my application scalable?

Performance Issues

Load Balancing

Synchronisation Overhead

Scalability

Difficult to examine without the right tools

A Reminder – Where are we?

```
Running 01_Converted To Intel.exe
100%

   78498 primes found between       0 and 1000000 in     2.28 secs
```

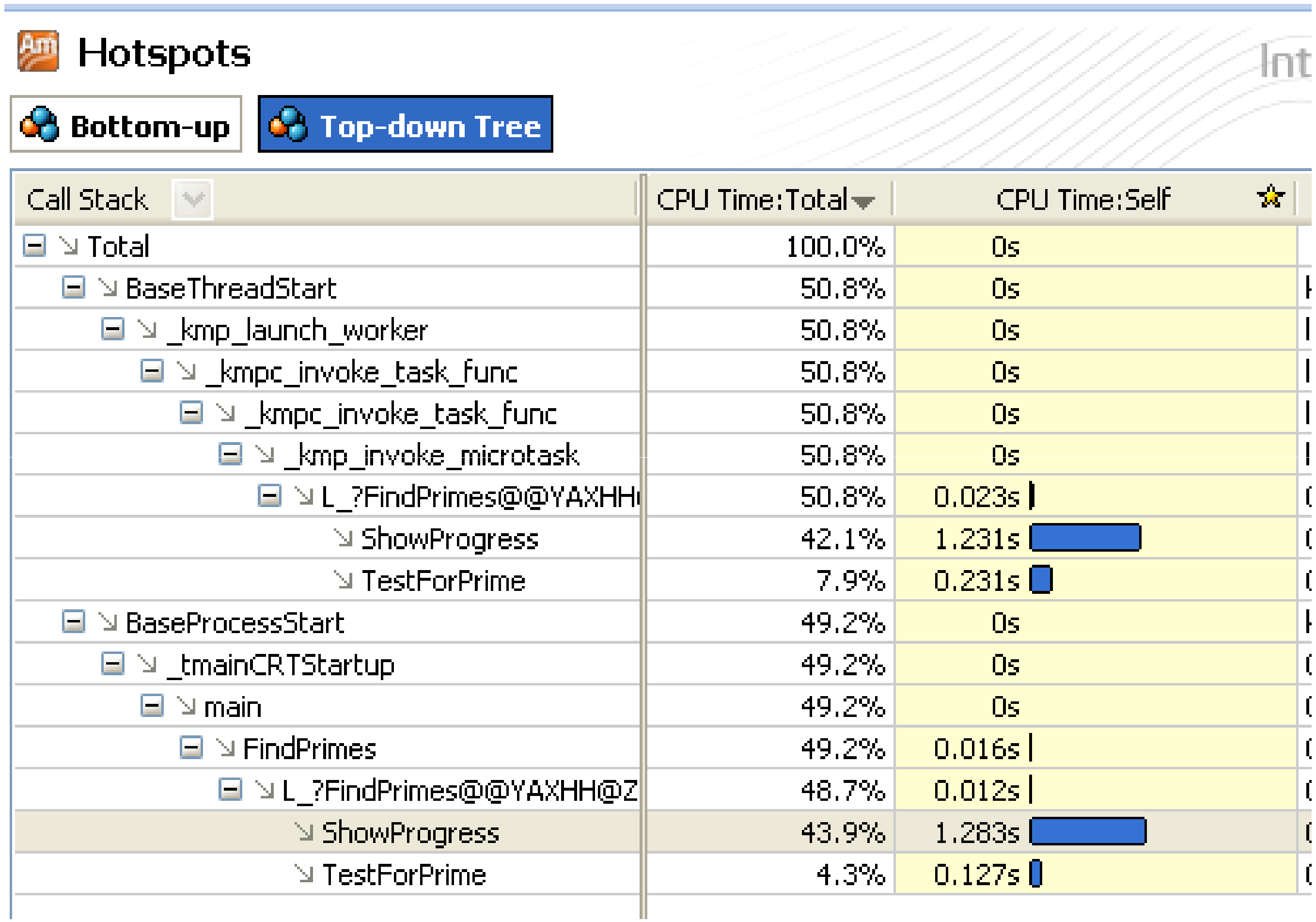
```
Running 06_Finding Performance Issues.exe
100%

   78498 primes found between       0 and 1000000 in     2.09 secs
```

Number of primes is correct

Almost as slow as the serial version

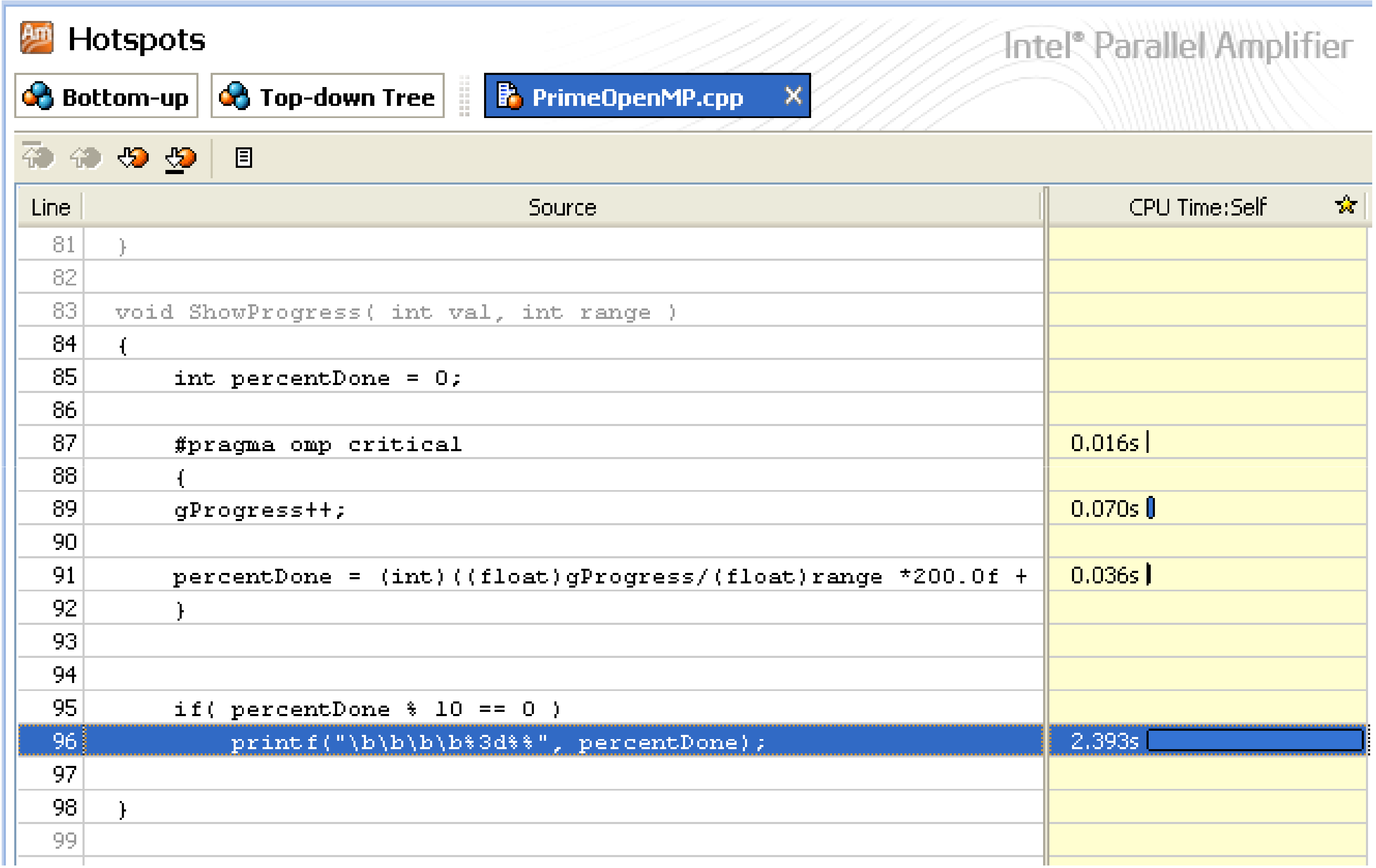
Demo 6 – Find the Threading Performance Issues

Hotspot Analysis



Call Stack	CPU Time:Total	CPU Time:Self
Total	100.0%	0s
BaseThreadStart	50.8%	0s
_kmp_launch_worker	50.8%	0s
_kmpc_invoke_task_func	50.8%	0s
_kmpc_invoke_task_func	50.8%	0s
_kmp_invoke_microtask	50.8%	0s
L_?FindPrimes@@YAXHH	50.8%	0.023s
ShowProgress	42.1%	1.231s
TestForPrime	7.9%	0.231s
BaseProcessStart	49.2%	0s
_tmainCRTStartup	49.2%	0s
main	49.2%	0s
FindPrimes	49.2%	0.016s
L_?FindPrimes@@YAXHH@Z	48.7%	0.012s
ShowProgress	43.9%	1.283s
TestForPrime	4.3%	0.127s

Source View


Hotspots
Intel® Parallel Amplifier
Bottom-up
Top-down Tree
PrimeOpenMP.cpp
Line
Source
CPU Time:Self
81 }
82
83 void ShowProgress(int val, int range)
84 {
85 int percentDone = 0;
86
87 #pragma omp critical
0.016s
88 {
89 gProgress++;
0.070s
90
91 percentDone = (int)((float)gProgress/(float)range *200.0f +
0.036s
92 }
93
94
95 if(percentDone % 10 == 0)
96 printf("\b\b\b\b%3d%%", percentDone);
2.393s
97
98 }
99

Improving the Performance

```
void ShowProgress( int val, int range )
{
    int percentDone;
    static int lastPercentDone = 0;
#pragma omp critical
    {
       gProgress++;
       percentDone = (int)((float)gProgress/(float)range*200.0f+0.5f);
    }
    if( percentDone % 10 == 0 && lastPercentDone < percentDone / 10){
        printf("\b\b\b\b%3d%%", percentDone);
        lastPercentDone++;
    }
}
```

The algorithm has many more updates than the 10 needed for showing progress

This change should fix the contention issue

Demo 7 – Fixing the Synchronisation issues

Superb Speedup … ???

```
Running 01_Converted To Intel.exe
100%

   78498 primes found between       0 and 1000000 in     2.28 secs
```

```
Running 07_Reducing Printfs.exe
100%

   78498 primes found between       0 and 1000000 in     0.31 secs
```

Speedup 7.36

On a dual core?

Demo 8 – Getting New Serial Benchmark

That’s better (but disappointing)…

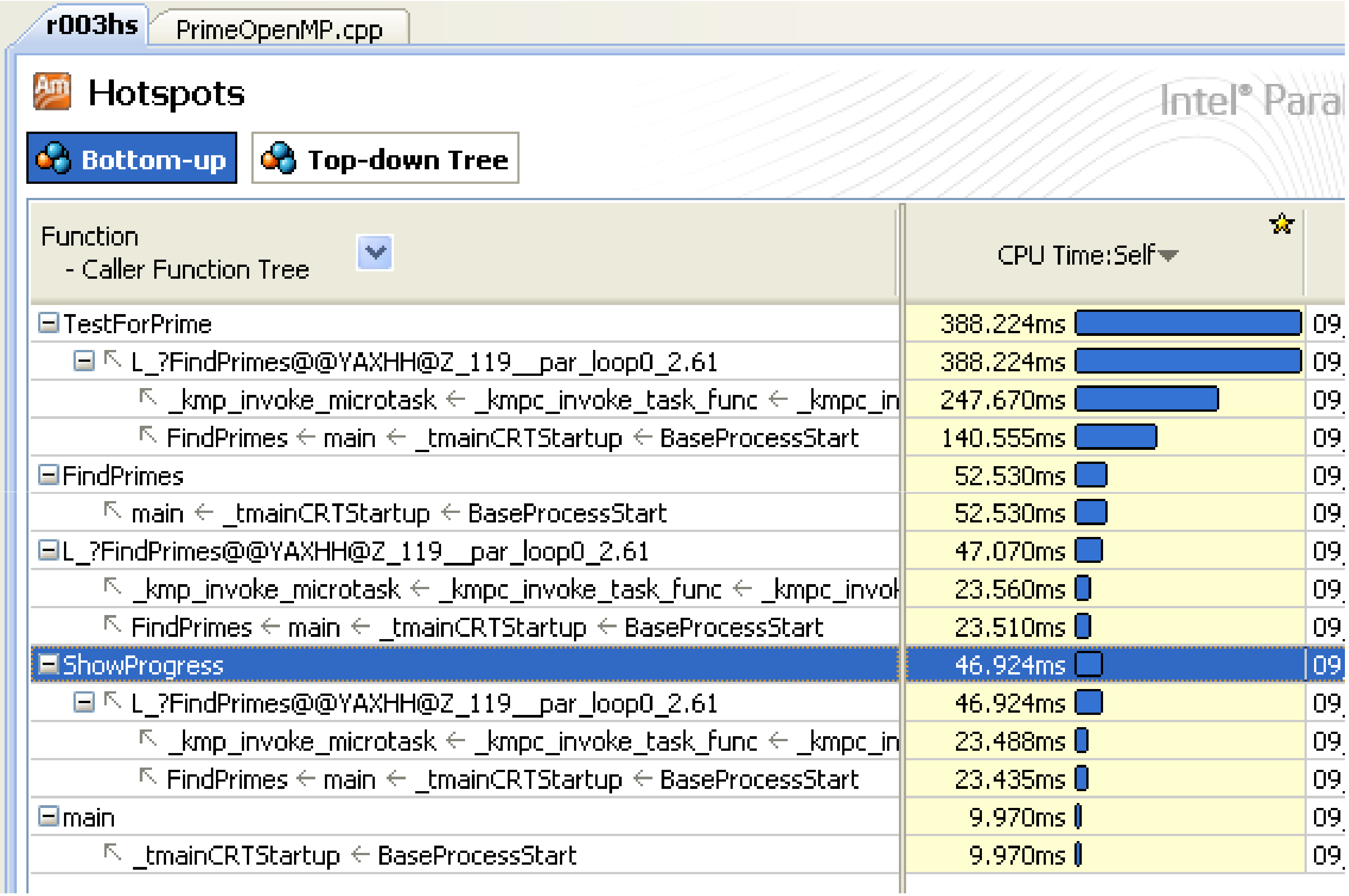
```
Running 08_New Serial Benchmark.exe
100%

   78498 primes found between       0 and 1000000 in     0.48 secs
```

```
Running 07_Reducing Printfs.exe
100%

   78498 primes found between       0 and 1000000 in     0.31 secs
```

Speedup 1.55

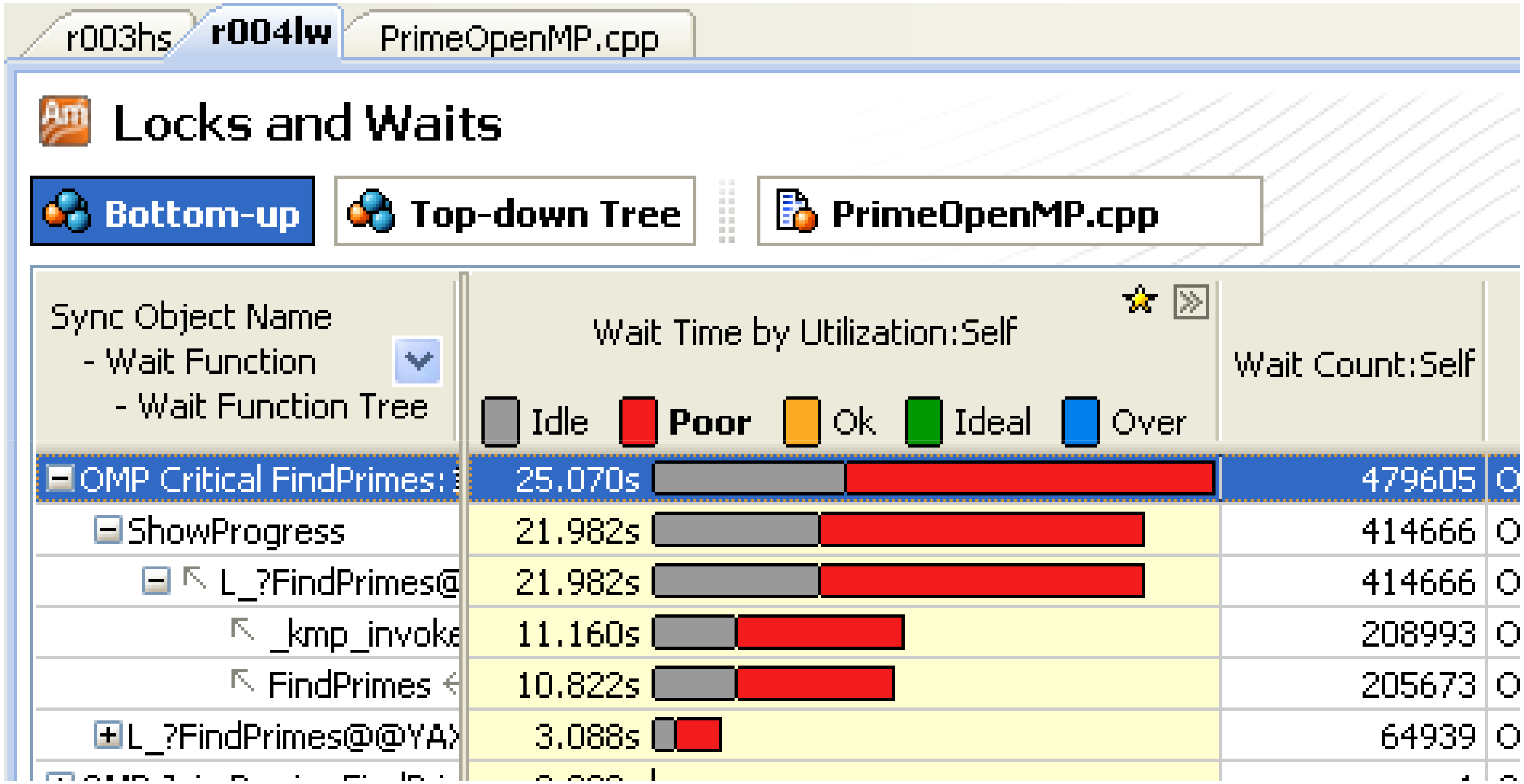
Demo 9 – Correcting the Synchronisation Issue

Hotspots



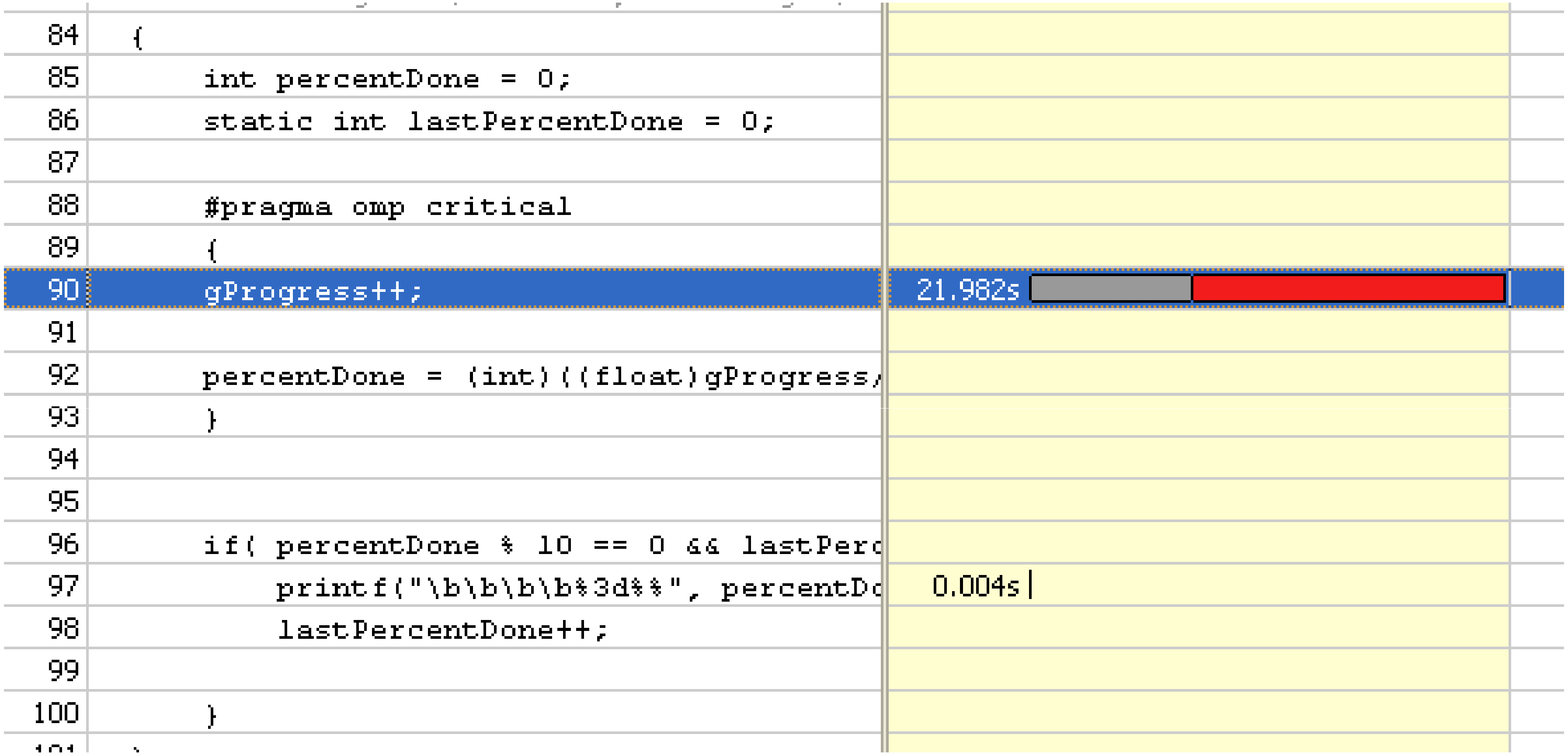
Function - Caller Function Tree	CPU Time:Self	
⊟ TestForPrime	388.224ms	09
⊟ L_?FindPrimes@@YAXHH@Z_119__par_loop0_2.61	388.224ms	09
_kmp_invoke_microtask ← _kmpc_invoke_task_func ← _kmpc_in	247.670ms	09
FindPrimes ← main ← _tmainCRTStartup ← BaseProcessStart	140.555ms	09
⊟ FindPrimes	52.530ms	09
main ← _tmainCRTStartup ← BaseProcessStart	52.530ms	09
⊟ L_?FindPrimes@@YAXHH@Z_119__par_loop0_2.61	47.070ms	09
_kmp_invoke_microtask ← _kmpc_invoke_task_func ← _kmpc_invok	23.560ms	09
FindPrimes ← main ← _tmainCRTStartup ← BaseProcessStart	23.510ms	09
⊟ ShowProgress	46.924ms	09
⊟ L_?FindPrimes@@YAXHH@Z_119__par_loop0_2.61	46.924ms	09
_kmp_invoke_microtask ← _kmpc_invoke_task_func ← _kmpc_in	23.488ms	09
FindPrimes ← main ← _tmainCRTStartup ← BaseProcessStart	23.435ms	09
⊟ main	9.970ms	09
_tmainCRTStartup ← BaseProcessStart	9.970ms	09

Locks & Waits



r003hs
r004lw
PrimeOpenMP.cpp
Locks and Waits
Bottom-up
Top-down Tree
PrimeOpenMP.cpp
Sync Object Name
- Wait Function
- Wait Function Tree
Wait Time by Utilization:Self
Idle
Poor
Ok
Ideal
Over
Wait Count:Self
OMP Critical FindPrimes:
25.070s
479605
ShowProgress
21.982s
414666
L_?FindPrimes@
21.982s
414666
_kmp_invoke
11.160s
208993
FindPrimes
10.822s
205673
L_?FindPrimes@@YA
3.088s
64939

Source Code View



Fixing the synchronisation issue -1

This fix removes the need for a critical section

```
void FindPrimes(int start, int end)
{
    // start is always odd
    int range = end - start + 1;

#pragma omp parallel for
    for( int i = start; i <= end; i += 2 )
    {
        if( TestForPrime(i) )
            globalPrimes[InterlockedIncrement(&gPrimesFound)] = i;

        ShowProgress(i, range);
    }
}
```

Fixing the synchronisation issue - 2

This fix removes the need for a critical section

```
void ShowProgress( int val, int range )
{
    long percentDone, localProgress;
    static int lastPercentDone = 0;

    localProgress = InterlockedIncrement(&gProgress);
    percentDone = (int)((float)localProgress/
       (float)range*200.0f+0.5f);

    if( percentDone % 10 == 0 && lastPercentDone < percentDone / 10)
        printf("\b\b\b\b%3d%%", percentDone);
        lastPercentDone++;
    }
}
```

That’s better (but still disappointing)…

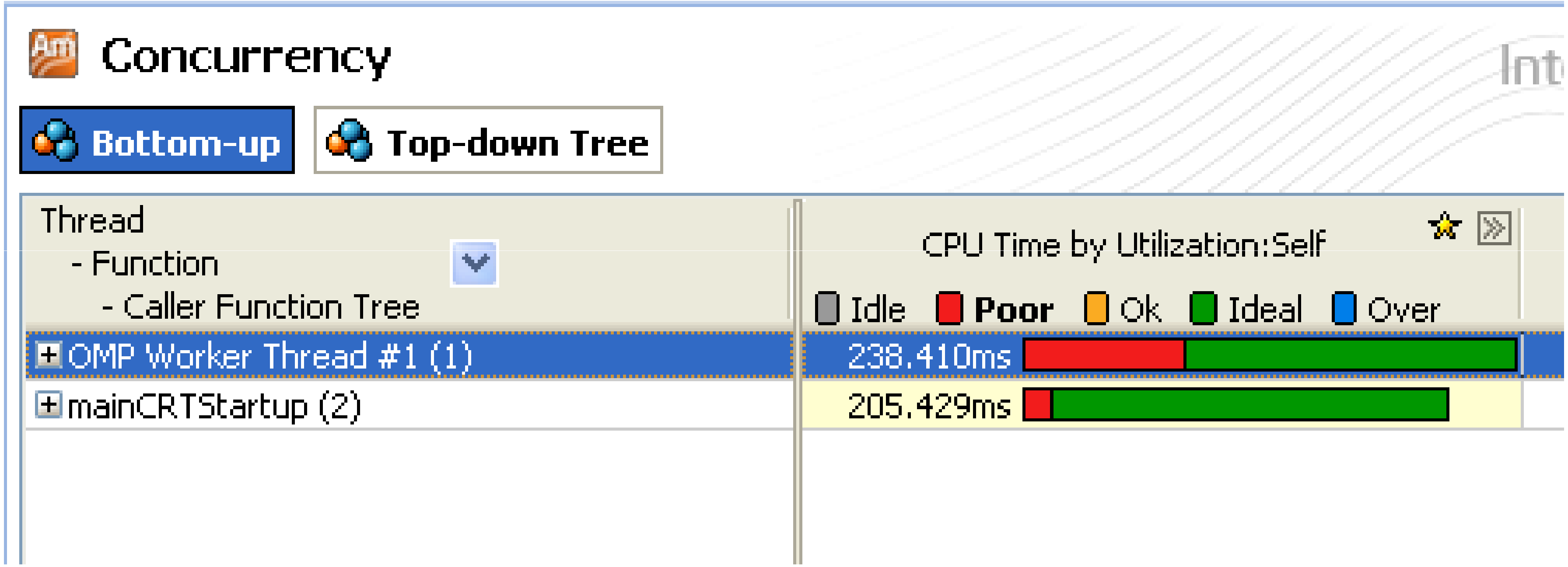
```
Running 08_New Serial Benchmark.exe
100%

   78498 primes found between       0 and 1000000 in     0.48 secs
```

```
Running 10_Fixing Synchronisation Issue.exe
100%

   78498 primes found between       0 and 1000000 in     0.30 secs
```

Speedup 1.6

Demo 9 – Improving the Load Balancing

Threads are not doing equal work



Fixing a Load Imbalance

Distribute the work more evenly

```
void FindPrimes(int start, int end)
{
    // start is always odd
    int range = end - start + 1;

#pragma omp parallel for schedule(static, 8)
    for( int i = start; i <= end; i += 2 )
    {
       if( TestForPrime(i) )
          globalPrimes[InterlockedIncrement(&gPrimesFound)] = i;
       ShowProgress(i, range);
    }
}
```

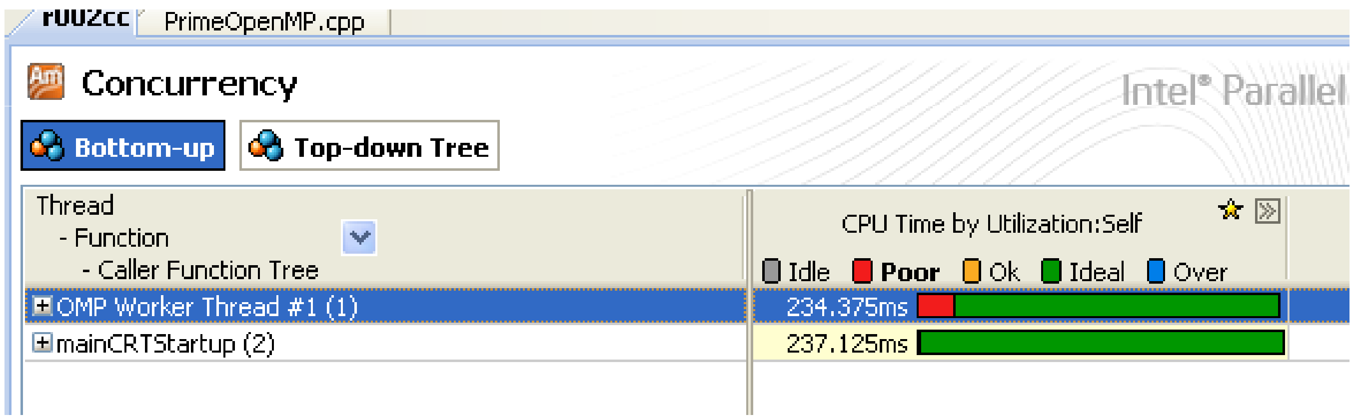
That’s better

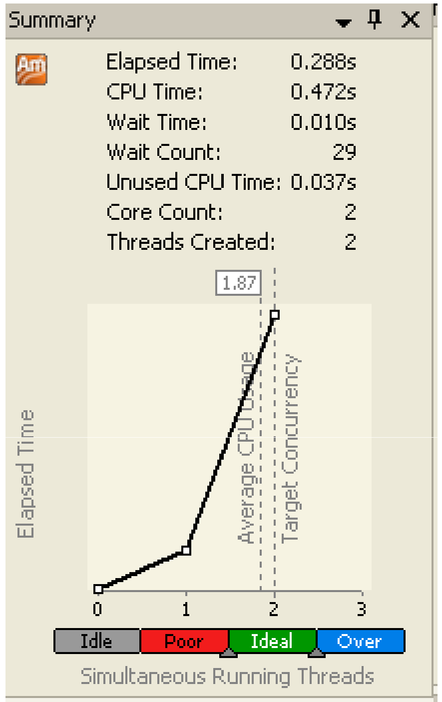
```
Running 08_New Serial Benchmark.exe
100%

   78498 primes found between       0 and 1000000 in    0.48 secs
```

```
Running 11_Load Balancing.exe
100%

   78498 primes found between       0 and 1000000 in    0.25 secs
```

Speedup 1.92

A Finely Balanced threaded Program!





Scalability

Key Questions -Tune

Is the threading running efficiently?

Do my tasks do equal amounts of work?

Is my application scalable?

Moving to Parallel – a view from some developers

Top 5 challenges

- Legacy
- Education
- Tools
- Fear of many cores
- Maintainability

Scalability http://paralleluniverse.intel.com

Intel® Parallel Universe Portal

1: Describe your application | 2: Configure | **3: Summary**

Step 3: Summary

Session Name	prime
Upload zip file	11_Load Balancing.zip
Command Line	11_Load Balancing.exe 1 1000000

Back Submit

Click on DONE to get the reports of the execution.

The Results

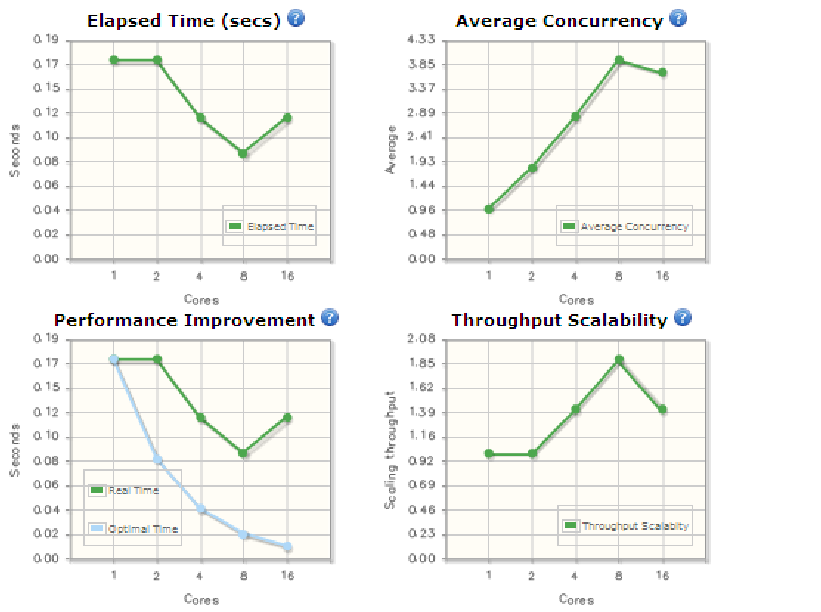
Command line: 11_Load Balancing.exe 1 1000000

Report: Download Intel® Parallel Amplifier report

Enterprise x64 Edition
Service Pack 2
Intel® Xeon® processor, CPU 2.80 GHz
Hyperthreaded 8-Core
12 GB RAM

Cores	Elapsed Time (secs)	Number of Executions
1	0.17	3
2	0.17	3
4	0.12	3
8	0.09	3
16	0.12	3

Cores	Average Concurrency	Number of Executions
1	1.00	3
2	1.81	3
4	2.81	3
8	3.94	3
16	3.70	3



Your job has completed!

parallel.universe@intel.com

Sent: Mon 07/06/2010 06:05

To: Blair-chappell, Stephen

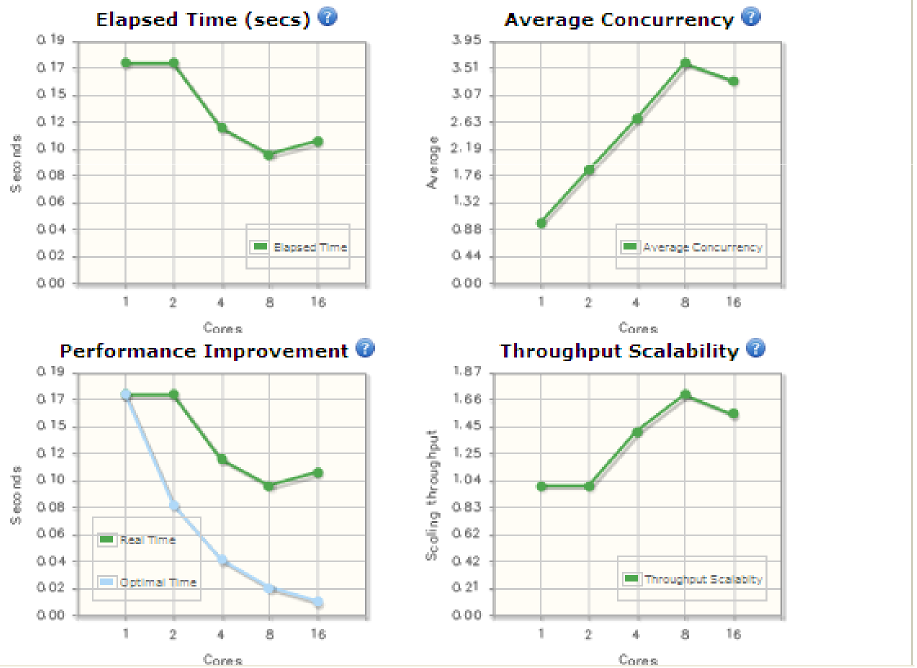
Your job has completed!

We're sending you this email because your job "prime" has successfully completed
Please click the link below to access to the results page
http://paralleluniverse.intel.com/Job/Summary/878388be-a821-46f9-a911-0592309db43b
Thanks for using Intel® Parallel Universe Portal!

Without the printfs

Cores	Elapsed Time (secs)	Number of Executions
1	0.17	3
2	0.17	3
4	0.12	3
8	0.10	3
16	0.11	3

Cores	Average Concurrency	Number of Executions
1	1.00	3
2	1.85	3
4	2.69	3
8	3.59	3
16	3.31	3



A run of 10 million

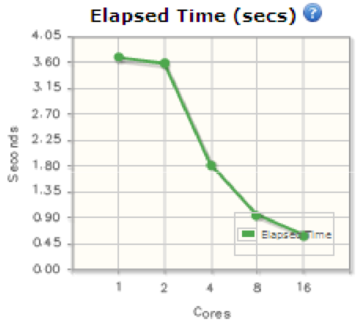
Command line: 11_Load Balancing.exe 1 10000000

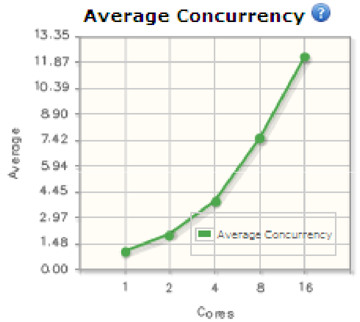
Report: Download Intel® Parallel Amplifier report

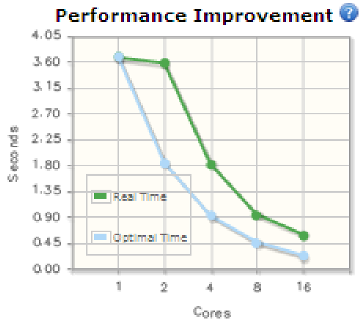
Service Pack 2
Intel® Xeon® processor, CPU 2.80 GHz
Hyperthreaded 8-Core
12 GB RAM

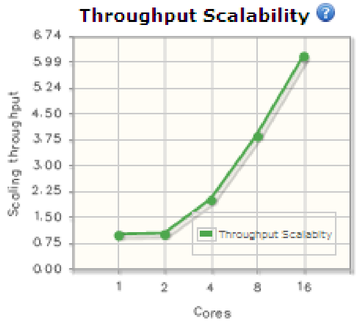
Cores	Elapsed Time (secs)	Number of Executions
1	3.68	3
2	3.58	3
4	1.83	3
8	0.95	3
16	0.60	3

Cores	Average Concurrency	Number of Executions
1	1.00	3
2	1.98	3
4	3.88	3
8	7.49	3
16	12.14	3

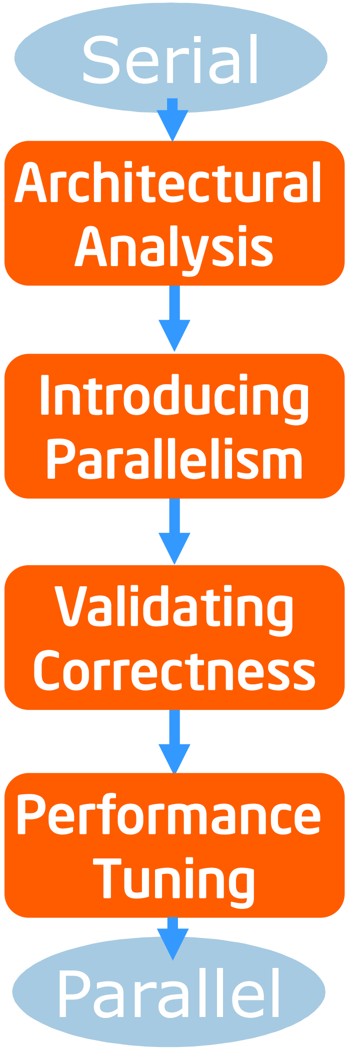








Tools



Existing Intel Software	Intel Parallel Studio
Intel® VTune™ Performance Analyzer	Advisor/Amplifier
Intel Compilers Parallel Libraries	Composer
Intel® Thread Checker	Inspector
Intel® Thread Profiler	Amplifier

thank you

intel.com / go / parallel

Q&A

thank you