Multicore: The Software View

James Cownie
Principal Engineer
SSG/DPD/PAT

12th EMEA Academic Forum

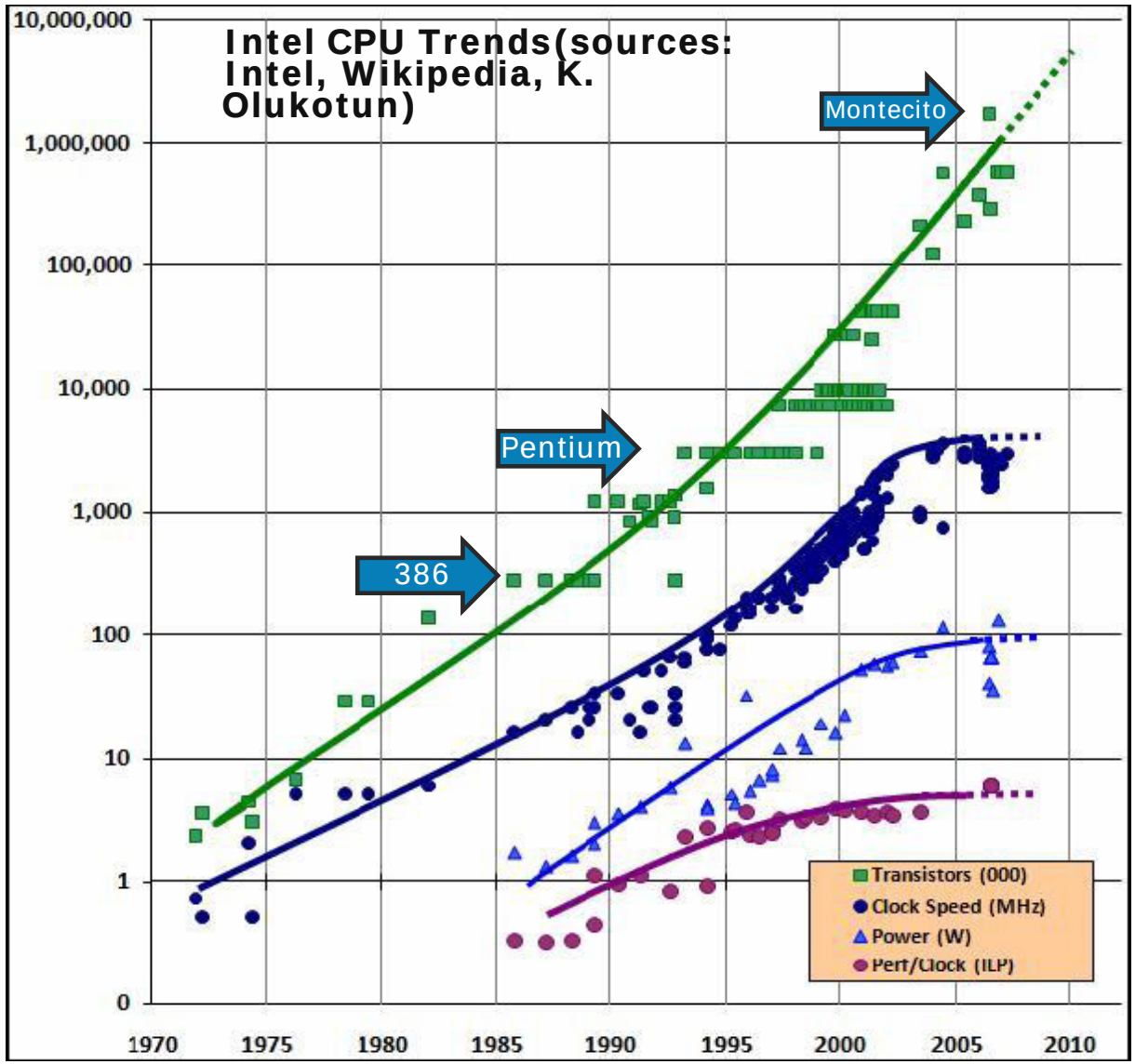
EMEA ACADEMIC FORUM

The Software View (Summary)

- Hardware technology
  - past
  - future
- Software we will need
- What Intel is doing about it
  - Now
  - In the future

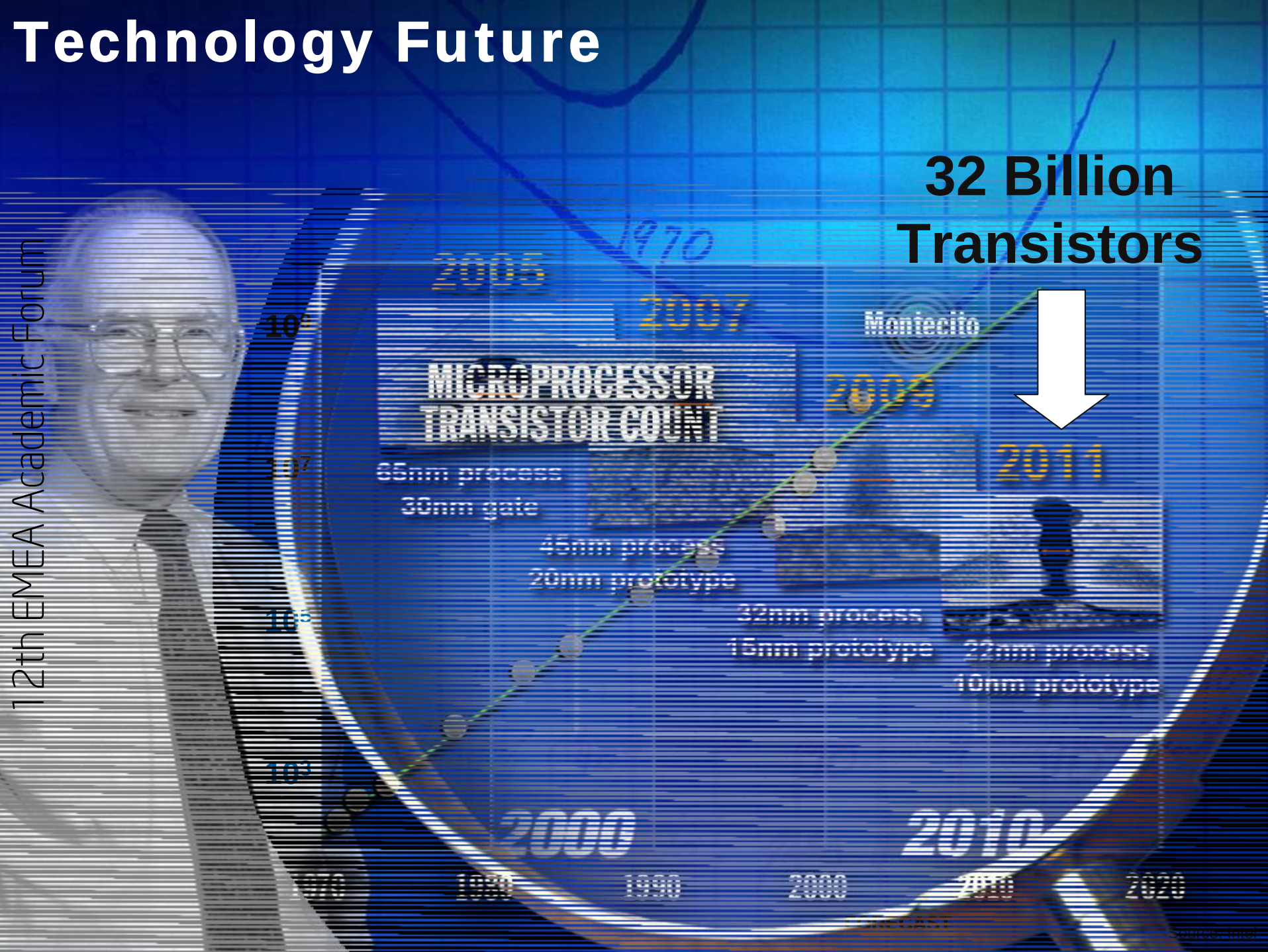
Technology Past

- Moore's law is alive and well so far.
- But performance per core is no longer increasing with it.



Graph © Herb Sutter, used with permission

Technology Future



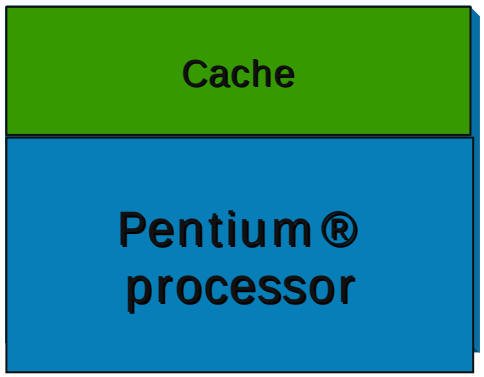
What Now?

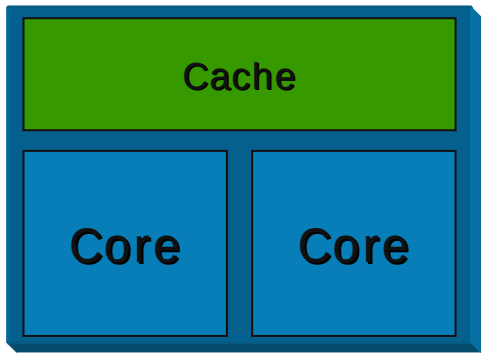
- How can we use all those transistors?
  - Clearly we can't design each one
    - at 1 man-minute per transistor 32 billion would take ~278,000 man-years of effort.
  - We can't push clock speed because of the power wall
  - Increased cache helps, but not enough
- We have to replicate components
  - Multiple cores on the same chip
  - Integrate more functions
- **BUT** this is a manufacturing imperative, it's not what us poor programmers are asking for; we just want an infinitely fast single processor

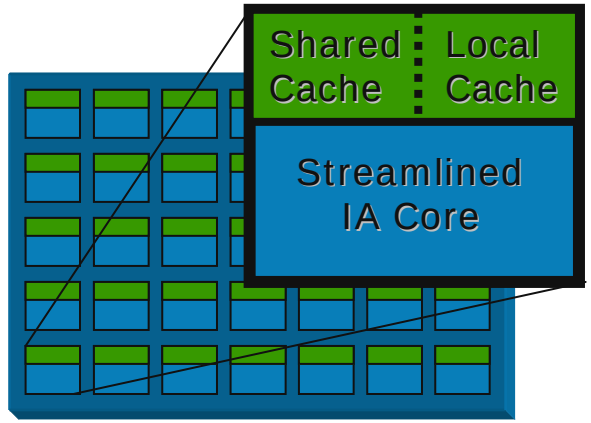
What future chips might look like:
From a few large cores to many lightweight cores

Optimized for speed

Optimized for performance/watt





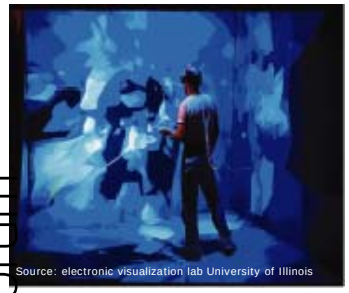


Pentium® processor era chips optimized for raw speed on single threads. Pipelined, out of order execution.

Today's chips use cores which balance single threaded and multi-threaded performance

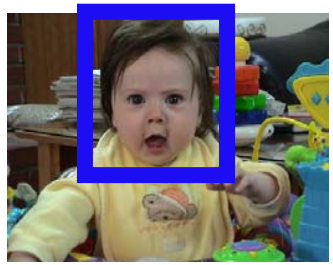
5-10 years: 10s-100s of energy efficient, IA cores optimized for multithreading

What can we do with these processors?



Personal Media Creation and Management

- Search for and edit photos and videos based on image; no need to tag the images
- Easily create videos with animation



Entertainment

- Watch yourself star in a movie or game
- Hold and interact with objects in the virtual world
- Control with speech and gesture
- Immersive: 3D, interactive

Source; http://vhp.med.umich.edu/Surgical-Simulation.jpg

Health

- Virtual health worker monitors and assists elders/patients living alone
- Real-time realistic 3D visualization of body systems
- Effects of changes in diet, exercise and disease on body

Learning and Travel

- Surround yourself with sights and sounds of far-away places
- Practice new languages and customs

Source: Steven K. Feiner, Columbia University

Telepresence & Collaboration

- As if you are in the same place with family and friends—without the travel
- Appointments with doctors, teachers, leaders
- Develop and perform art with those far away

Where are we now?

- Most code is written in sequential languages
  - C/C++
  - MRTE languages
  - Scripting languages
- We have
  - Threads
  - Locks
  - OpenMP
  - Lots of programmers who have never written a parallel program, and don't want to.

Where do we need to be?

- Parallel programming must be simple
- Everyone writes scalable parallel codes without thinking about it.

What makes parallel programming hard?

- Identifying parallelism
- Shared state
- Requirement for non-local reasoning
  - Data races
  - Locks
  - Thread interaction
- Lack of language support

How can we address these problems?

- Tools
- Better programming models and languages
- Application specific libraries which hide the parallelism

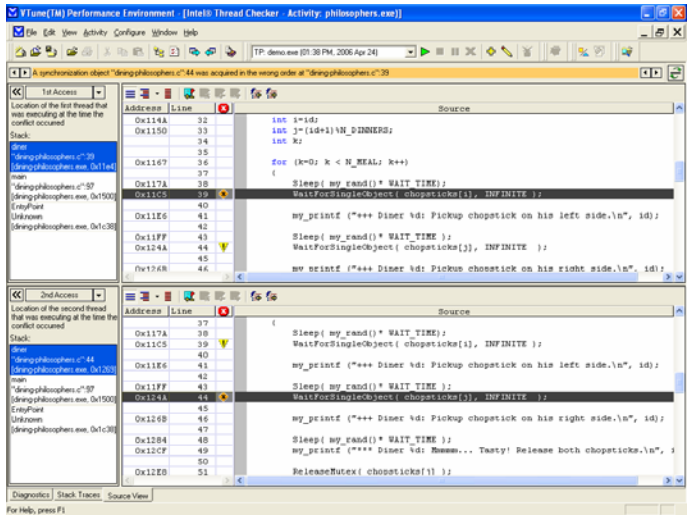
What is Intel already doing?

- Support for existing programming models
- Tools
  - Compilers
  - Intel® Thread Checker
  - Intel® Thread Profiler
  - Math Kernel Library, Integrated Performance Primitives, …
- Ways to express parallelism
  - OpenMP*
  - Intel® Threading Building Blocks

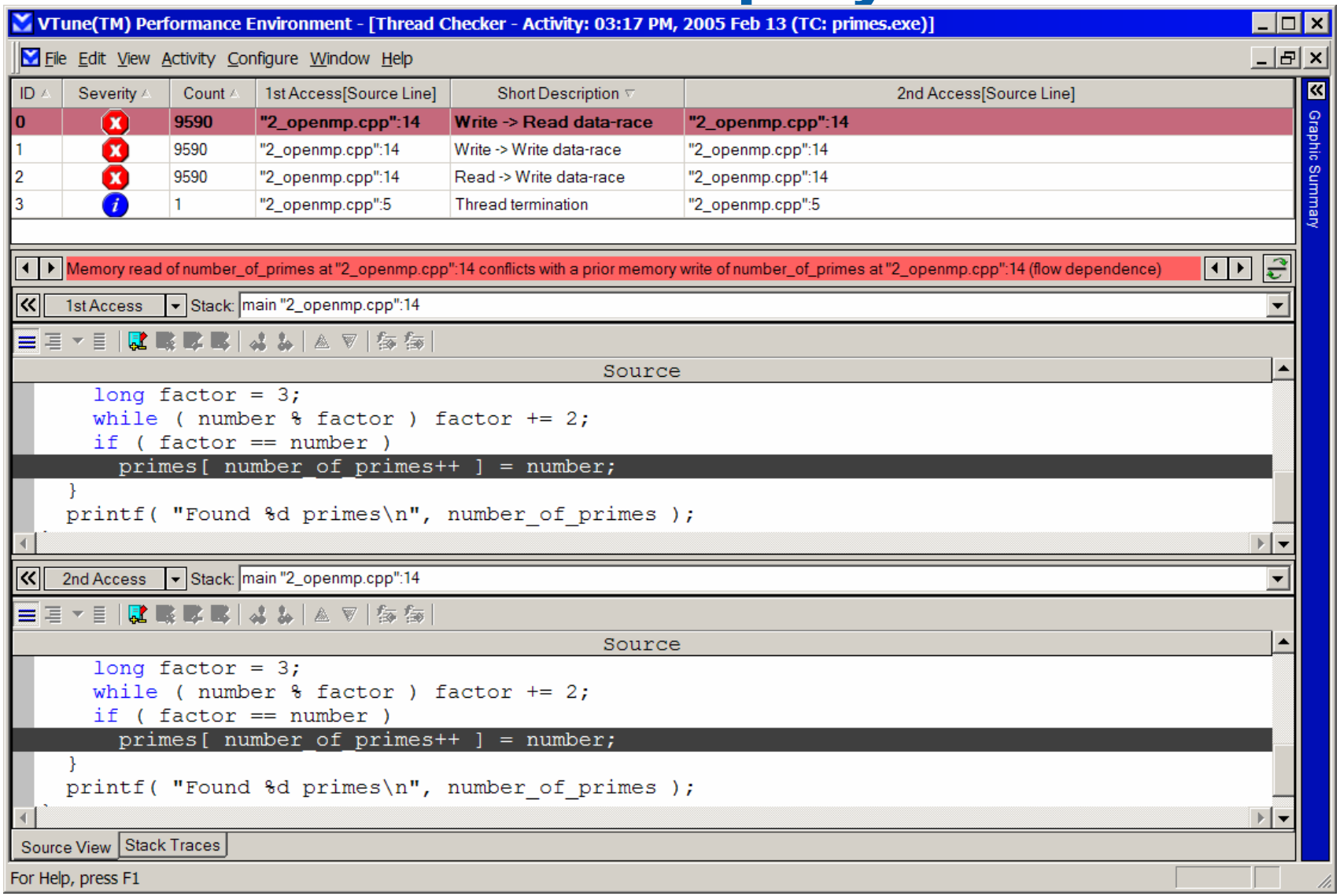
Intel® Thread Checker

Create Threads Faster

- Detect data races even if they did not occur in a particular run.
- View errors in the context of the source code.
- Powerful sorting and filtering.
- Can be used in automated regression testing.
- Uses dynamic binary instrumentation to log read and write accesses to memory (see Moshe's PIN presentation!)



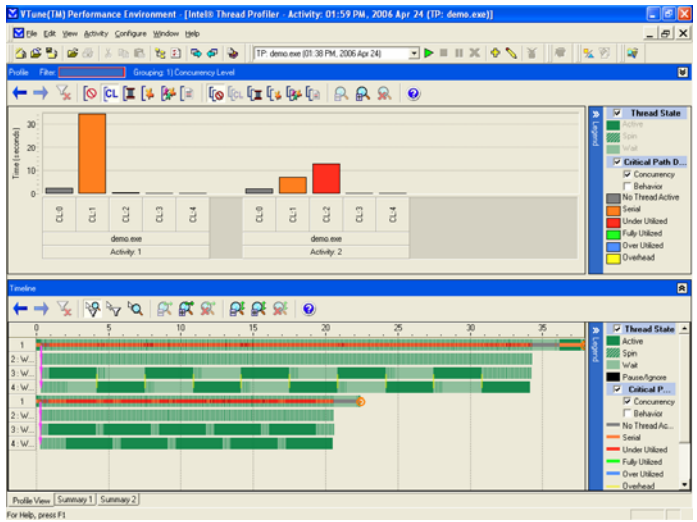
Thread Checker Display



Intel® Thread Profiler

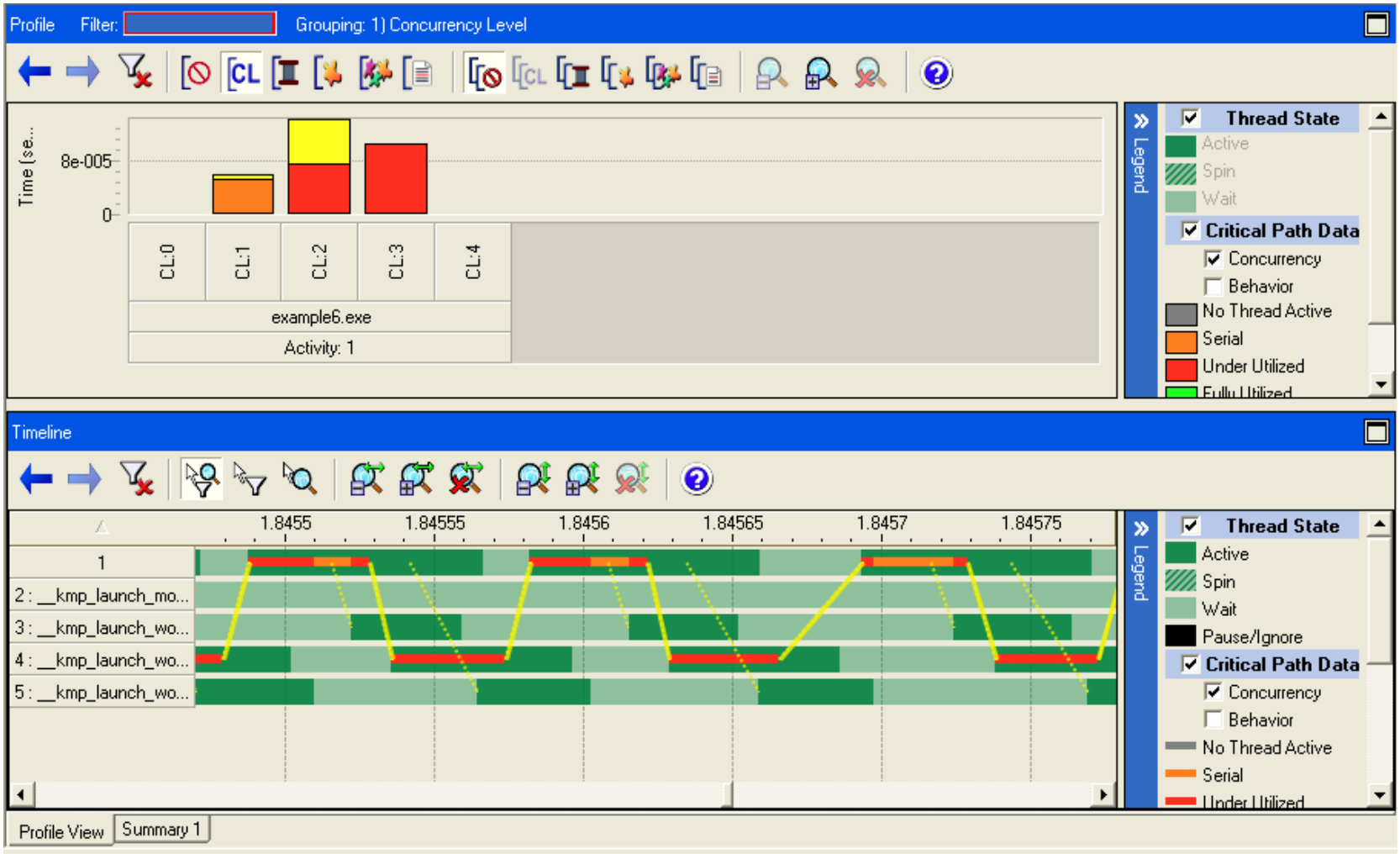
Optimize Threads Faster

- Features & Benefits
  - Observe the synchronization behavior of your program
  - View application concurrency level to ensure core utilization
  - Identify where thread and synchronization related overhead impacts performance
  - Understand the distribution of work to threads
  - Understand when threads are active and inactive
  - Estimate the performance potential of different design choices
  - Detect lock contention
  - Perform critical path analysis



12th EMEA Academic Forum

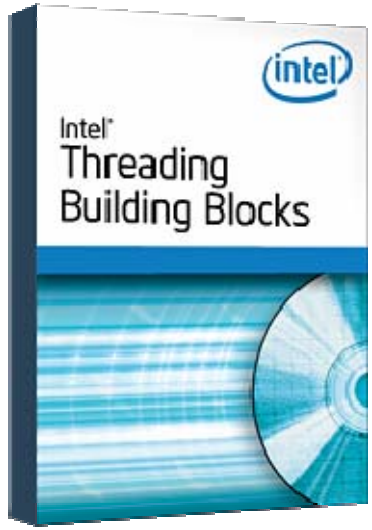
Thread Profiler display



12th EMEA Academic Forum

Intel® Threading Building Blocks

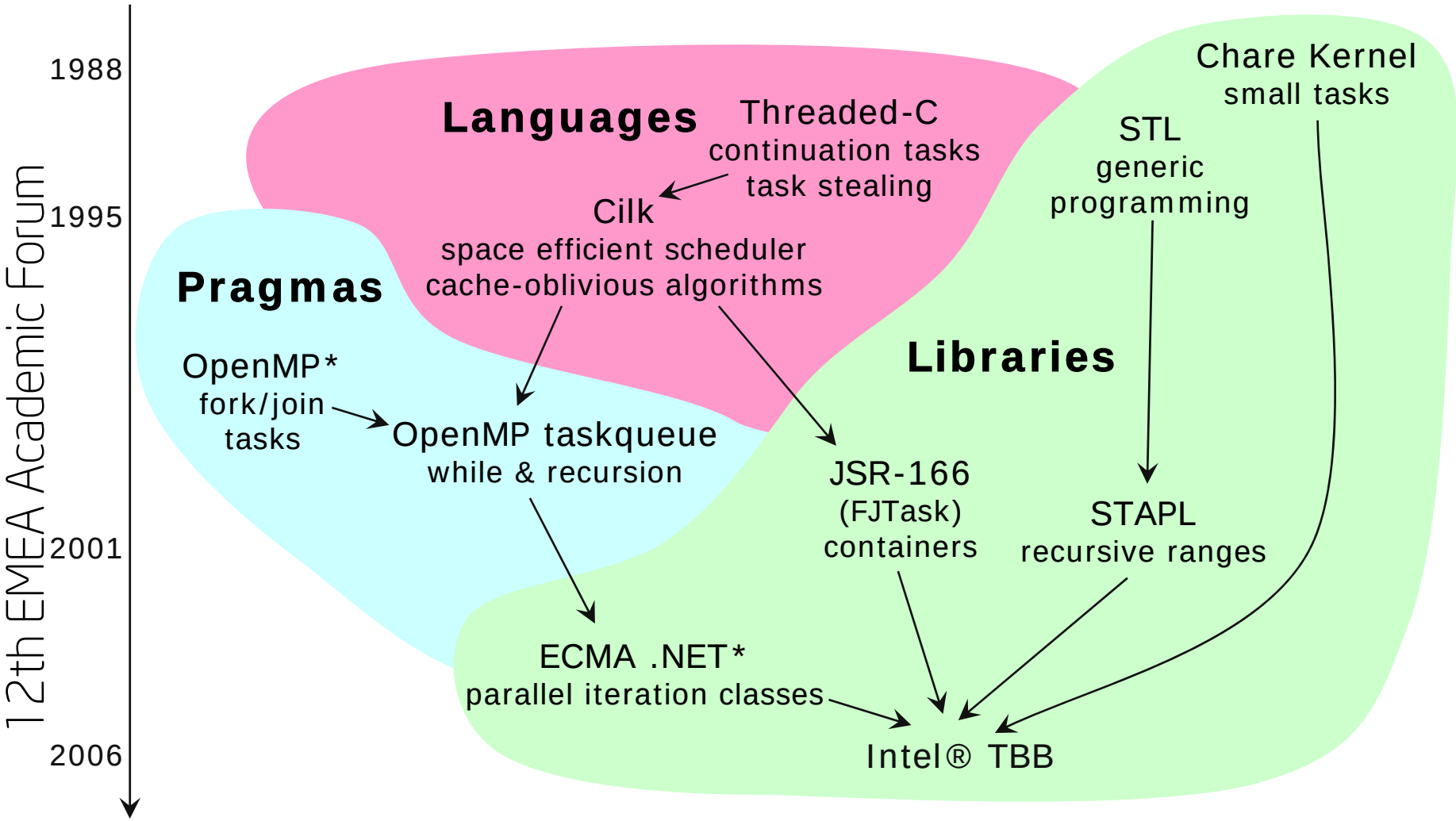
- C++ Template library for expressing parallelism
- Can be used with standards conforming C++ compilers (not restricted to the Intel compiler)
- Raises the level at which parallelism is expressed above threads
- Emphasizes *scalable, data parallel* programming
  - Solutions based on functional decomposition usually do not scale.



Intel® Threading Building Blocks

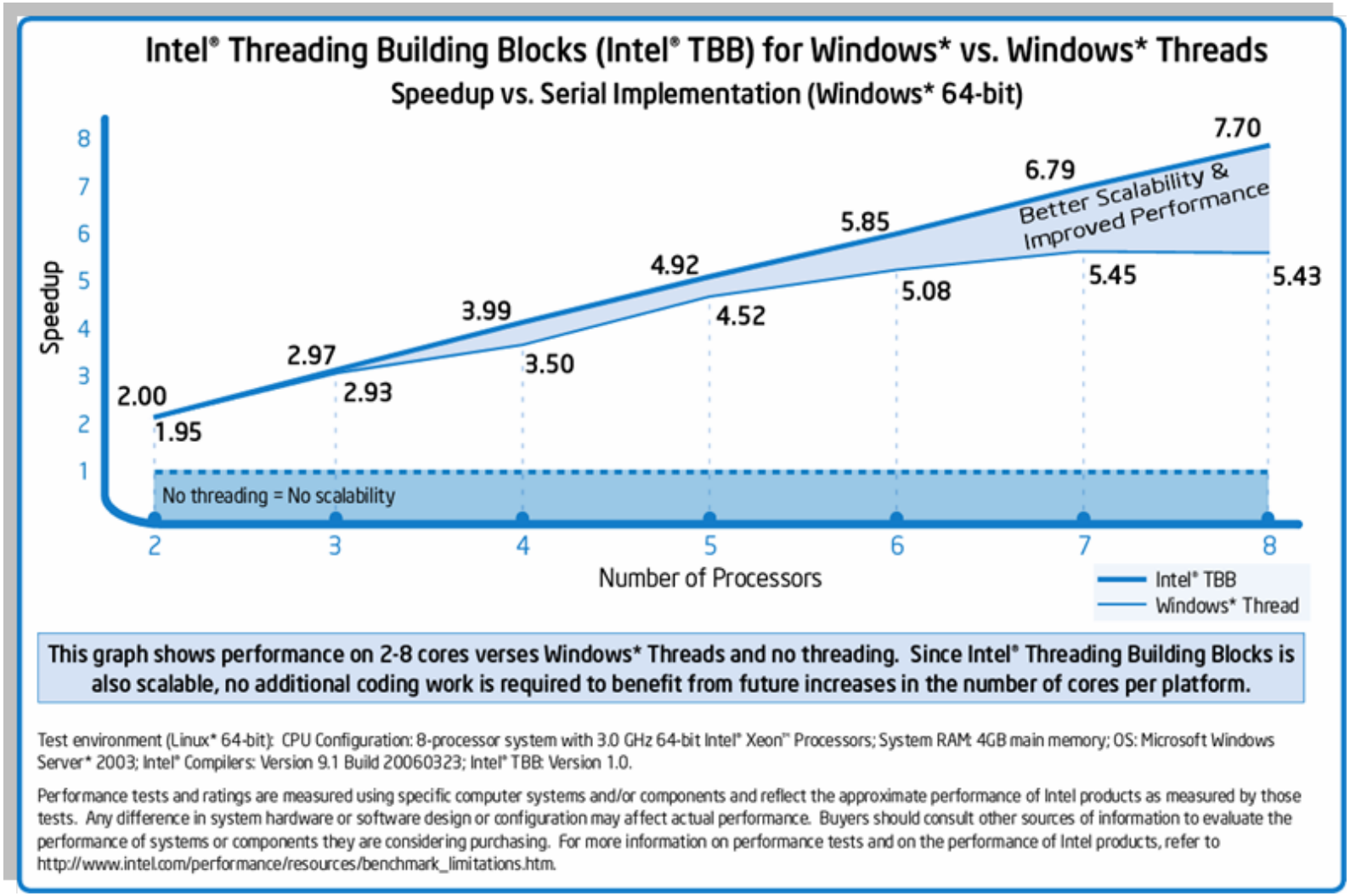
- You specify *task patterns* instead of threads
  - Library maps user-defined logical tasks onto physical threads, efficiently using cache and balancing load
  - Full support for *nested parallelism*
- Targets threading for *robust performance*
  - Designed to provide portable scalable performance for computationally intense portions of shrink-wrapped applications.
- *Compatible* with other threading packages
  - Designed for CPU bound computation, not I/O bound or real-time.
  - Library can be used in concert with other threading packages such as native threads and OpenMP.
- Emphasizes *scalable, data parallel* programming
  - Solutions based on functional decomposition usually do not scale.

TBB Family Tree



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

TBB Performance



This graph shows performance on 2-8 cores verses Windows* Threads and no threading. Since Intel® Threading Building Blocks is also scalable, no additional coding work is required to benefit from future increases in the number of cores per platform.

Test environment (Linux* 64-bit): CPU Configuration: 8-processor system with 3.0 GHz 64-bit Intel® Xeon™ Processors; System RAM: 4GB main memory; OS: Microsoft Windows Server* 2003; Intel® Compilers: Version 9.1 Build 20060323; Intel® TBB: Version 1.0.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to http://www.intel.com/performance/resources/benchmark_limitations.htm.

What is Intel doing for the future?

- Better tools
  - to locate parallelism.
  - to express parallelism.
  - to validate parallel codes.
- Support for new programming models
  - Transactional memory
  - Data-parallel programming
- BUT we need more so…

You tell us!

- This is the Academic Forum, so you tell us.
- What are the solutions?
- Where are the new language ideas?
- Can you design a statically checked race-free language which is useful?
- Can naïve users really use functional languages?
- Any language which talks about threads is too low level for most users. So how do we raise the language level?
- Should we be doing message passing inside the node?
  - It's the only demonstrated way to achieve high scalability.
  - Do we really need to bring back Occam? ☺

Backup

Intel Threading Tools URLs

- Intel® Threading Building Blocks
  - Try 30-day evaluation copy
    - >Linux, Mac, Windows
  - Documentation can be downloaded for free
    - >*Getting Started Guide*, *Tutorial*, *Reference*

http://www.intel.com/software/products/tbb

- Thread Checker – finds threading errors like conditions
- Thread Profiler – finds threading performance problems

http://www.intel.com/software/products/threading