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**
- Intel Atom
- Intel Core i3
- Intel Core i5
- Intel Core i7
- Intel Xeon

**Supports**
- Windows
- Linux
- Android

**Addresses**
- HPC
- Cloud Computing
- Embedded Systems

**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

Find out more at: [http://software.intel.com/intel-tbb](http://software.intel.com/intel-tbb)
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.
### Rich Feature Set for Parallelism

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

<table>
<thead>
<tr>
<th>Generic Parallel Algorithms</th>
<th>Flow Graph</th>
<th>Concurrent Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient scalable way to exploit the power of multi-core without having to start from scratch.</td>
<td>A set of classes to express parallelism as a graph of compute dependencies and/or data flow</td>
<td>Concurrent access, and a scalable alternative to containers that are externally locked for thread-safety</td>
</tr>
</tbody>
</table>

#### Task Scheduler
- Sophisticated work scheduling engine that empowers parallel algorithms and the flow graph

#### 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

#### Memory Allocation
- Scalable memory manager and false-sharing free allocators

#### 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

---

*Optimization Notice*

Copyright © 2017, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
## Features and Functions List

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

#### 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_rwlock
- speculative_spin_mutex
- speculative_spin_rwlock
- queuing_mutex
- queuing_rwlock
- null_mutex
- null_rwlock
- 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)
Excellent Performance Scalability with Intel® Threading Building Blocks 2017 on Intel® Xeon® Processor

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/technology/resources/benchmark_limitations.htm](http://www.intel.com/technology/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

Copyright © 2015, Intel Corporation. All rights reserved.
* Other names and brands may be claimed as the property of others.
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.mpspp_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.

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 #20110604.
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

```c
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;
}
```

```c
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);
    }
}
```
Mandelbrot Speedup
Intel® Threading Building Blocks (Intel® TBB)

```cpp
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;
}
```

Parallel algorithm

```cpp
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);
    });
```

Task is a function object

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

```cpp
// Create graph
auto g = tbb::flow::graph();

// Create nodes and edges
auto h = tbb::flow::continue_node<tbb::flow::continue_msg>(g, [] (const tbb::flow::continue_msg & msg) {
  std::cout << "Hello ";
});
auto w = tbb::flow::continue_node<tbb::flow::continue_msg>(g, [] (const tbb::flow::continue_msg & msg) {
  std::cout << "World\n";
});

// Make edge between nodes
auto e = tbb::flow::make_edge(h, w);

// Send a message to node h
h.try_put(tbb::flow::continue_msg());

// Wait for all tasks to complete
auto r = 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 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
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

---

**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