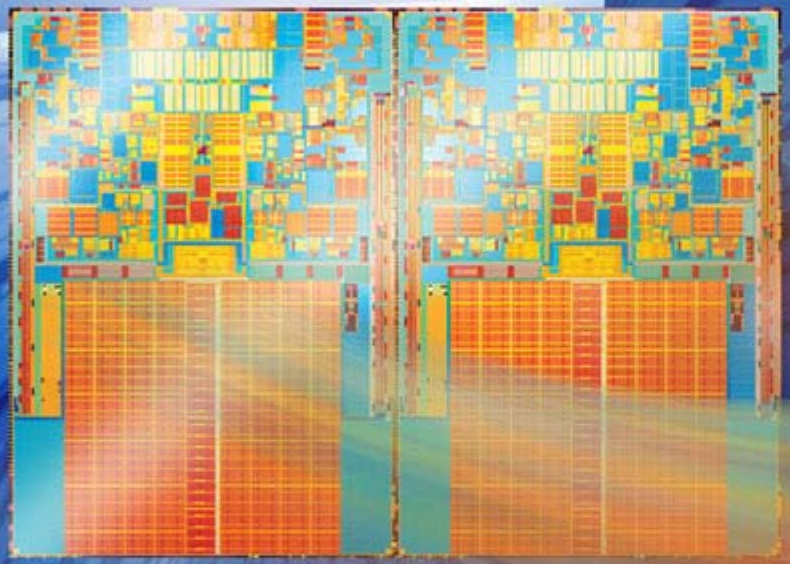
Parallel Programming 2.0



Wei Li
Senior Principal Engineer
Director, Emerging Products Lab

Santa Clara, California
USA

Quiz

How old is Intel?

- 10 years
- 20 years
- 30 years
- 40 years

Please look for the Celebration Ball at the Intel booth to win a prize!

Agenda

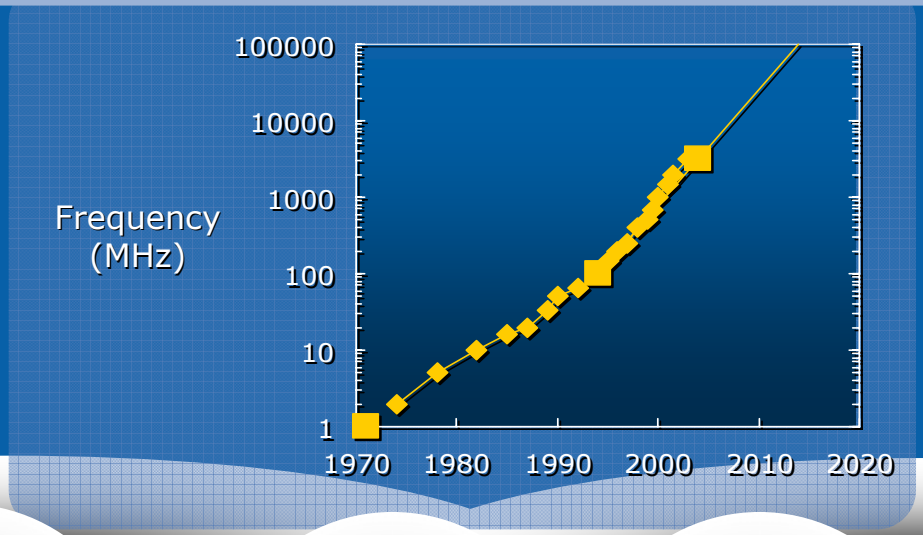
Why Parallel Programming 2.0?

A Case Study

Intel Software Products

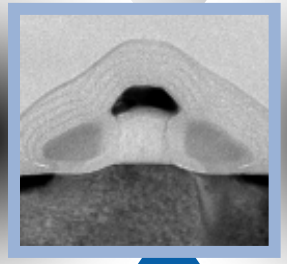
Historical Driving Force

Increased Performance via Increased Frequency

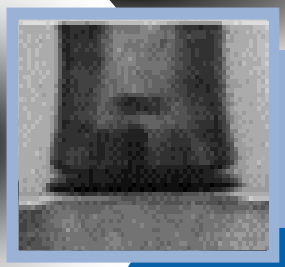


1946
20 Numbers in Main Memory

1971
I4004 Processor 2300 Transistors



2005
65nm 1B+ Transistors

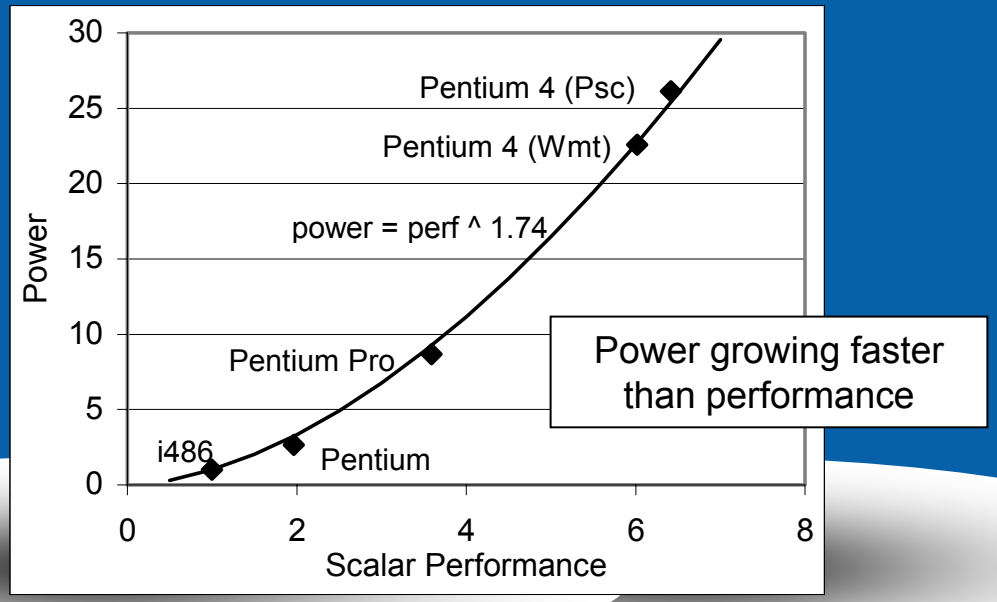


2007
45nm

2009
32nm

The Challenge

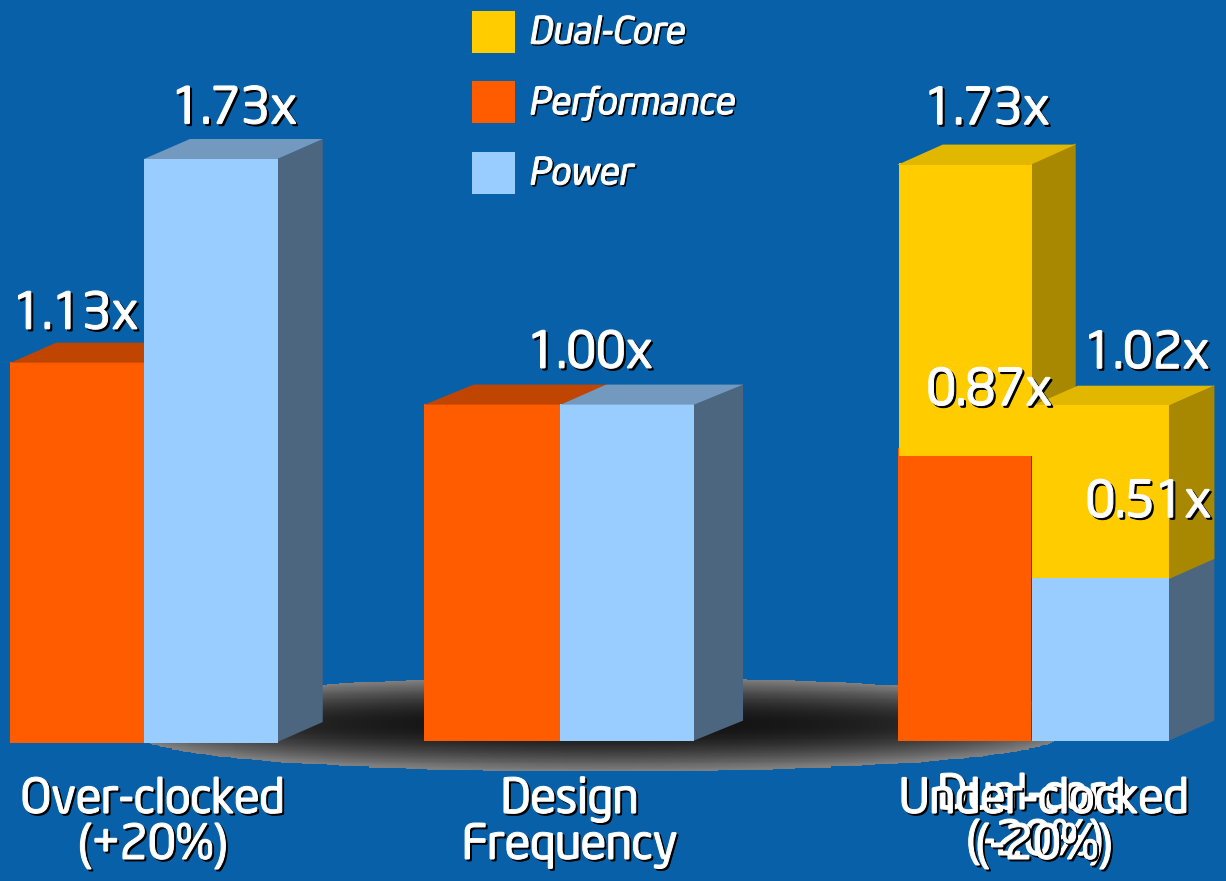
Power Limitations



Power and performance normalized to i486

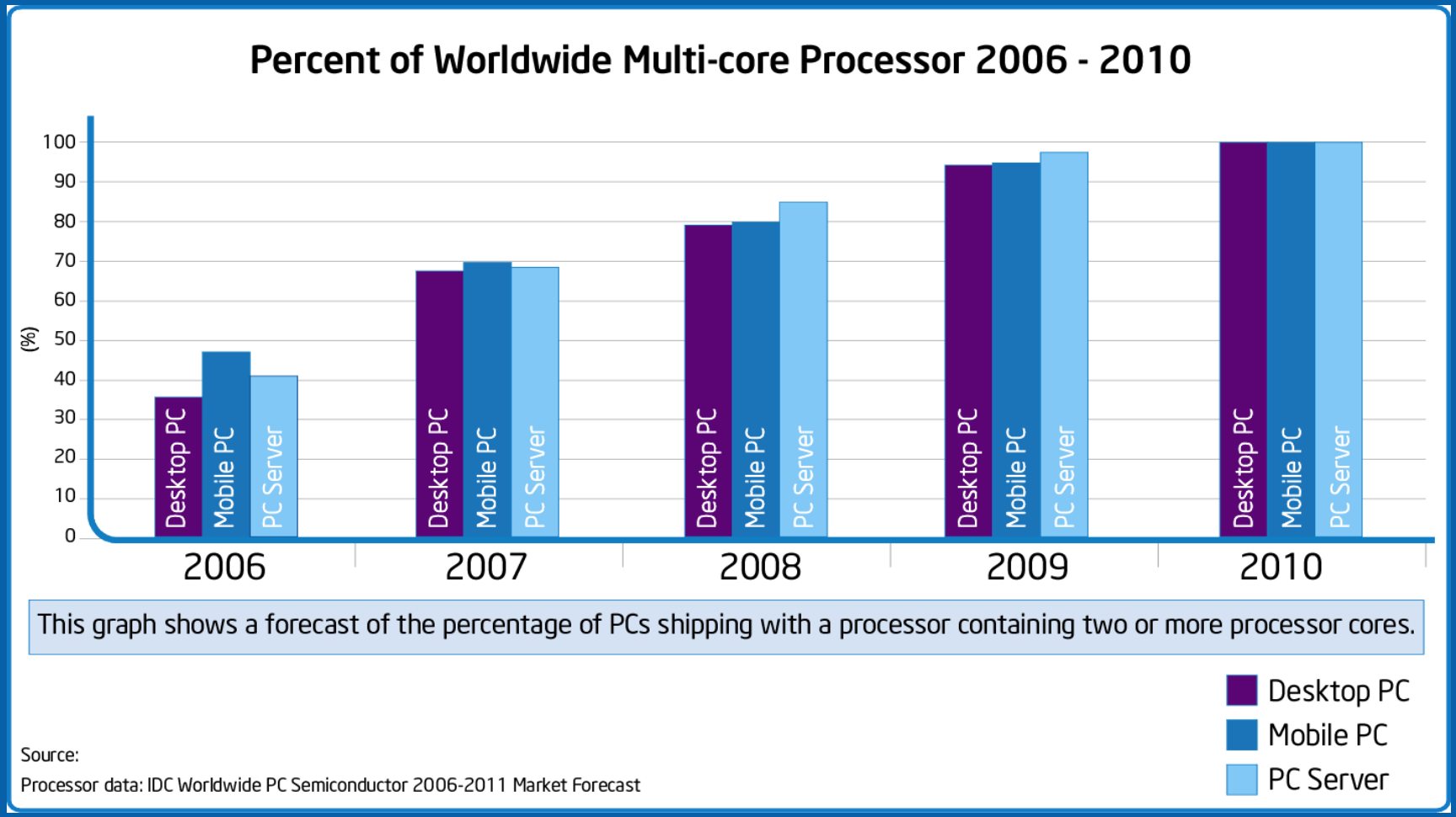
Unsustainable Power Growth

MULTI-CORE MOTIVATION



Relative single-core frequency and Vcc

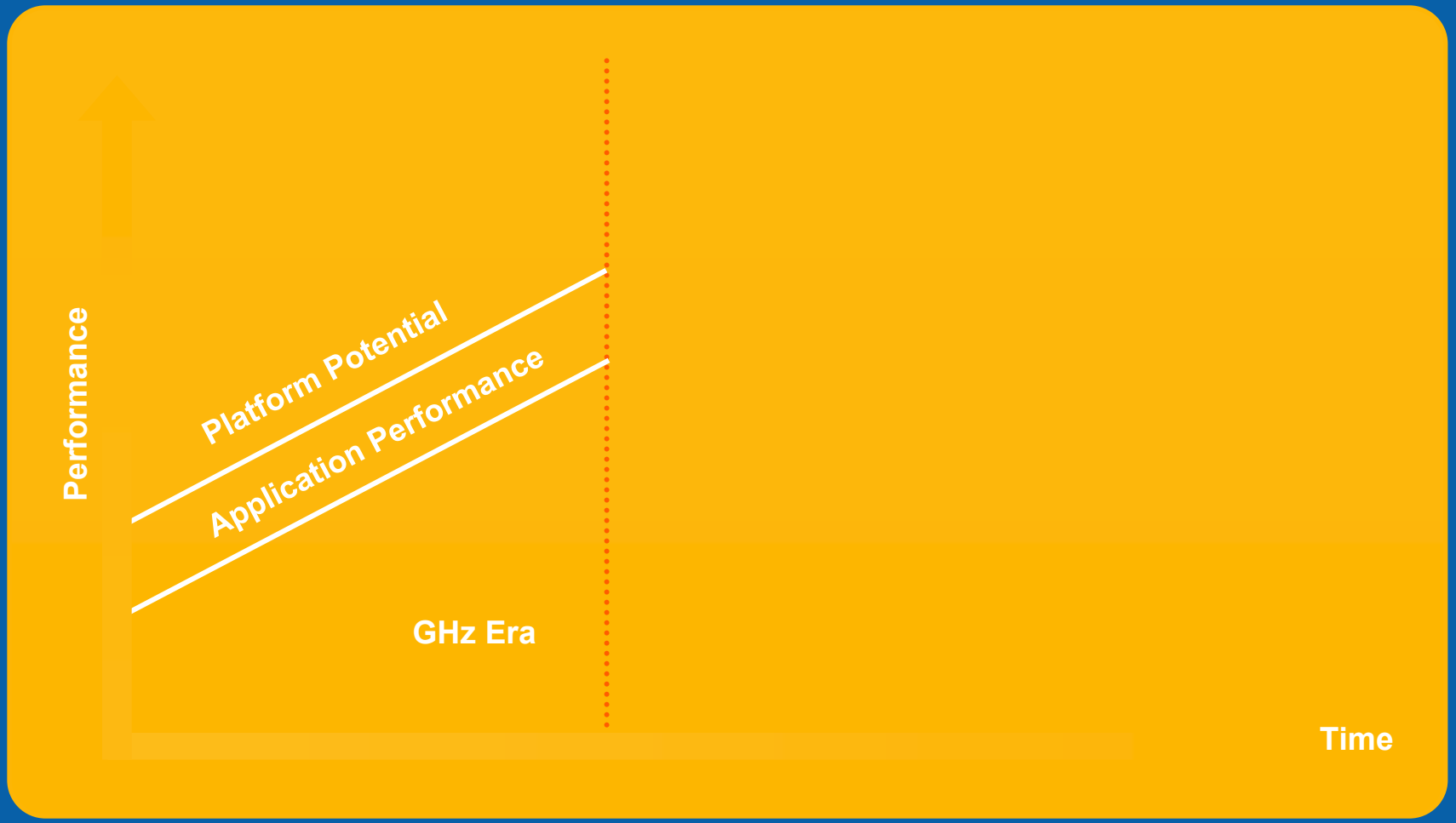
Approaching 100% of processors are parallel processors



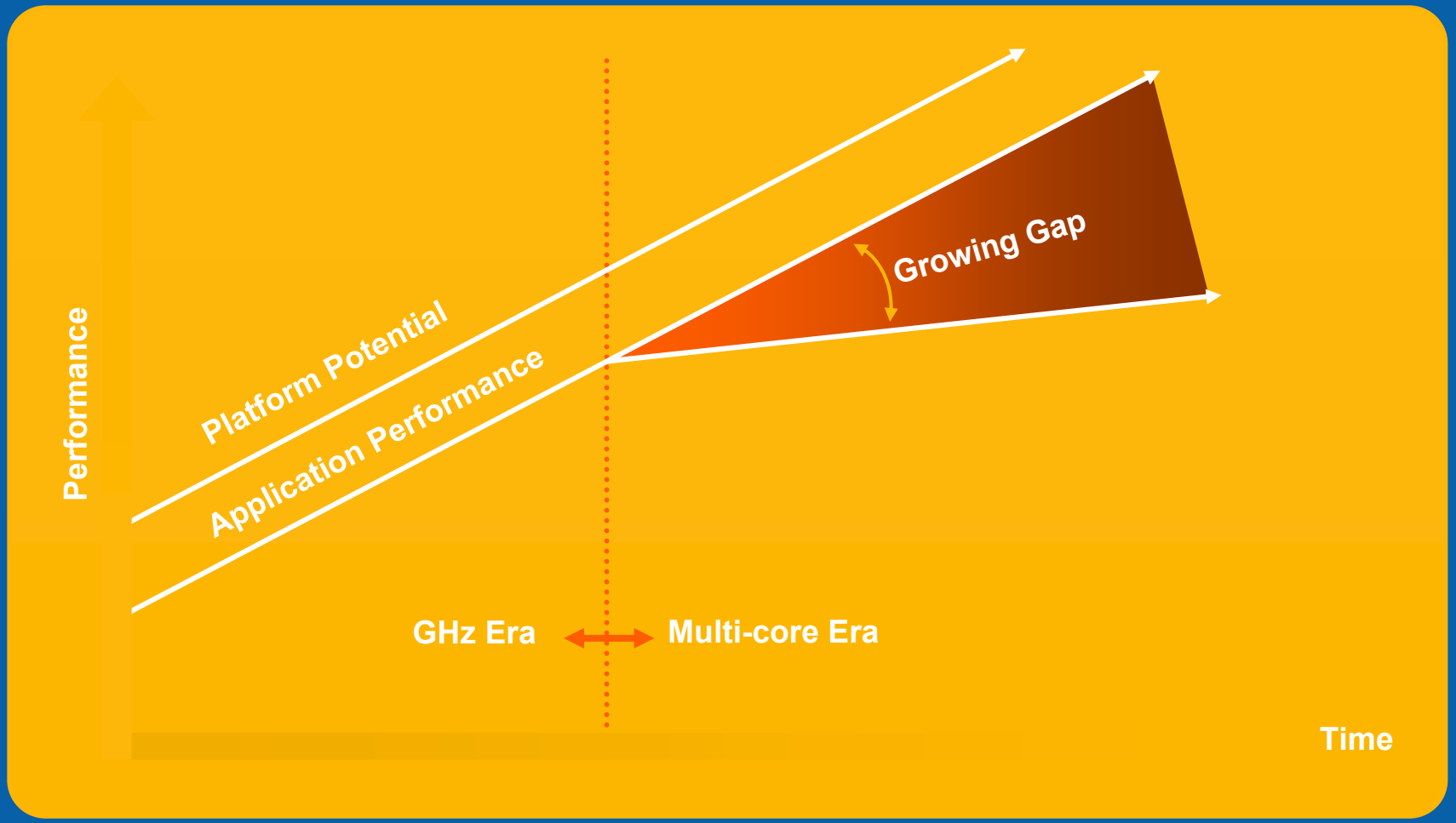
Multi-core is performance delivered in a new way.

Our job is to make sure the software industry makes the most of that performance.

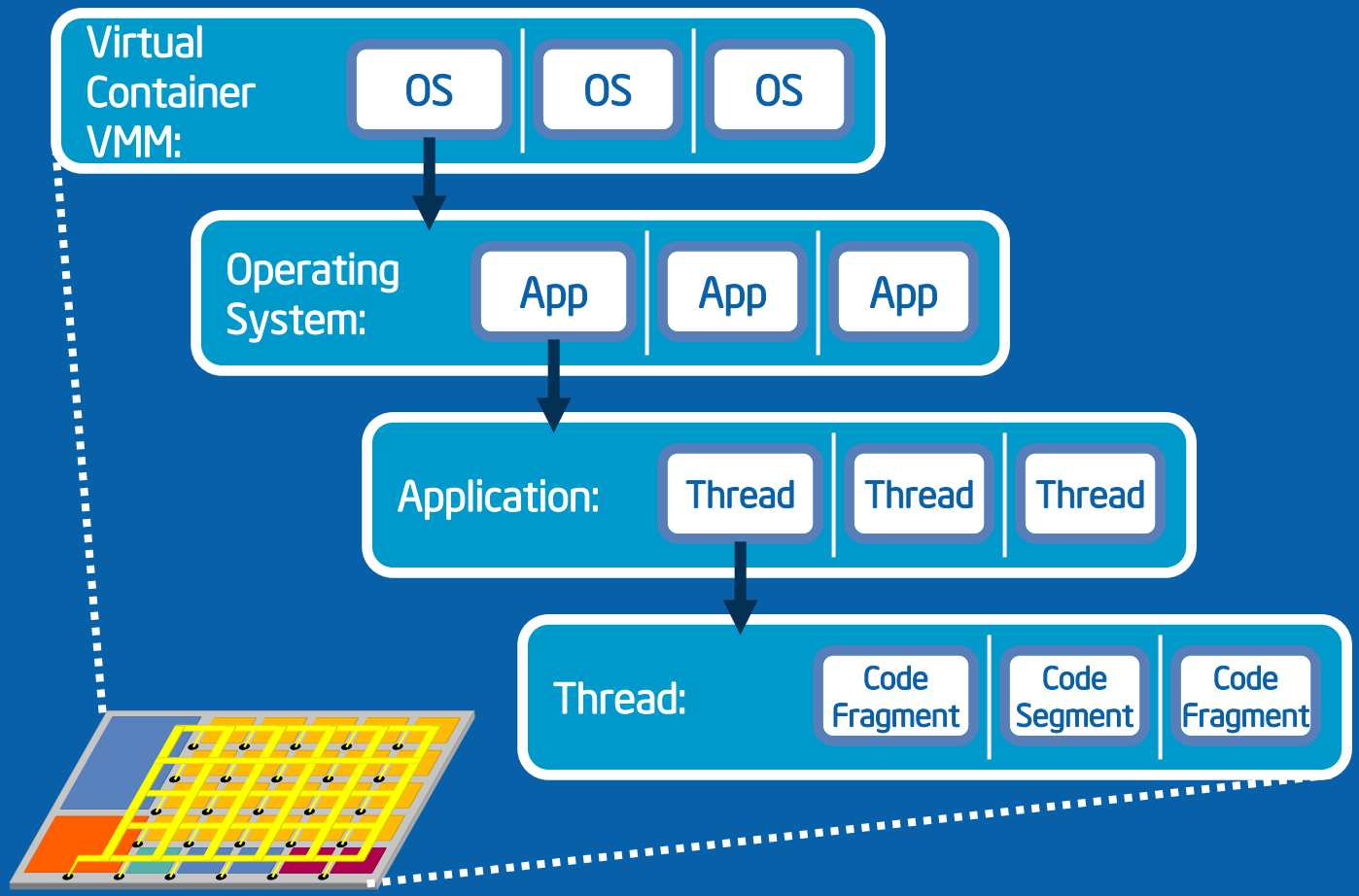
The Burden on Software



The Burden on Software



Parallelism at many levels



Parallel Programming 1.0

- HPC applications for peak performance
- Manual, lots of hand tuning by experts
  - Difficult, often not possible without specific tools
  - Does not scale, need to re-do for each app

Goals for Parallel Programming 2.0

- Mainstream applications, not peak performance
- High productivity programming
  - Raise the level of programming abstraction – easy to learn and parallelize
  - Make tools easy to use – Ph.D. not required
  - Bring parallelism to mainstream programming – undergraduate-level

Agenda

Why Parallel Programming 2.0?

A Case Study

Intel Software Products

Example: Threading the Compiler

main.c

```
Compiler {

  Read options

  For Each line
    parse
    update Tables
  End

  For Each Function
    Optimize_func()
  End
}
```

control.c

```
Optimize_func {
    Count loops


    Global_opts
    Translate
    Local_opts
    Allocate_regs
    Generate_obj
}
```

opt_gen.c

```
Global_opts {

     For Each Block
       For Each Inst
        If(  )
       End
     End

}
Generate_obj {
   Generate_mem
   For Each Block
     For Each Inst
       encode
       write
     End
    End
}
```

Parallelization

```
Compiler {

  Read options

  For Each line
    parse
    update Tables
  End
  //parallel
  For Each Function
    Optimize_func()
  End
}
```

```
Optimize_func {
    Count loops


    Global_opts
    Translate
    Local_opts
    Allocate_regs
    Generate_obj
}
```

```
Global_opts {

     For Each Block
       For Each Inst
        If(  )
       End
     End

}
Generate_obj {
   Generate_mem
   For Each Block
     For Each Inst
       encode
       write
     End
    End
}
```

Global Variables

```
Compiler {
  threshold=false
  Read options

  For Each line
    parse
    update Tables
  End
  //parallel
  For Each Function
    Optimize_func()
  End
}
```

```
Optimize_func {
    Count loops
    if(loops>100)
     threshold=True
    Global_opts
    Translate
    Local_opts
    Allocate_regs
    Generate_obj
}
```

```
Global_opts {
  if !threshold{
    For Each Block
      For Each Inst
       If(  )
      End
    End
  }
}
Generate_obj {
  Generate_mem
  For Each Block
    For Each Inst
      encode
      write
    End
   End
}
```

Characteristics

- Parallelizable code spread across ~100 modules and ~100 thousand lines of code
- Global variables
  - 3787 global symbols!!
  - Large number of global variables written in loop
- Serial portion
  - asm and object generation

Dealing with Globals

- Identify globals without cross iteration dependence
  - Only read in loop
  - Privatizable
- Identify globals with cross iteration dependence
  - Reduction for counters, timers, statistics
- Globals requiring synchronization
  - I/O

Agenda

Why Parallel Programming 2.0?

A Case Study

Intel Software Products

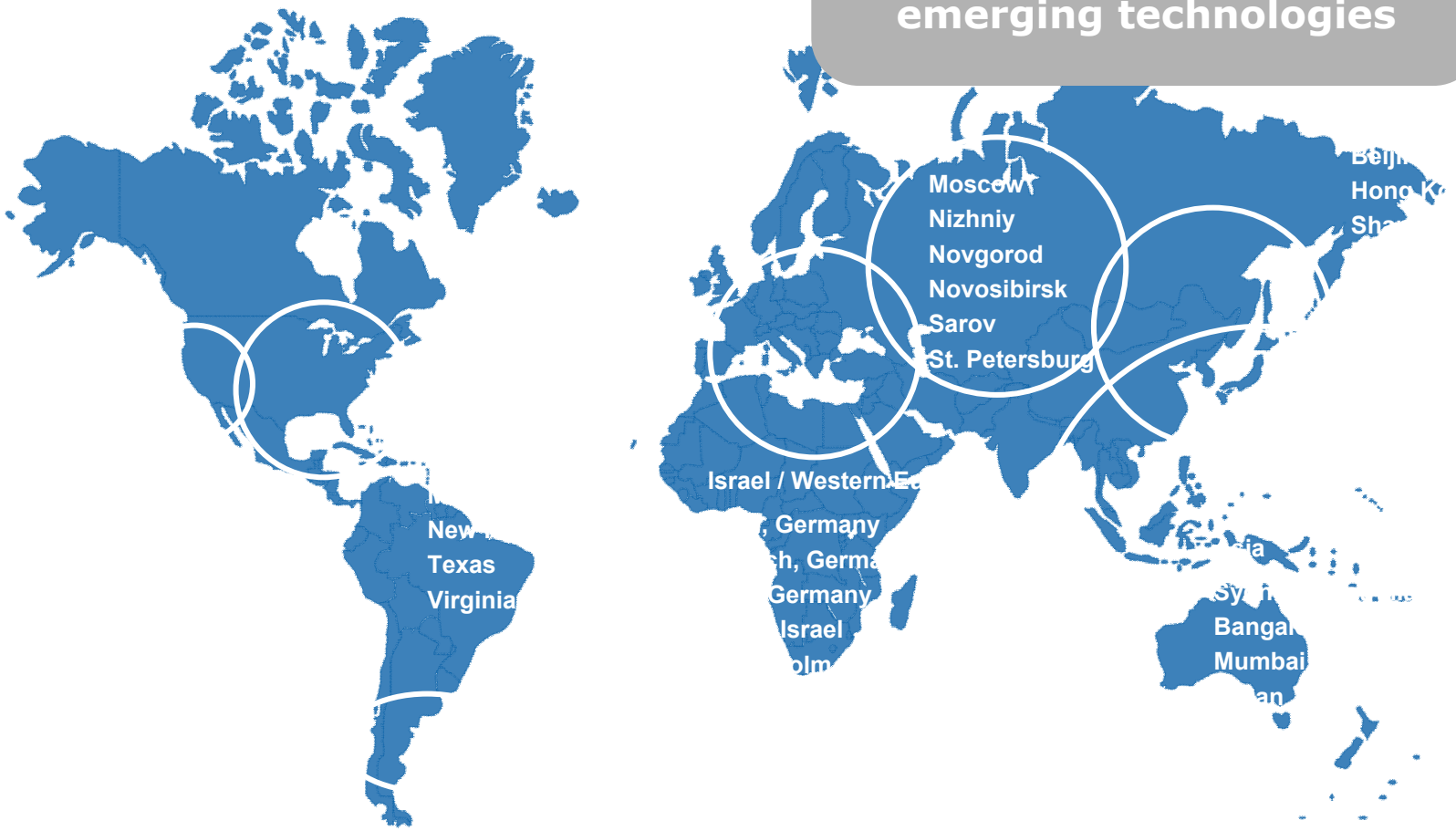
Software @ Intel

Ensure Intel Architecture is the platform of choice by:

- Software ecosystem co-development & enabling
- Leadership developer products
- Development of Intel platform software & services

Intel SW sites

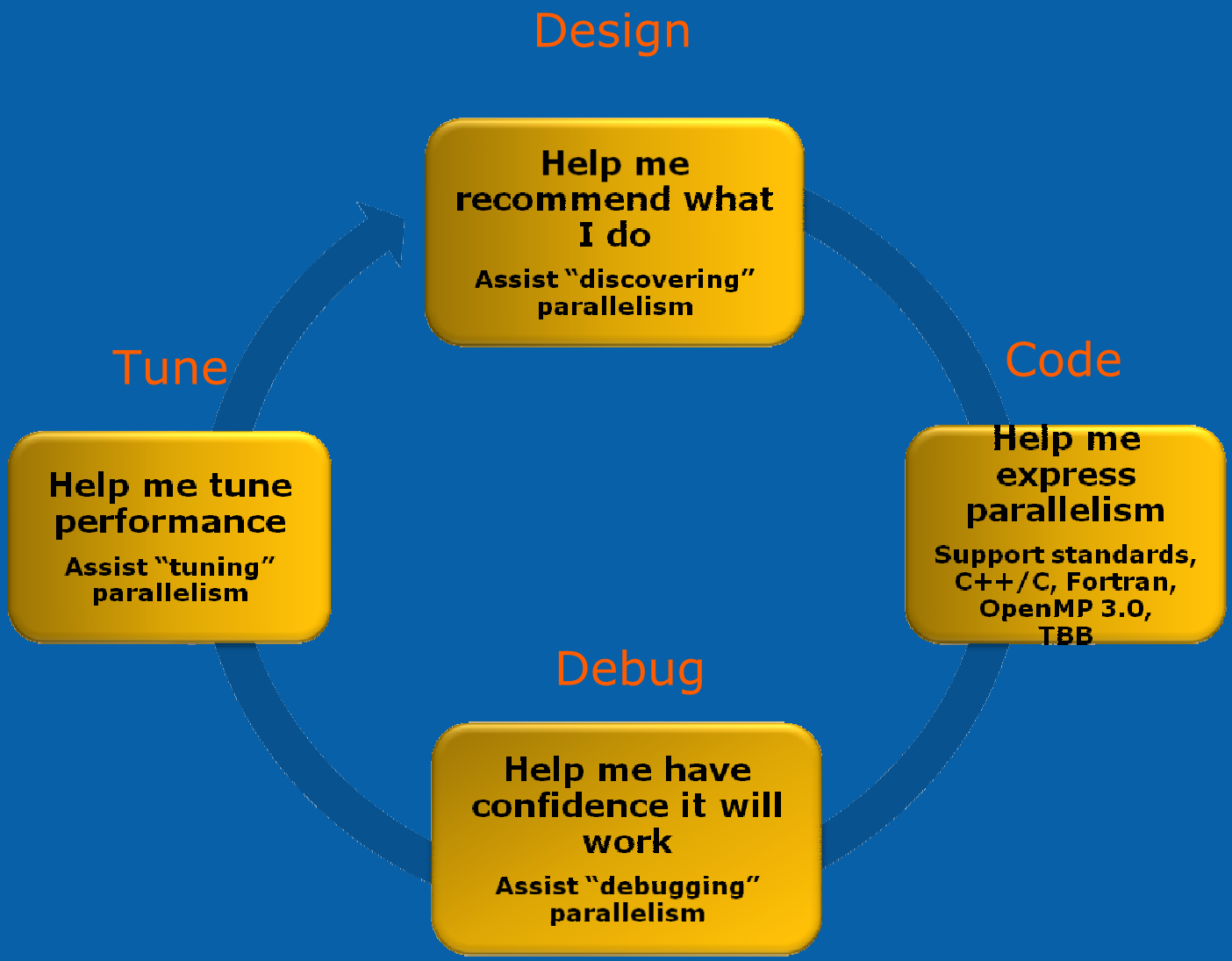
Our global presence helps us keep a pulse on developing markets and emerging technologies



Development Across Environments

Compilers | VTune™ Analyzers | Performance Libraries | Threading Tools | Cluster Tools | Digital Home Tools | Mobile Tools

Parallelization Methodology



Development with Intel® Tools

Architectural Analysis | **Introduce Threads** | **Confidence/Correctness** | **Optimize / Tune**

- **VTune™ Analyzer**
  - Find the code that can benefit from threading
  - Find hotspots that limit performance
- **Compilers**
  - Built-in optimization
  - OpenMP
- **Libraries**
  - Media
  - Math Processing
  - Threading
  - XML
- **Intel® Thread Checker**
  - Find deadlocks and race conditions
- **VTune Analyzer**
  - Tune for performance and scalability
- **Intel® Thread Profiler**
  - Visualize efficiency of threaded code

Unstructured Windows Threads: too low level

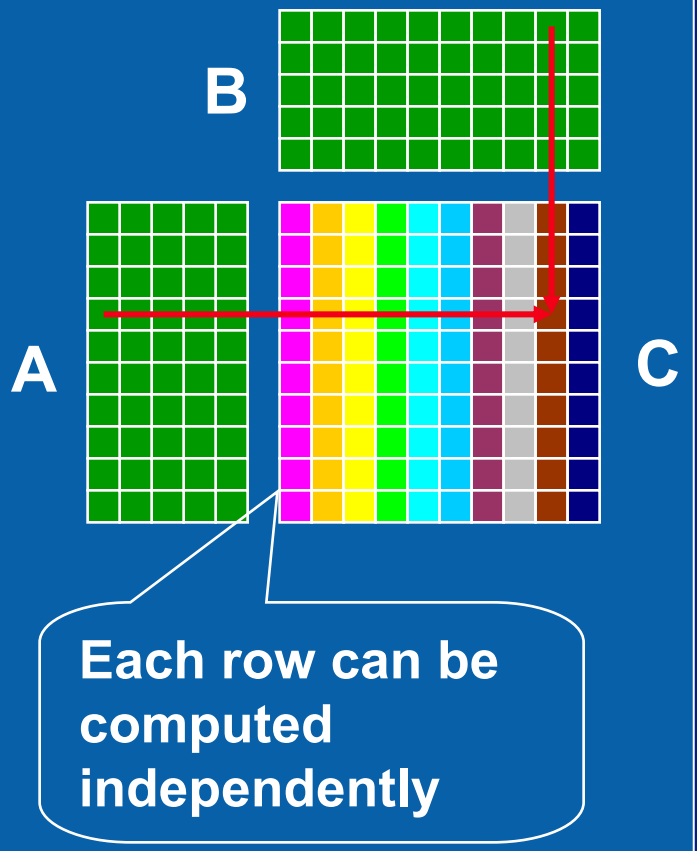
```
#include <iostream>
#include <windows.h>
using namespace std;
const int numThreads = 4;

DWORD WINAPI HelloFunc (LPVOID arg)
{
   cout << "Hello Thread\n";
   return 0;
}
main ()
{
   HANDLE hThread[numThreads];

   for (int i = 0; i < numThreads; i++)
      hThread[i] =
         CreateThread (NULL, 0, HelloFunc, NULL, 0, NULL );

   WaitForMultipleObjects (numThreads, hThread, TRUE, INFINITE);
}
```

Example: OpenMP Matrix Multiply




```
#pragma omp parallel for shared(C)
  private(i,j)
for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
        C[i][j] = 0.0;

#pragma omp parallel for
  shared(A,B,C) private(i,j,k)
for (i = 0; i < M; i++)
    for (k = 0; k < L; k++)
        for (j = 0; j < N; j++)
            C[i][j] +=
                    A[i][k] * B[k][j];
```

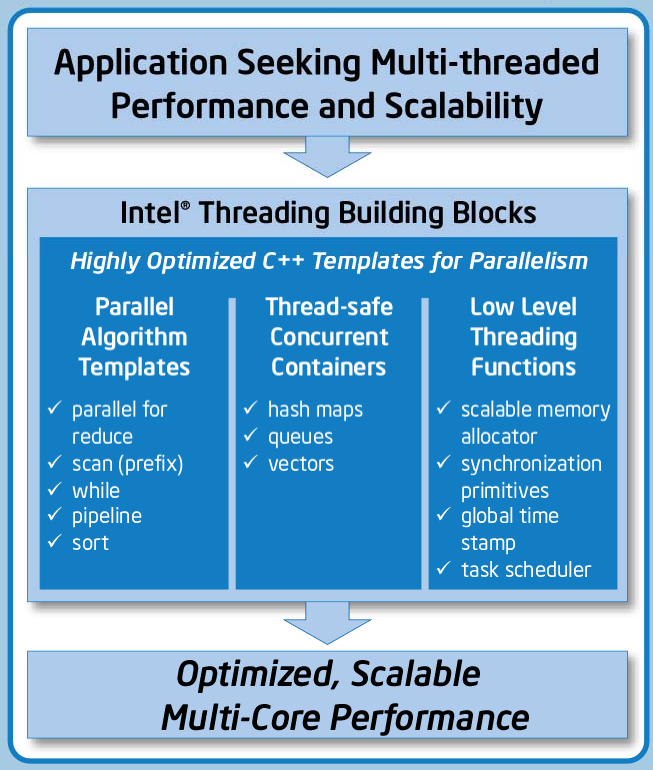
Intel® Threading Building Blocks
Extend C++ for parallelism

- Features
  - A C++ runtime library that uses familiar task patterns, not threads
  - A high level abstraction requiring less code for threading without sacrificing performance
  - Appropriately scales to the number of cores available
  - The thread library API is portable across Linux, Windows, or Mac OS platforms
  - Works with all C++ compilers (i.e. Microsoft, GNU and Intel)
- What's New
  - Open source version available at www.threadingbuildingblocks.org
  - Auto_partitioner for better parallel algorithms
  - Microsoft Vista* support
  - Full, native 64 bit support for Mac OS X*

"We're excited about the potential of Intel® Threading Building Blocks to **bring scalable performance automatically**, without requiring us to update our code to support the latest multi-core processor."

Gerry Hawkins
Maya Team Leader
Media & Entertainment
Autodesk

JOLT 17th Annual PRODUCTIVITY AWARD



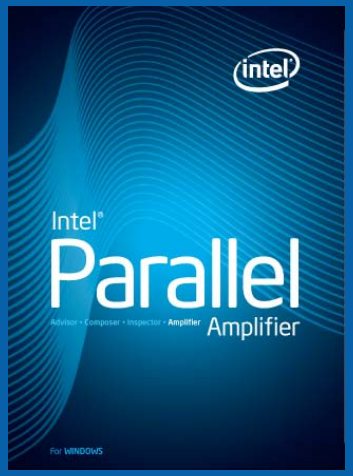
coming soon… Intel® Parallel Studio

Helps programmers throughout the development cycle

Design

Code

Debug



Tune

Software products that help solve the greatest parallelism challenges developers face

Intel® Parallel Advisor

Insight into where applications benefit most from parallelism

- Advisor is a new category of development product
- Advisor helps understand where to add parallelism to existing source code.
  - How to implement threads and provide suggestions areas
  - Spotlights where parallelism can be added
  - Helps make better design decisions
    - Shows consequences of decisions – identifies conflicts
    - Suggest ways to resolve conflicts
- Microsoft* Visual Studio* Integration
- Beta mid-2009, product late 2009

Intel® Parallel Composer

Speeds software development incorporating parallelism with a C/C++ compiler and comprehensive threaded libraries

- Simplifies threading for improved developer productivity
  - "Think Parallel" and code it without low-level thread management
- Enables Microsoft* Visual Studio* developers to add parallelism to applications
- Intel® Threading Building Blocks
- Support for lambda functions
- Pre-threaded domain-specific libraries
- Parallel debugging functionality
- Data parallel arrays
- Simple concurrency functions
- OpenMP* 3.0
- Auto-vectorization, auto-parallelization
- Innovative "Parallel Lint" helps detect parallel errors at compile time
- Microsoft* Visual Studio* Integration
- Beta Q4 2008, product mid-2009

- Spawn/par
- Parallel debug plug-in.
- Intel® Integrated Performance Primitives (Intel® IPP)
- Interoperate with all other Intel tools
- Parallel valarray
- Interoperate with Microsoft tools

Intel® Parallel Inspector

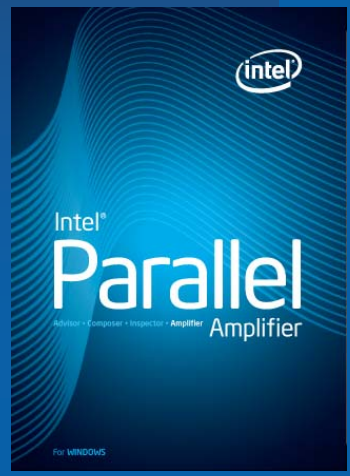
Proactive "bug finder"; flexible tool to add reliability regardless of parallelism models used

- Inspector sets a "must use" standard for shipping stable and reliable threaded applications – a proactive "bug finder."
- Does not require that application uses a single particular model of parallelism to get safety.
- Unlike traditional debuggers, Inspector detects hard-to-find threading errors in multi-threaded C/C++ Windows applications.
  - Root-cause analysis for crash-causing defects such as data races and deadlocks
  - Automatically monitoring the runtime behavior of the code to ensure application reliability
  - Critical for nondeterministic (the execution sequence can change from run to run) errors that are difficult to reproduce
  - Based on Intel® Thread Checker technology, plus more!
- Microsoft* Visual Studio* Integration
- Beta by January 2009, product mid-2009

Intel® Parallel Amplifier

Find unexpected serialization which limits scaling, to optimize performance to use all processor cores.

- Amplifier makes it simple to quickly find multi-core performance bottlenecks, for everyone – not just "experts"
  - Provides quick access to scaling information for faster and improved decision-making
  - No need to know the processor architecture or assembly code
  - Takes away the guesswork by accurately measuring programs performance behavior
  - Designed with significant user input – Intel application engineers, customers, and Whatif.intel.com community (PTU)
  - Makes Intel® Thread Profiler and Intel® VTune Performance Analyzer technology much more accessible
- Microsoft* Visual Studio* Integration
- Beta by January 2009, product mid-2009



Enabling the Next Generation

Working with professors for teaching

We asked:
How can we share our expertise (training) for professionals, and help educators?

Needed in *all* undergraduate programming courses.

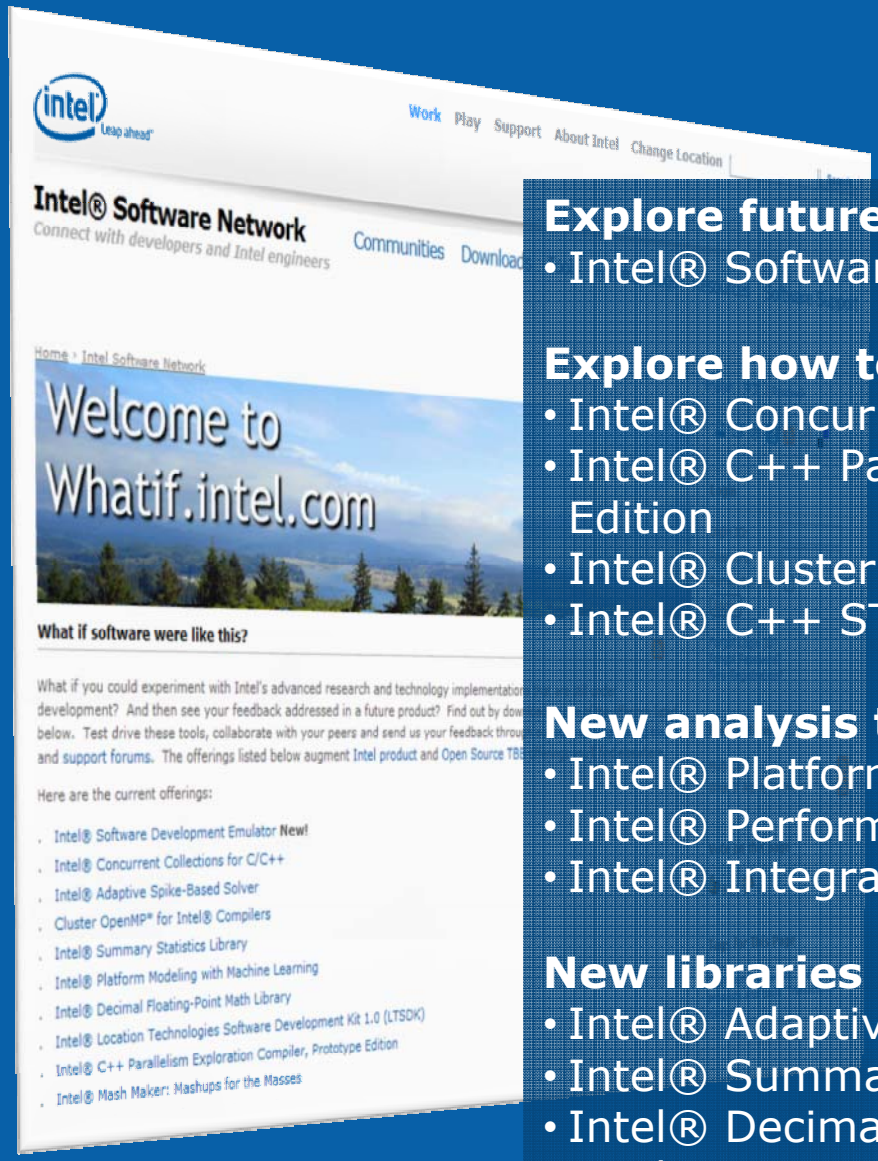
2006: 40 universities
2007: 407 universities
2008: 822 universities and growing

Over 65K students used material from this program already in 2008.

intel.com/software/college

WhatIf.intel.com

Access innovations… *in the formative stages*



Explore future processor instructions sets

- Intel® Software Development Emulator ***added AUGUST '08***

Explore how to CODE for parallelism

- Intel® Concurrent Collections for C/C++ ***added mid-2008***
- Intel® C++ Parallelism Exploration Compiler, Prototype Edition
- Intel® Cluster OpenMP* for Intel® Compilers
- Intel® C++ STM Compiler, Prototype Edition 2.0

New analysis tools

- Intel® Platform Modeling with Machine Learning ***RECENT +***
- Intel® Performance Tuning Utility 3.1 ***MOST POPULAR***
- Intel® Integrated Debugger for Java*/JNI Environments

New libraries

- Intel® Adaptive Spike-Based Solver ***RECENT ADD***
- Intel® Summary Statistics Library
- Intel® Decimal Floating-Point Math Library ***RECENT ADD***
- Intel® Location Technologies Software Development Kit 1.0

New web technologies

- Intel® Mash Maker: Mashups for the Masses ***GRADUATE***

Intel® C++ STM Compiler, Prototype Edition 2.0 - Intel® Software Network - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://software.intel.com/en-us/articles/intel-c-stm-compiler-prototype-edition-20 Go Links

Work Play Support About Intel Change Location Search

intel

Intel® Software Network

Connect with developers and Intel engineers

Communities Downloads Tools Forums/Blogs Resources Software Support

Home -> Articles

English | 中文 | Русский

Intel® C++ STM Compiler, Prototype Edition 2.0

Submit New Article

Last Modified On : November 19, 2007 9:34 AM PST

Rate

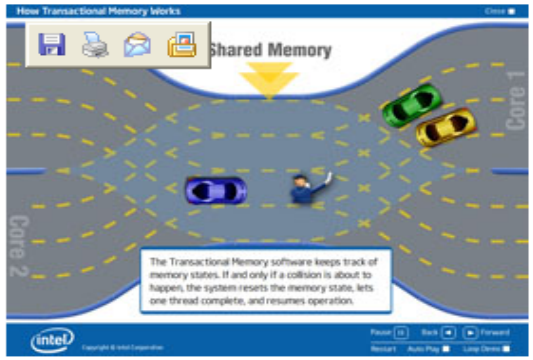
What If Home | Product Overview | Technical Requirements | FAQ | Primary Technology Contacts | Discussion Forum | Blog

Download Now!

Product Overview

Parallel programming has traditionally been considered using locks to synchronize concurrent access to shared data. Lock-based synchronization, however, has known pitfalls: using locks for fine-grain synchronization and composing code that already uses locks are both difficult and prone to deadlock. Transactional memory is proposed to simplify parallel programming by supporting "atomic" and "isolated" execution of user-specified tasks. It provides an alternate concurrency control mechanism that avoids these pitfalls and eases parallel programming. The Transactional Memory C++ language constructs that are included open the door for users to exercise the new language constructs for parallel programming, understand the transaction memory programming model, and provide feedback on the usefulness of these extensions with Intel® C++ STM Compiler Prototype Edition. This posting includes the Intel® C++ STM Compiler Prototype Edition 2.0 and runtime libraries for Intel transactional memory language construct extensions.

Flash Demo



Login

Login ID:

Password:

Remember Me?

Login

Forgot Login ID?
Forgot Password?
New Registration?

Search

Go

Advanced Search

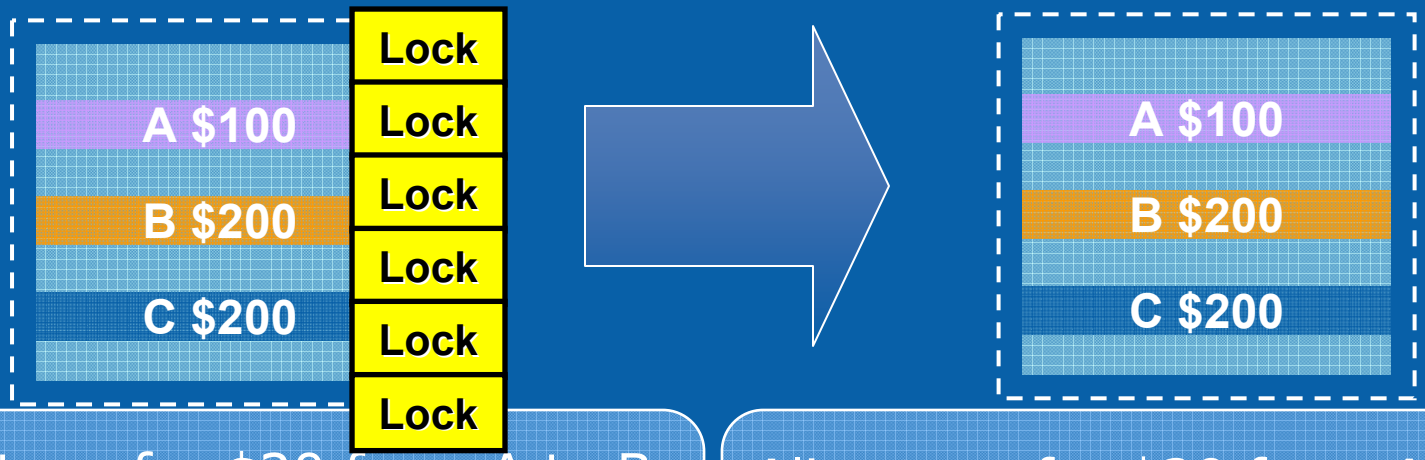
Author

javascript: flashWin();

Trusted sites

start 10 Microsoft... P:\talks\secr08 Q4 2008 Trav... Intel® C++ S... 3 Microsoft ... 100% 3:41 PM

Transactional Memory



Alice transfer $20 from A to B

Locks A
Locks B
A = A – 20
B = B + 20
Unlocks A
Unlocks B

Alice transfer $20 from A to B

begin_xaction
A = A – 20
B = B + 20
end_xaction

Programmer
manually ensures no race

System
automatically ensures no race

A C++ Example using the prototype

```
int s = 0;
class B
{  public:
   __declspec(tm_callable)
   virtual void inc()
   {
      s = s + 1;
   }
};
class C : public B
{  public:
   __declspec(tm_callable)
   void inc()
   {
     s = s + 1;
   }
};
```

```
int main()
{  B  *x, *y;

#pragma omp parallel sections
num_threads(2)
  {
    __tm_atomic {
       x = new B();
       x->inc();
    }

#pragma omp section
     __tm_atomic {
        y = new C();
        y->inc();
    }
  }
}
```

Summary

- Programming is not "EASY"
  - Neither is parallel programming
- There isn't one magic solution for Parallel Programming 2.0
  - Methodology: design, code, debug, tune
- The right tools such as the Intel products will help make parallel programming EASIER.