# INTEL® THREADING BUILDING BLOCKS (INTEL® TBB) 2017

# Multi-threading and heterogeneous computing made easy with Intel TBB

## What is Intel® TBB?

Intel TBB is a highly templatized C++ library designed to simplify the task of adding parallelism to your application by taking advantage of all the CPU's either on a single device or across multiple devices (heterogeneity).

## Why should you use Intel® TBB?

- High Performance
- Easy to use API's
- Faster Time To Market
- Production Ready

**Optimized for**

**Supports**

**Addresses**


*Find out more at:* ***http://software.intel.com/intel-tbb***
*Contact us through our forum:*
***http://software.intel.com/en-us/forums/intel-threading-building-blocks***

## How to get Intel® TBB?

Intel Parallel Studio XE
Intel System Studio
Free Tools Program
Open source site

## Applications

- Animation Rendering
- Numeric weather prediction
- Oceanography & Astrophysics
- Artificial Intelligence & Automation
- Genetic Engineering
- Medical applications (Image processing, MRI reconstruction)
- Remote sensing applications
- Socio Economics
- Financial sector (stock derivative pricing, statistics)
- Bulk updating data files
- Any Big Data problems

# Customer Success Stories

Dreamworks Fur Shader used Intel® TBB which produced an average of 5x speedup on fur generation loop

Intel's TBB was an invaluable help in multi-threading our in-house renderer CGIStudio and is now also used in animation and simulation software. Beside the ease of use, it takes care of the two most important aspects of running an application on multiple cores -- load balancing and scalability.

- Maurice van Swaaij, Blue Sky Studios

"Intel® TBB provided us with optimized code that we did not have to develop or maintain for critical system services. I could assign my developers to code what we bring to the software table."

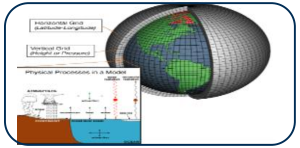
- Michaël Rouillé, CTO, Golaem

"Using Intel TBB's new flow graph feature, we accomplished what was previously not possible, parallelize a very sizable task graph with thousands of interrelationships – all in about a week.

- Robert Link
GCAM Project Scientist, Pacific Northwest National Laboratory

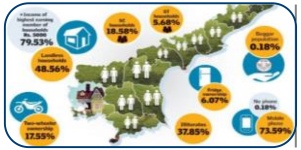
# What Kind of applications needs to be multi-threaded?

Multi-threading is for applications where the problem can be broken down into tasks that can be run in parallel or the problem itself is massively parallel as some mathematics or analytical problems are.

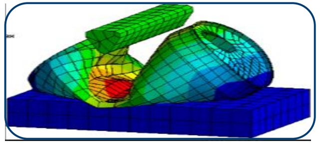


**Numeric Weather Prediction**
Mathematical modelling
Forecasting the future state of weather
Data assimilation

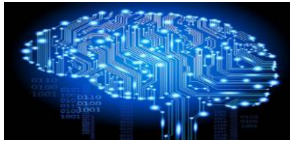
**Oceanography & Astrophysics**
Study the wealth of ocean
PIC, PM and n-body simulations
Astrophysics research




**Socio Economics**
Modelling economy of nation/world
Scenario calculations & optimizations of economic models

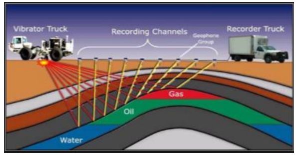


**Finite Element Analysis**
Multi-physics problems
Design of huge structures like ships, dams, supersonic jets.
Solving Partial Differential Equations (PDE"S)



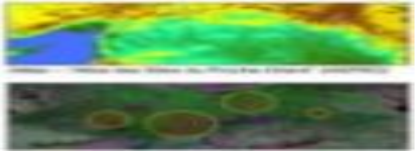
**AI & Automation**
Image Processing
Expert Systems
Natural Language Processing(NLP)
Pattern Recognition
ADAS

**Genetic Engineering**
DNA Sequence Analysis



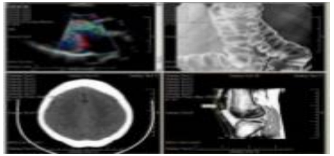

**Seismic Exploration**
Use seismic waves to estimate earth's properties
Sensor Analysis

**Weapon Research & Defense**
Nuclear weapon's performance & certification
Plutonium research



**Remote Sensing Applications**
Used in Agriculture/Forestry to read specialized formats containing:

- sensor image data
- Georeferencing information
- Sensor metadata



**Medical Applications**
Medical Image Processing
Scanning human body/brain
MRI reconstruction
Vertebra detection & segmentation in X-ray images
Brain Fiber tracking

**Energy Resource Exploration**
Gather and manage information about energy resources like oil, natural gas
Maintain records of global energy crisis, nuclear reactor safety

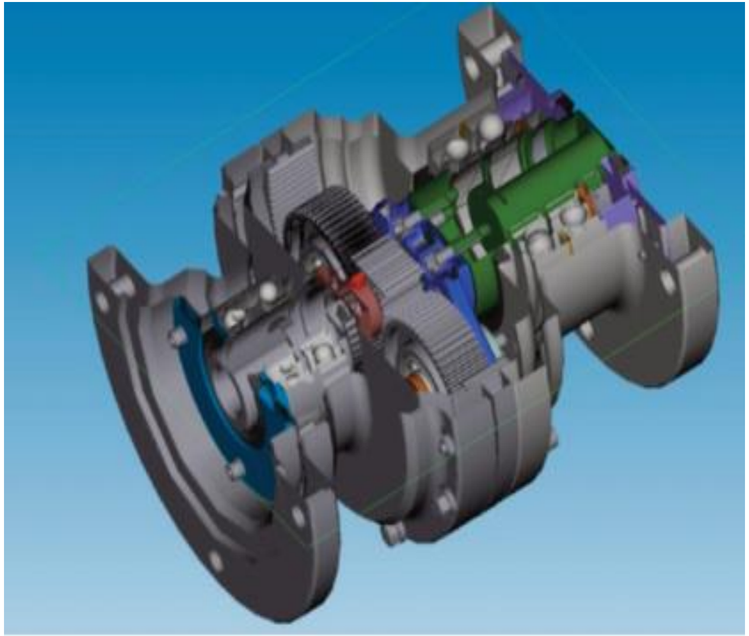
# Advantages of using Intel TBB over other threading models

- Specify tasks instead of manipulating threads. Intel® TBB maps your logical tasks onto threads with full support for nested parallelism
- Intel TBB uses proven , efficient parallel patterns.
- Intel TBB uses work stealing to support the load balance of unknown execution time for tasks.  This has the advantage of low-overhead polymorphism.
- Flow graph feature in Intel TBB allows developers to easily express dependency and data flow graphs.
- Has high level parallel algorithms and concurrent containers and low level building blocks like scalable memory allocator , locks and atomic operations.

# Intel TBB – " The backbone of CAD Exchanger's parallelism"

Parallelism Brings CAD Exchanger* Software Dramatic Gains in Performance and User Satisfaction, Plus a Competitive Advantage "CAD Exchanger* is broadly using multi-threaded algorithms to increase performance on multi-core systems," said Roman Lygin of CADEX, Ltd. "This is the key advantage over our competitors." Benchmarks show how it outperforms earlier editions in significant ways:

- Some heavyweight computational algorithms, such as blended surface approximation, were **accelerated by 15X** over single-thread mode.
- Multi-threaded visualization significantly increased the responsiveness of the GUI application, which in turn improved the user experience. **Less time spent waiting** means more time to interact and innovate.
- Parallel file I/O is **2.5× faster**, and visualization time was **reduced by up to 4X**



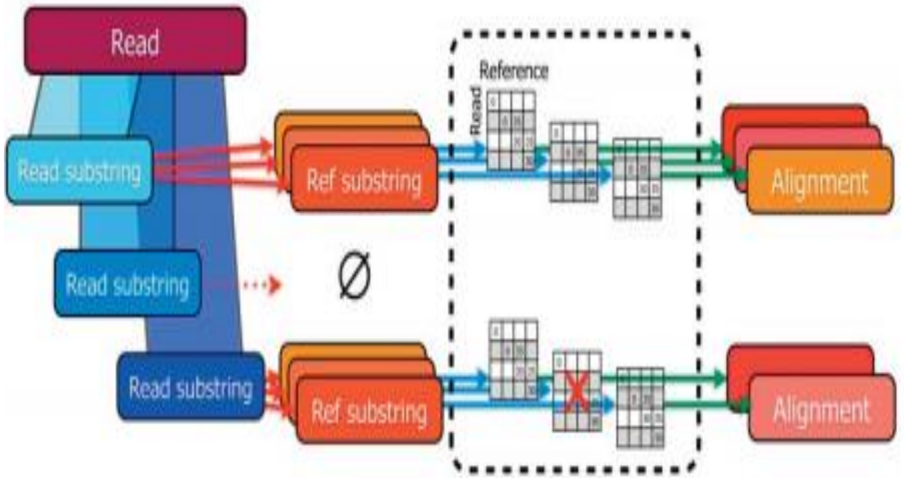
Case Study link here

# Intel TBB helps John Hopkins University prepare for a many core future

Modern DNA sequencing provides an inexpensive and high-resolution window into diverse aspects of biology, genetics, and disease. Like a microscope, a sequencer produces a snapshot of a collection of cells. Unlike a microscope, a sequencer does not provide a finished, ready-to-interpret image. Rather, it produces billions of tiny snippets (reads) of DNA that must first be composed into longer, interpretable units such as genes or chromosomes. Bowtie* and Bowtie 2* are widely used software tools produced in the University's Langmead Lab that allow biologists to piece together the fragmentary evidence generated by DNA sequencers.

Johns Hopkins and Intel have been collaborating on the Bowtie 2 application. Adding parallelism via Intel TBB resulted in a substantial speedup of the application. By splitting reads from parsing in a critical section, the team saw essentially ideal **scaling up to 120 threads**.

The team was able to **effectively prepare these core genomics software tools for the many-core future around the corner.**




Case Study link here

# Rich Feature Set for Parallelism
# Intel® Threading Building Blocks (Intel® TBB)

Parallel algorithms and data structures

Threads and synchronization

Memory allocation and task scheduling

**Generic Parallel Algorithms**

Efficient scalable way to exploit the power of multi-core without having to start from scratch.

**Flow Graph**

A set of classes to express parallelism as a graph of compute dependencies and/or data flow

**Concurrent Containers**

Concurrent access, and a scalable alternative to containers that are externally locked for thread-safety

**Synchronization Primitives**

Atomic operations, a variety of mutexes with different properties, condition variables

**Task Scheduler**

Sophisticated work scheduling engine that empowers parallel algorithms and the flow graph

**Timers and Exceptions**

Thread-safe timers and exception classes

**Threads**

OS API wrappers

**Thread Local Storage**

Efficient implementation for unlimited number of thread-local variables

**Memory Allocation**

Scalable memory manager and false-sharing free allocators

# Features and Functions List
# Intel® Threading Building Blocks (Intel® TBB)

Parallel algorithms and data structures

Threads and synchronization

Memory allocation and task scheduling

## Generic Parallel Algorithms

- parallel_for
- parallel_reduce
- parallel_for_each
- parallel_do
- parallel_invoke
- parallel_sort
- parallel_deterministic_reduce
- parallel_scan
- parallel_pipeline
- pipeline

## Flow Graph

- graph
- continue_node
- source_node
- function_node
- multifunction_node
- overwrite_node
- write_once_node
- limiter_node
- buffer_node
- queue_node
- priority_queue_node
- sequencer_node
- broadcast_node
- join_node
- split_node
- indexer_node

## Concurrent Containers

- concurrent_unordered_map
- concurrent_unordered_multimap
- concurrent_unordered_set
- concurrent_unordered_multiset
- concurrent_hash_map
- concurrent_queue
- concurrent_bounded_queue
- concurrent_priority_queue
- concurrent_vector
- concurrent_lru_cache (preview)

## Synchronization Primitives

- atomic
- mutex
- recursive_mutex
- spin_mutex
- spin_rw_mutex
- speculative_spin_mutex
- speculative_spin_rw_mutex
- queuing_mutex
- queuing_rw_mutex
- null_mutex
- null_rw_mutex
- reader_writer_lock
- critical_section
- condition_variable
- aggregator (preview)

## Task Scheduler

- task
- task_group
- structured_task_group
- task_group_context
- task_scheduler_init
- task_scheduler_observer
- task_arena

## Timers and Exceptions

- tick_count
- tbb_exception
- captured_exception
- movable_exception

## Threads

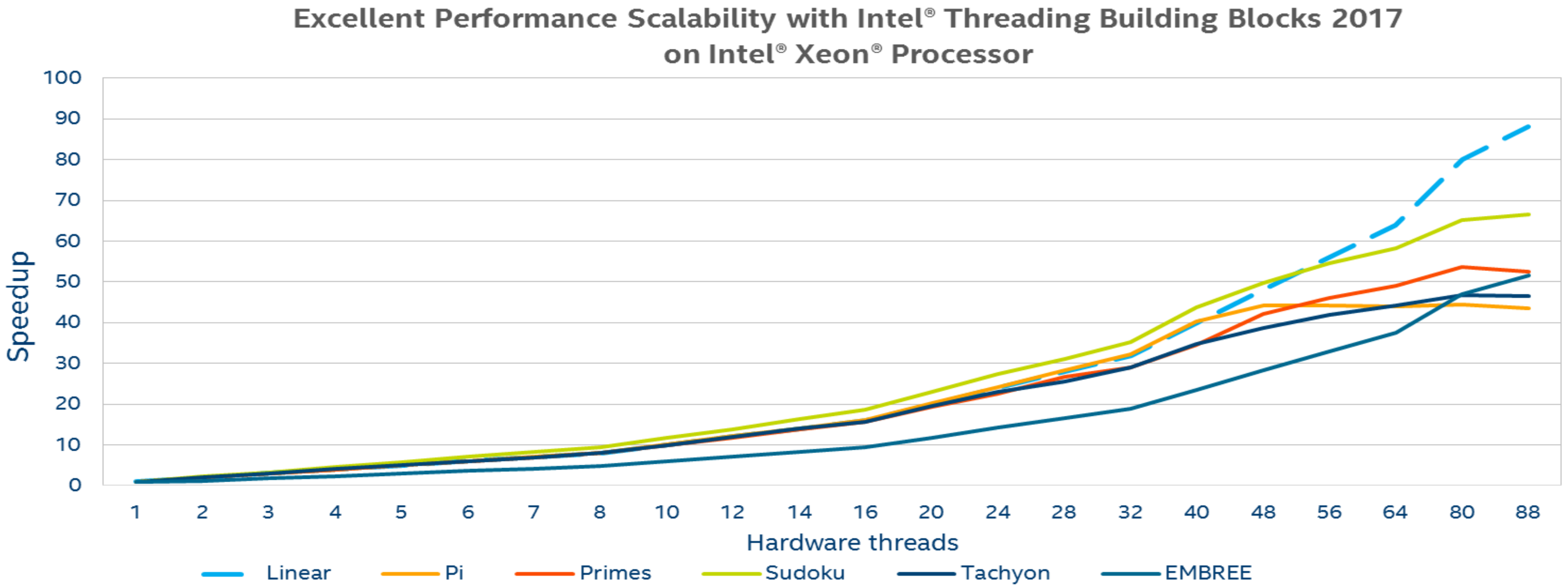
- thread

## Thread Local Storage

- combinable
- enumerable_thread_specific

## Memory Allocation

- tbb_allocator
- scalable_allocator
- cache_aligned_allocator
- zero_allocator
- aligned_space
- memory_pool (preview)

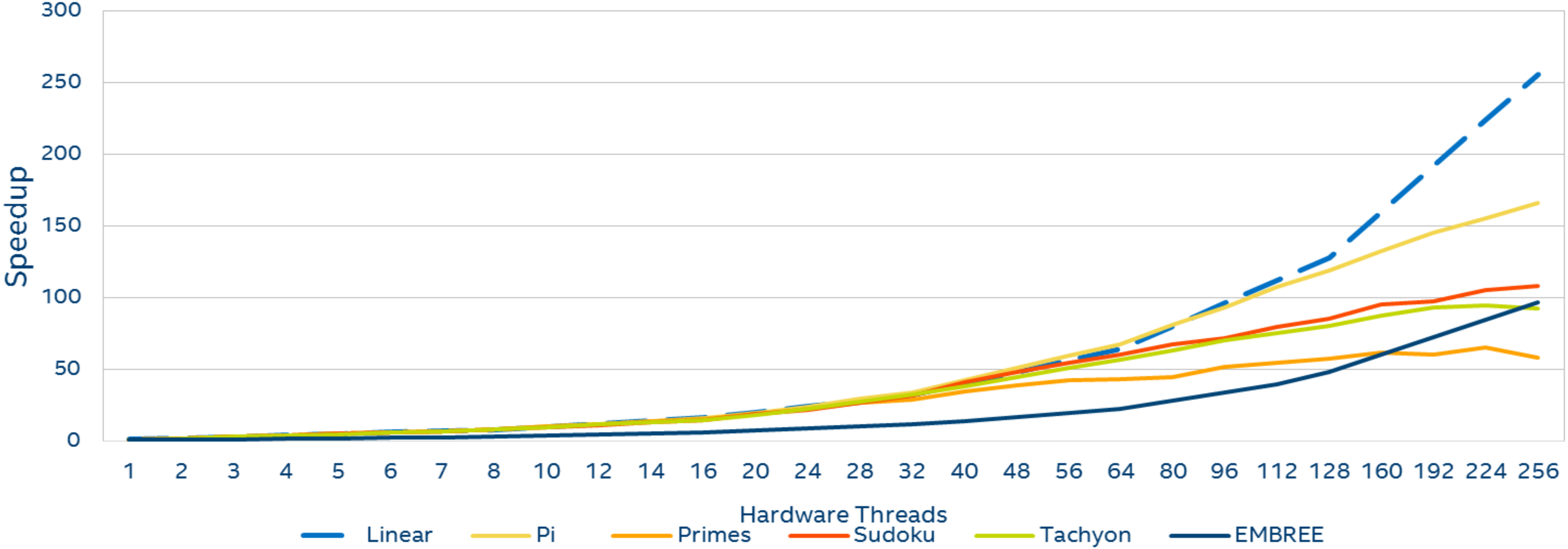



Configuration Info: Software Versions: Intel® C++ Intel® 64 Compiler, Version 17.0, Intel® Threading Building Blocks (Intel® TBB) 2017; Hardware: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz 44/88T)), 128GB Main Memory; Operating System: Red Hat Enterprise Linux Server release 7.2 (Maipo), kernel 3.10.0-327.4.5.el7.x86_64; Benchmark Source: Intel Corp. Note: sudoku, primes and tachyon are included with Intel TBB. 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 www.intel.com/performance/resources/benchmark_limitations.htm .



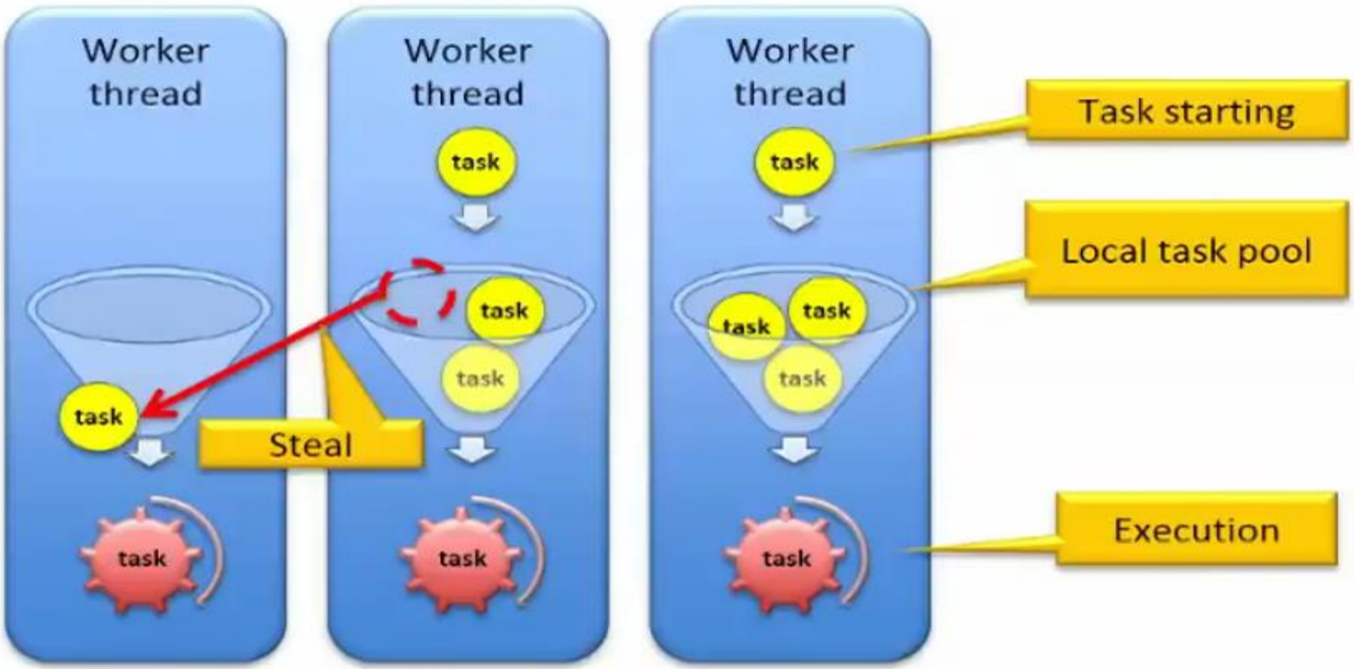
**Optimization Notice**: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

# Excellent Performance Scalability with Intel® Threading Building Blocks 2017 on Intel® Xeon Phi® Processor




Configuration Info: Software Versions: Intel® C++ Intel® 64 Compiler, Version 17.0, Intel® Threading Building Blocks (Intel® TBB) 2017; Hardware: KNL (Intel(R) Xeon Phi(TM) CPU 7250 @ 1.40GHz (68C/272T)), 128GB Main Memory; Operating System: Red Hat Enterprise Linux Server release 7.2 (Maipo), kernel 3.10.0-327.13.1.el7.mpsp_1.3.2.100.x86_64; Benchmark Source: Intel Corp.Note: sudoku, primes and tachyon are included with Intel TBB.
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 www.intel.com/performance/resources/benchmark_limitations.htm.
* Other brands and names are the property of their respective owners.
**Optimization Notice:** Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

# Task Execution in Intel TBB



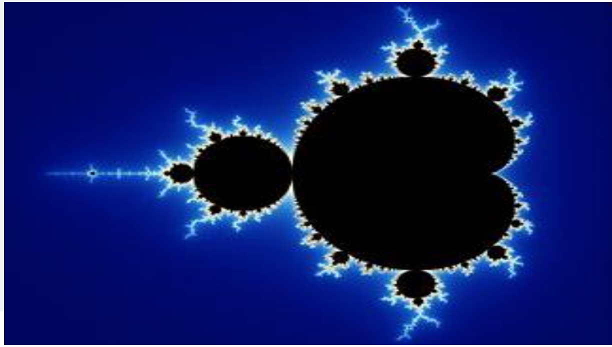

(A simplified version of the scheduler)

# Generic algorithms allow reuse of proven parallel patterns

## Intel® Threading Building Blocks (Intel® TBB)

Sequential version

```
int mandel(Complex c, int max_count) {
  int count = 0; Complex z = 0;
  for (int i = 0; i < max_count; i++) {
    if (abs(z) >= 2.0) break;
    z = z*z + c; count++;
  }
  return count;
}
```



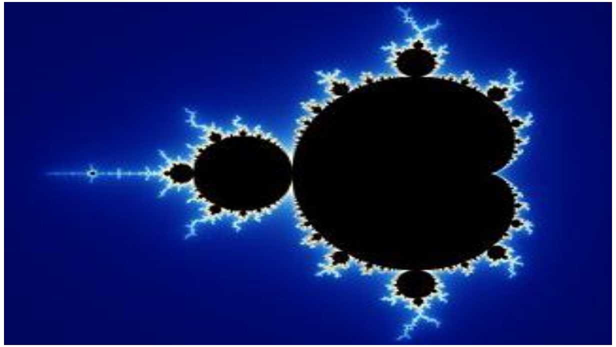
```
for (int i = 0; i < max_row; i++) {
  for (int j = 0; j < max_col; j++ ) {
    p[i][j] = mandel( Complex(scale(i), scale(j)), depth);
  }
}
```

For each point, is  z = z*z + c  bounded?

# Mandelbrot Speedup

## Intel® Threading Building Blocks (Intel® TBB)

```
int mandel(Complex c, int max_count) {
  int count = 0; Complex z = 0;
  for (int i = 0; i < max_count; i++) {
    if (abs(z) >= 2.0) break;
    z = z*z + c; count++;
  }
  return count;
}
```



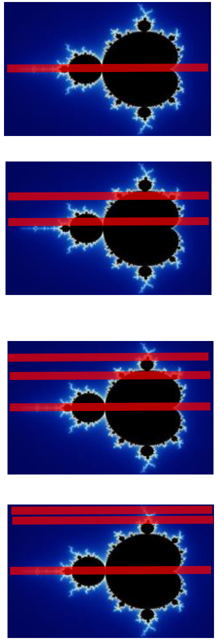
Task is a function object

Parallel algorithm

```
parallel_for( 0, max_row,
  [&](int i) {
    for (int j = 0; j < max_col; j++)
      p[i][j]=mandel(Complex(scale(i),scale(j)),depth);
  }
);
```

Use C++ lambda functions to define function object in-line

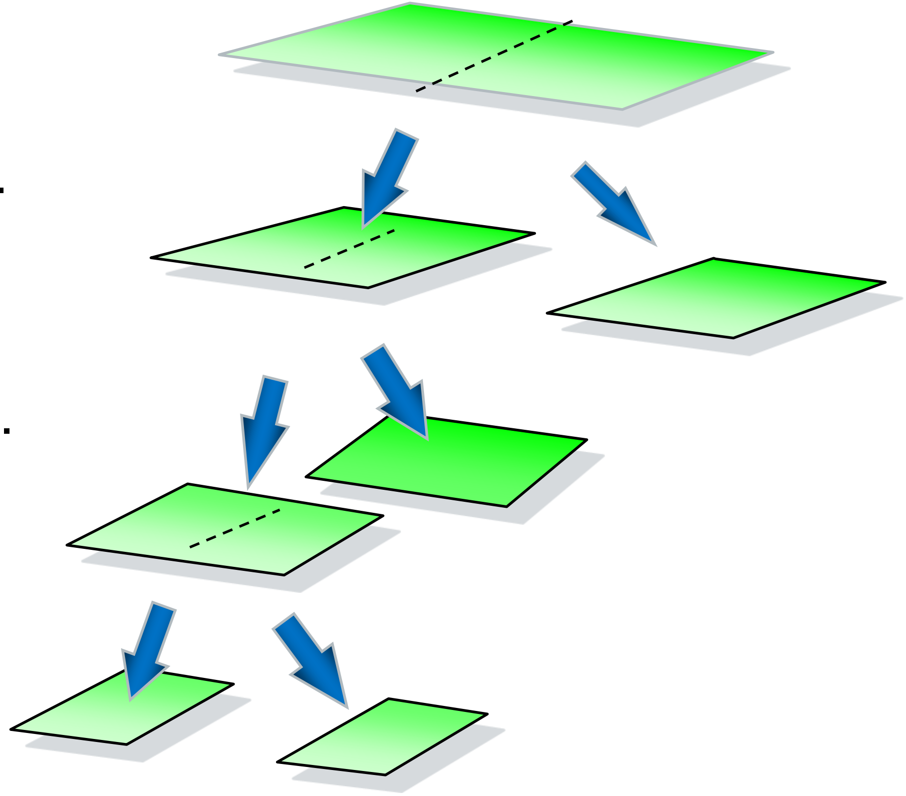
# A parallel_for recursively divides the range into subranges that execute as tasks - Intel® Threading Building Blocks (Intel® TBB)



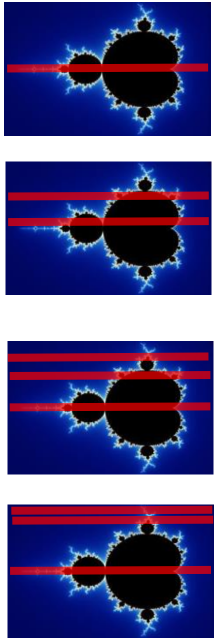
Split range...

.. recursively...

...until ≤ grainsize.



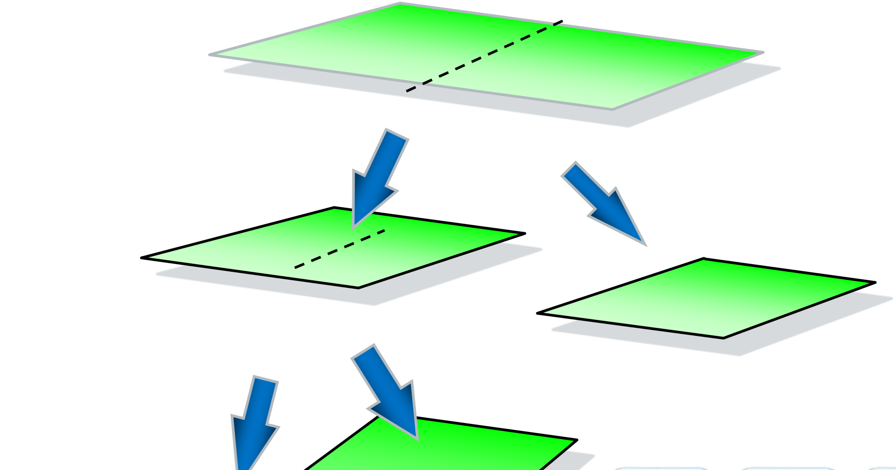
# A parallel_for recursively divides the range into subranges that execute as tasks - Intel® Threading Building Blocks (Intel® TBB)

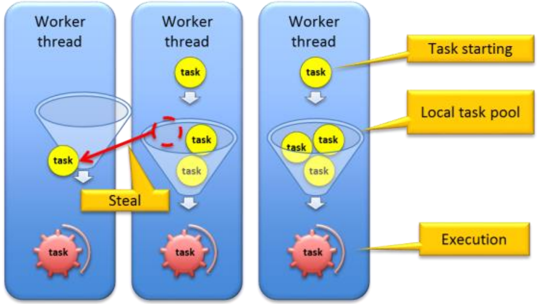


Split range...

.. recursively...

...until ≤ grainsize.

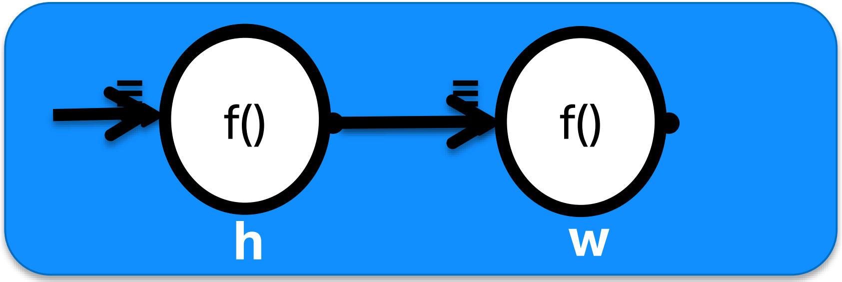




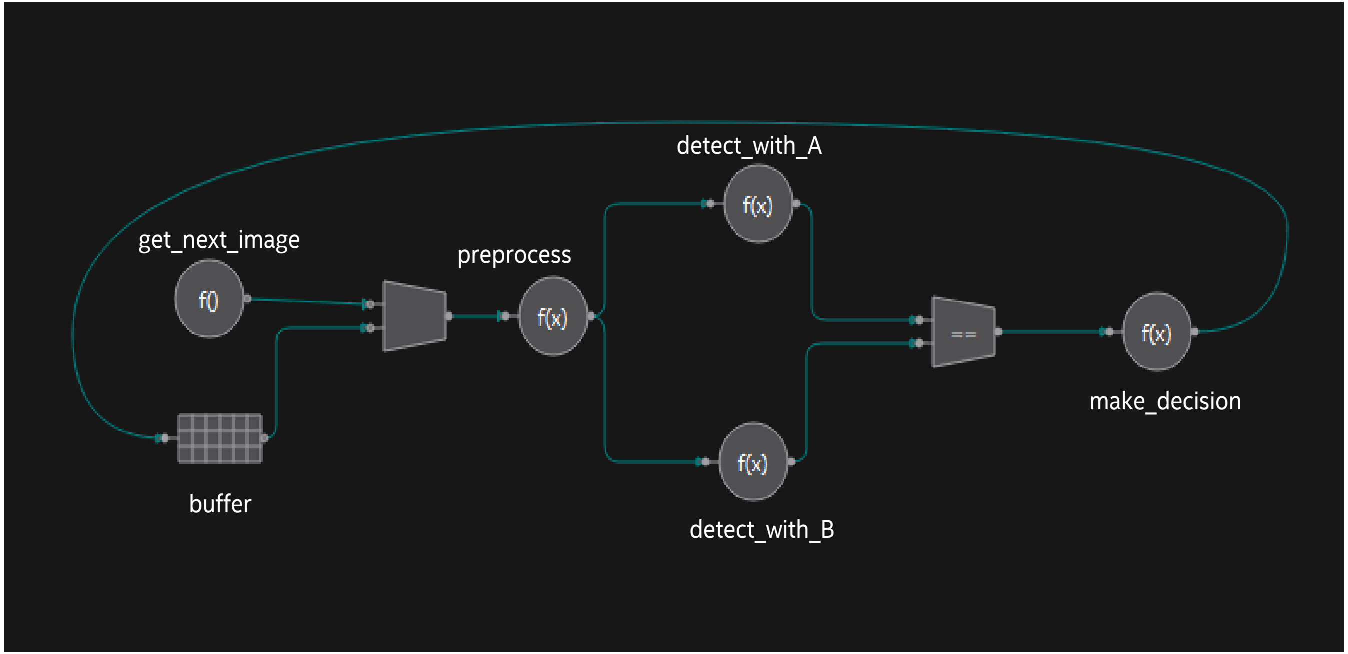
# Flow Graph Hello World Example

## Users create nodes and edges, interact with the graph and wait for it to complete

```
tbb::flow::graph g;

tbb::flow::continue_node< tbb::flow::continue_msg >
  h( g, []( const continue_msg & ) { std::cout << "Hello "; } );
tbb::flow::continue_node< tbb::flow::continue_msg >
  w( g, []( const continue_msg & ) { std::cout << "World\n"; } );
tbb::flow::make_edge( h, w );

h.try_put(continue_msg());

g.wait_for_all();
```

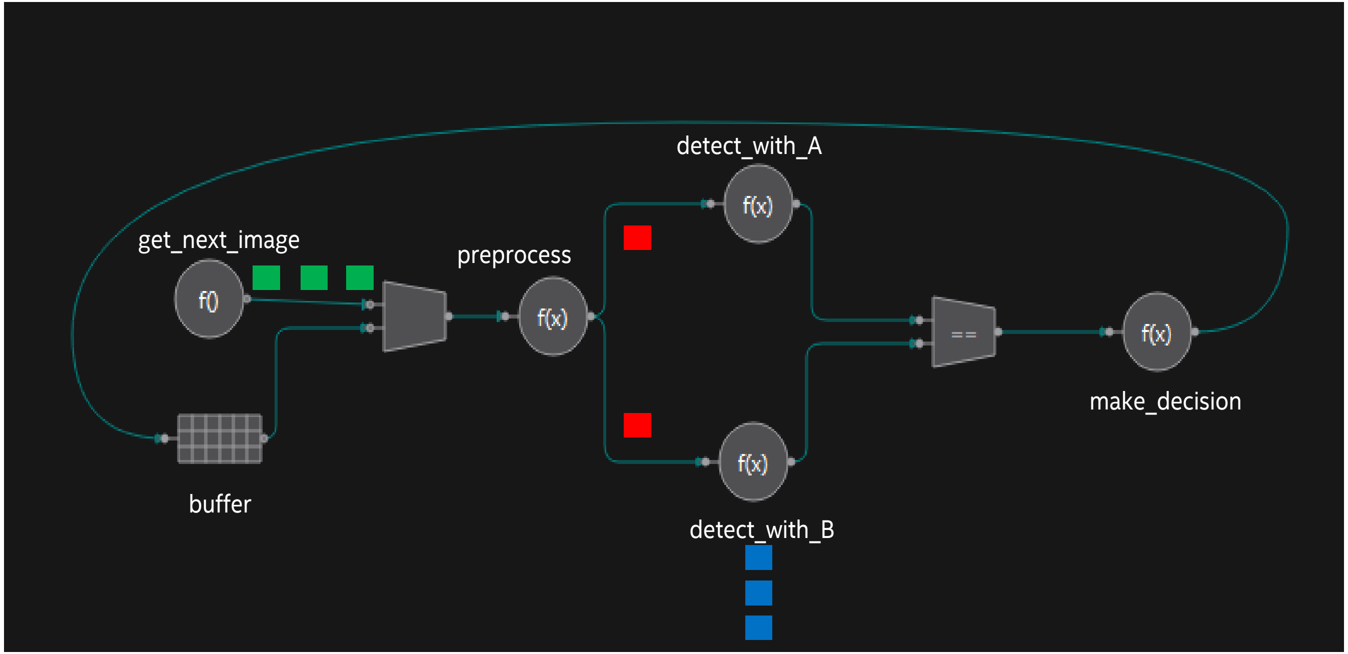


# An example feature detection algorithm




Can express pipelining, task parallelism and data parallelism

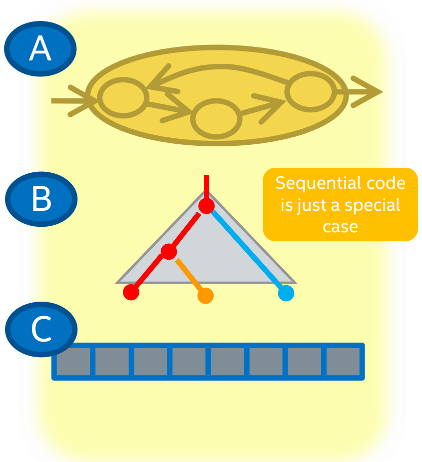
# An example feature detection algorithm




Can express pipelining, task parallelism and data parallelism

*And supports nested parallelism with Intel TBB, OpenMP*,*
*Intel® Cilk™ Plus, Intel® Math Kernel Library (Intel® MKL), etc…*

# CPU Programming Model Hierarchy




- **Message Driven (TBB Flow Graph)**
  Uses same resources/scheduler as (B) since (A) is just a another hierarchical layer
- **Fork Join / Tasking (TBB Tasks)**
  Tolerant of unanticipated CPU loads and support efficient composition
- **SIMD**
  Requires compiler support. New standardization proposal for parallel STL in C++ will integrate this layer into the same software stack.

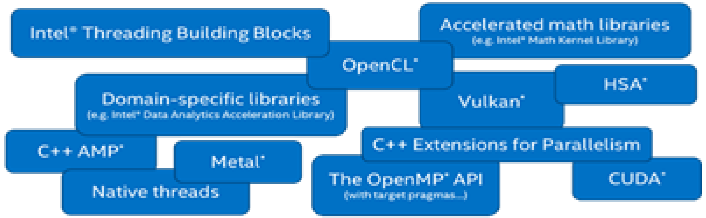
Intel® Threading Building Blocks (Intel® TBB) is the C++ library that provides what is needed for the **Message Driven** and **Fork Join / Tasking** layers

# Use all your available compute resources across HW and SW through Intel TBB

**Hardware**
Integrated graphics, media, CPU's along with discrete co-processors & accelerators (FPGA's, fixed function devices etc)

**Software**
Other threading as well as domain specific libraries and API's

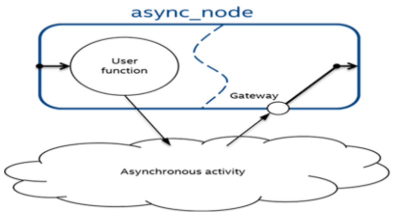
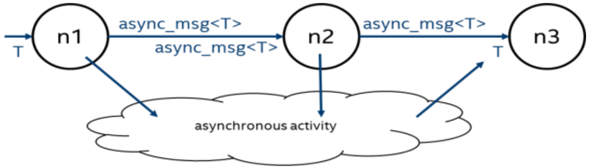
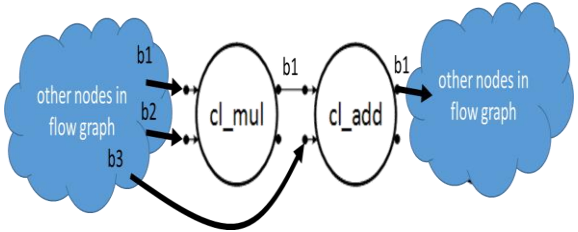
+




**Composability layer with Intel TBB**
One threading engine under all hardware (CPU) side work

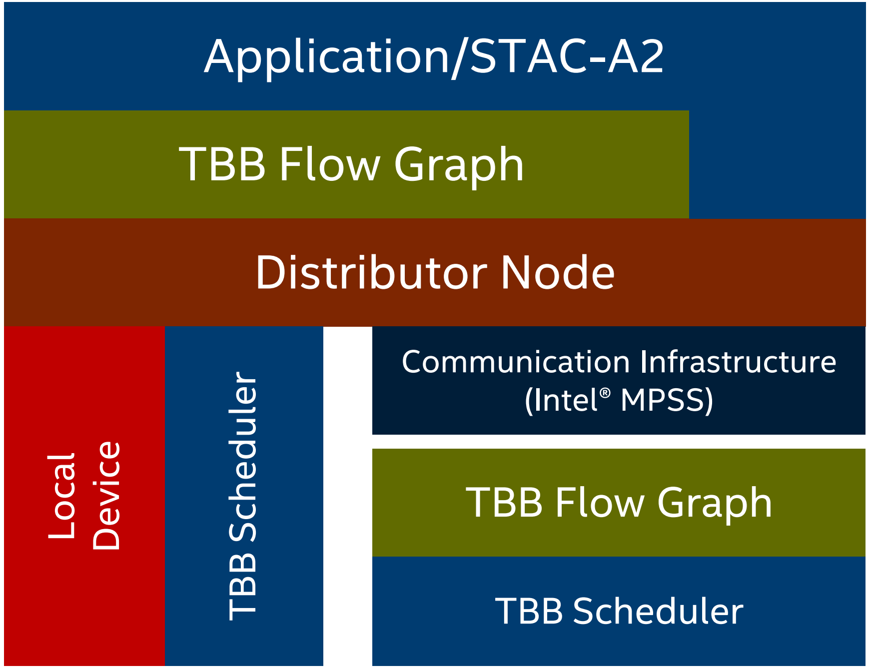
**Co-ordination layer with Intel TBB flow graph**
Be the glue connecting HW & SW, expose parallelism between blocks & simplify integration

# Heterogeneous Features supported today through Intel TBB flow graph API

| Feature | Description | Diagram |
|---|---|---|
| *Async_node* | Allows flow graph to offload data to any asynchronous activity and receive it back to continue execution on CPU. |  |
| *Async_msg* | Enables async communication with chaining across graph nodes. User manages communication. Graph runs on host. |  |
| *Opencl_node* | A specialization of heterogeneous node for openCL. User provide openCL program & Kernel, runtimes handles initialization, buffer management, comms, Graph runs on host |  |

# STAC-A2 Implementation of Intel TBB's Heterogeneous feature

- Compute-intense analytic workload involved in pricing and risk management
- Implemented with Intel® TBB flow graph, TBB parallel algorithms and OpenMP vectorization
- Uses asynchronous support in flow graph, distributor_node, to offload to the Intel® Xeon Phi coprocessor
- Using a token-based system enables dynamic load balancing between CPU and coprocessor
- Which Greeks are calculated on which resource is determined dynamically based on resource availability




https://stacresearch.com/news/2015/11/03/stac-report-stac-a2-system-dual-xeon-phi-cards

# Legal Disclaimer & Optimization Notice



**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

intel®

intel®

experience what's inside™