Parallel Programming Models

- Intel® Cilk Plus Tasking
- Intel® Threading Building Blocks

Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Agenda

- Introdcution
- Intel® Cilk Plus Tasking
- Intel® Threading Building Blocks
- Summary

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

A Family of Parallel Programming Models

Developer Choice

Intel® Cilk™ Plus	Intel® Threading Building Blocks	Domain-Specific Libraries	Established Standards	Research and Development
C/C++ language extensions to simplify parallelism *Open sourced* *Also an Intel product*	Widely used C++ template library for parallelism *Open sourced* *Also an Intel product*	Intel® Integrated Performance Primitives Intel® Math Kernel Library	Message Passing Interface (MPI) OpenMP* Coarray Fortran OpenCL*	Intel® Concurrent Collections Offload Extensions Intel® Array Building Blocks Intel® SPMD Parallel Compiler

Choice of high-performance parallel programming models

- Libraries for pre-optimized and *parallelized functionality*
- Intel® Cilk™ Plus and Intel® Threading Building Blocks supports composable parallelization of a wide variety of applications.
- OpenCL* addresses the needs of customers in specific segments, and provides developers an additional choice to maximize their app performance
- MPI supports distributed computation, combines with other models on nodes

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Parallelism of Modern Compute Platforms
Available on 4 Levels

- **ILP** - Instruction Level Parallelism
  - Pipelined Execution
  - Super-scalar execution
- **DLP** - Data Level Parallelism
  - SIMD (Single Instruction Multiple Data) vector processing
  - Implemented via vector registers and instructions
- **TLP** – Thread/Task-Level Parallelism
  - Hardware support for hyper-threading
  - Multi-core architecture
  - Cache-coherent multiple sockets
  - Heterogenous environments (host+accelerator, host+GPU)
- **CLP** - Cluster Level Parallelism
  - Multiple platforms connected via interconnection network
  - No hardware-supported cache coherence

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

How can Developer Exploit these Levels ?

ILP

- Almost transparent to developer
- Taken care of by compiler

DLP

- Implicit: Automatic Vectorization
- Explicit: Intel® Cilk Plus data parallel features, Fortran array sections

TLP

- Pthreads, Win32 Threads, ….
- OpenMP*,
- Intel® Cilk Plus tasking,
- Intel® Threading Building Blocks
- OpenCL*
- FORTRAN specific: Coarray Fortran, DOCONCURRENT

CLP

- Coarray Fortran
- Message Passing Interface (MPI)

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

What should we Expect from a TLP-Model ?

- **Composability**
  - Never think about “mapping to hardware” (core etc) - use tasks instead of threads
- **Scalability**
- **Simplicity**
- **Performance**
- **Portability**
- **Availability**
- **Support for correctness checking**

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Intel® Cilk Plus Tasking

Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Intel® Cilk™ Plus

Task Level Parallelism

Simple Keywords

Set of keywords, for expression of task parallelism:

```
cilk_spawn
cilk_sync
cilk_for
```

Reducers (Hyper-objects)

Reliable access to nonlocal variables without races

```
cilk::reducer_opadd<int> sum(3);
```

Data Level Parallelism / Explicit Vector Programming

Array Notation

Provide data parallelism for sections of arrays or whole arrays

```
mask[:]=[:]<b[:] ? -1:1;
```

SIMD Functions

Define actions that can be applied to whole or parts of arrays or scalars

SIMD Directive

User mandated vectorization

Execution Parameters

Runtime system APIs, Environment variables, pragmas

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

What is Cilk Tasking ?

- C\C++ Extension to support task parallelism
- Developed at MIT some 20 years ago
- Commercialized by startup Cilk Arts Inc.
  - Acquired by Intel in 2009
- Very simple model: Three key words only
- Parallel intent not parallel control
- Open-source now
  - GCC: Part of compiler since version 4.9
  - LLVM: Being worked on too

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Programmers view of Cilk Tasking

- Key words

```
#include <cilk/cilk.h>
```

cilk_spawn
cilk_sync
cilk_for

- API

__cilkrts_set_param(“nworkers”, “4”)
__cilkrts_get_nworkers()
__cilkrts_get_total_workers()
__cilkrts_get_worker_number()

- Environment variables

    CILK_NWORKERS, …

- Hyperobects (Reducers, …)

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

The Loop Construct: `cilk_for loop`

- Looks like a normal for loop.

```
cilk_for (int x = 0; x < 1000000; ++x) { … }
```

- Any or all iterations may execute in parallel with one another.
- All iterations complete before program continues.
- Constraints:
  - Limited to a single control variable.
  - Must be able to jump to the start of any iteration at random.
  - Iterations should be independent of one another
- A ‘grain size’ controls when a small iterations space is no longer split (serial execution)

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

cilk_for Examples

```
cilk_for (int x; x < 1000000; x += 2) { … }
```

```
cilk_for (vector<int>::iterator x = y.begin();
          x != y.end(); ++x) { … }
```

```
cilk_for (list<int>::iterator x = y.begin();
          x != y.end(); ++x) { … }
```

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

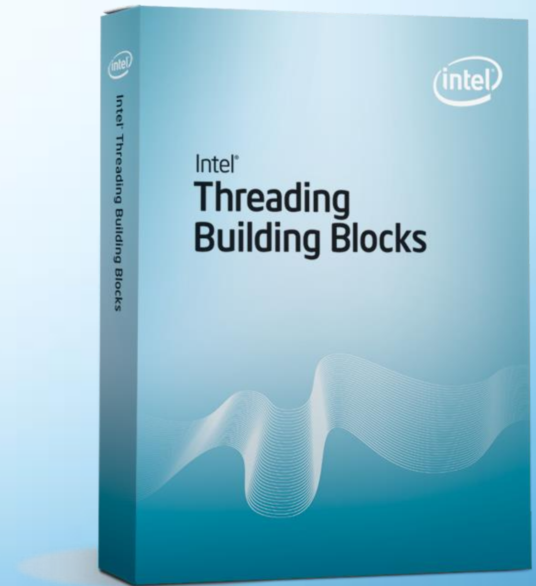
Cilk Summary

- Cilk is a simple extension to C and C++ for shared-memory fork-join parallelism.
- Cilk's serial semantics and simple syntax does not obscure the program logic.
- Cilk's work-stealing scheduler automatically load-balances if there sufficient parallelism.
- Cilk is suitable for both loop and recursive (divide an conquer) parallelism.
- Cilk is an excellent choice for parallelizing both new and legacy C and C++ software.

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Intel® Threading Building Blocks

- Takes care of managing multithreading
  - Pure tasking model
- Runtime library
  - Scalability to available number of threads
  - Scheduler very similar to the one of Cilk
- Cross-platform
  - Windows, Linux, Mac OS* and others
- Dual licensing
  - Open-source release http://threadingbuildingblocks.org/
  - Commercially aligned Intel TBB library releases
    - Part of Composer bundle
    - Premier support
    - Regular updates, and maintenance



Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Intel® Threading Building Blocks

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

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

TBB Flow Graph

Task scheduler
The engine that empowers parallel algorithms that employs task-stealing to maximize concurrency

Thread Local Storage
Scalable implementation of thread-local data that supports infinite number of TLS

Synchronization Primitives
User-level and OS wrappers for mutual exclusion, ranging from atomic operations to several flavors of mutexes and condition variables

Miscellaneous
Thread-safe timers

Threads
OS API wrappers

Memory Allocation
Per-thread scalable memory manager and false-sharing free allocators

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Generic Algorithms

Loop parallelization

parallel_for

parallel_reduce

- load balanced parallel execution
- fixed number of independent iterations

parallel_deterministic_reduce

- run-to-run reproducible results

parallel_scan

- computes parallel prefix

  y[i] = y[i-1] op x[i]

Parallel Algorithms for Streams

parallel_do

- Use for unstructured stream or pile of work
- Can add additional work to pile while running

parallel_for_each

- parallel_do without an additional work feeder

pipeline / parallel_pipeline

- Linear pipeline of stages
- Each stage can be parallel or serial in-order or serial out-of-order.
- Uses cache efficiently

Parallel sorting

parallel_sort

Parallel function invocation

parallel_invoke

- Parallel execution of a number of user-specified functions

Computational graph

flow::graph

- Implements dependencies between tasks
- Pass messages between tasks

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Classical Parallel Algorithm Usage Example

```
#include "tbb/blocked_range.h"
#include "tbb/parallel_for.h"
using namespace tbb;

class ChangeArray{
    int* array;
public:
    ChangeArray (int* a): array(a) {}
    void operator()( const blocked_range<int>& r ) const{
        for (int i=r.begin(); i!=r.end(); i++ ){
            Foo (array[i]);
        }
    }
};

void ChangeArrayParallel (int* a, int n )
{
    parallel_for (blocked_range<int>(0, n), ChangeArray(a),  auto_partitioner());
}

int main (){
    task_scheduler_init init;
    int A[N];
    // initialize array here…
    ChangeArrayParallel (A, N);
    return 0;
}
```

ChangeArray class defines a for-loop body for parallel_for

blocked_range – TBB template representing 1D iteration space

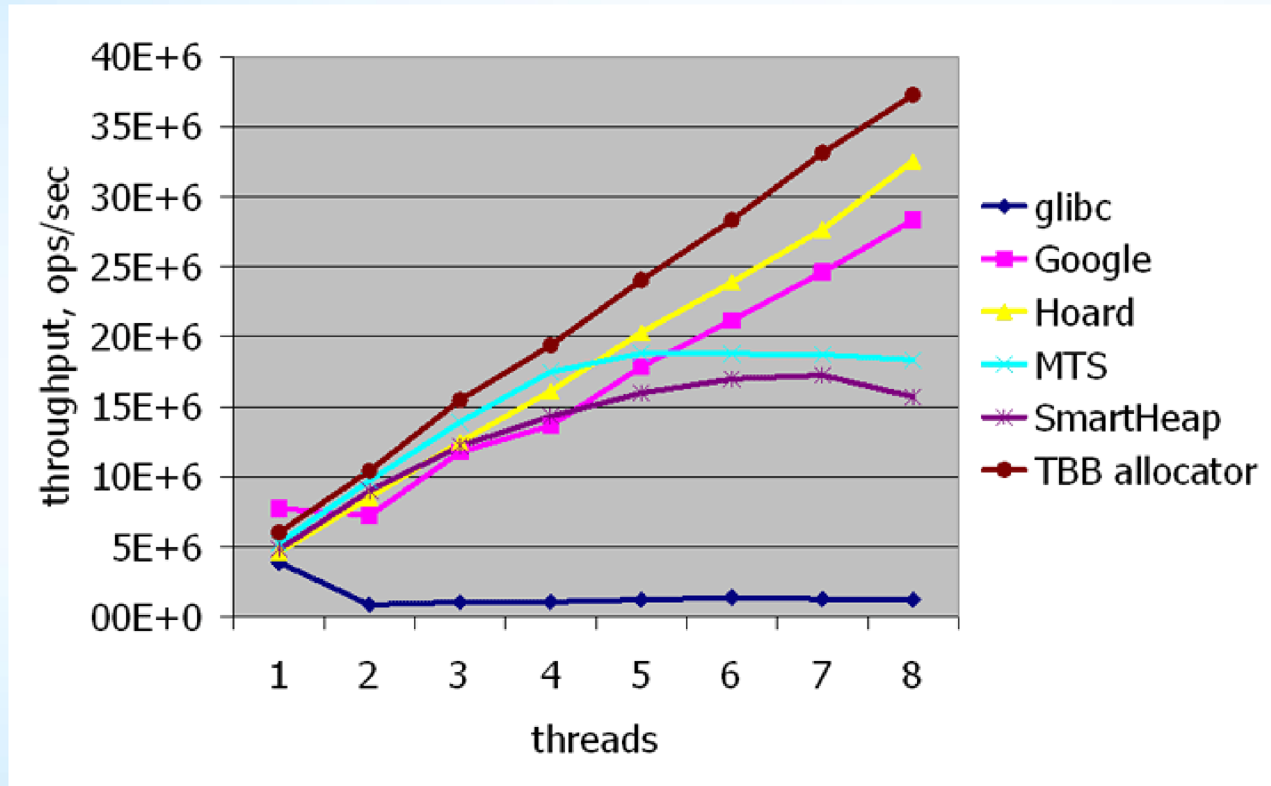
As usual with C++ function objects the main work is done inside operator()

A call to a template function parallel_for<Range, Body>: with arguments
Range → blocked_range
Body → ChangeArray

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Optimization Notice

Intel® Threading Building Blocks scalable memory allocator improves parallel performance



The throughput, in allocations per second, of the Larson benchmark running on 1 through 8 threads

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Flexible Memory Pools

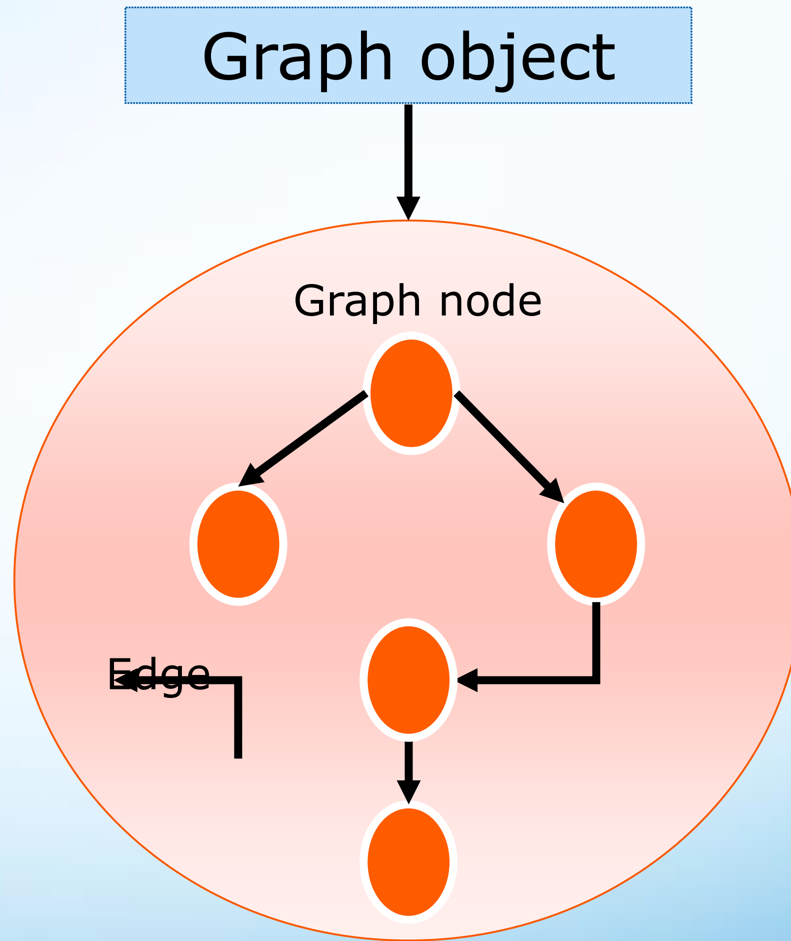
```
#define TBB_PREVIEW_MEMORY_POOL 1
#include "tbb/memory_pool.h"
...
tbb::memory_pool<std::allocator<char> > my_pool();
void* my_ptr = my_pool.malloc(10);
void* my_ptr_2 = my_pool.malloc(20);
…
my_pool.recycle();  // destructor also frees everything
```

```
#define TBB_PREVIEW_MEMORY_POOL 1
#include "tbb/memory_pool.h"
...
char buf[1024*1024];   // provide your own memory area

tbb::fixed_pool my_pool(buf, 1024*1024);
void* my_ptr = my_pool.malloc(10);
my_pool.free(my_ptr);}
```

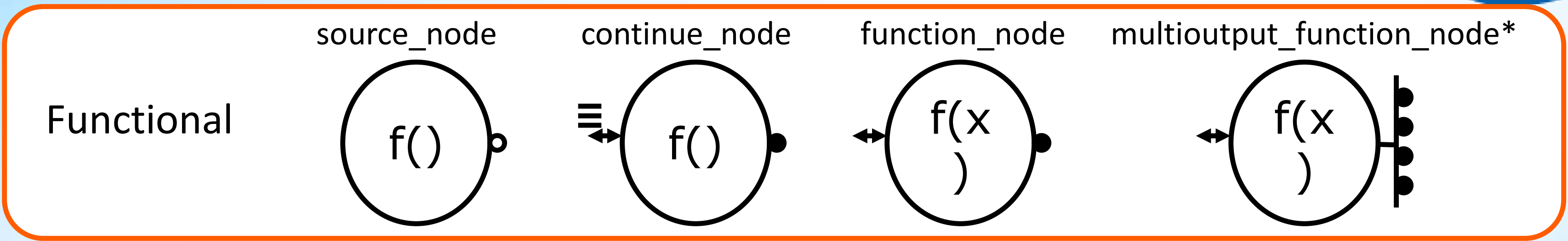

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

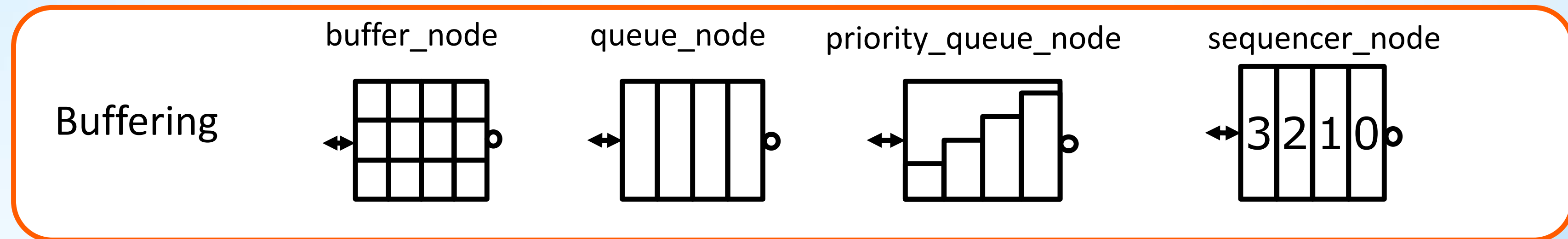
An Flow Graph consists of a Graph Object, Nodes and Edges

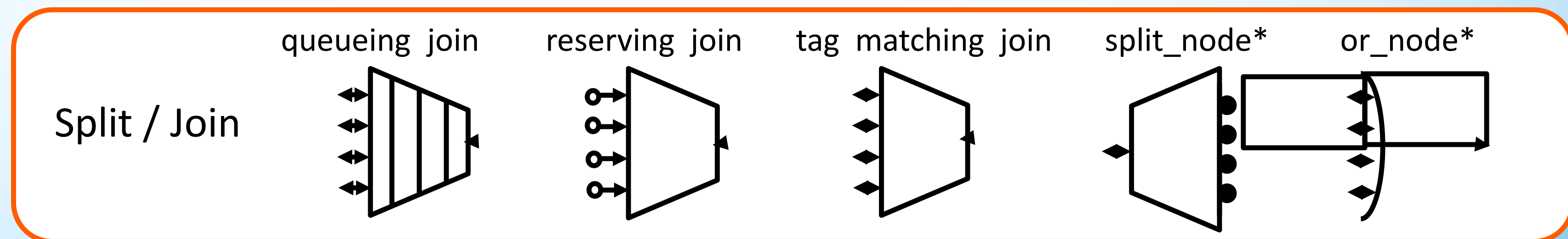


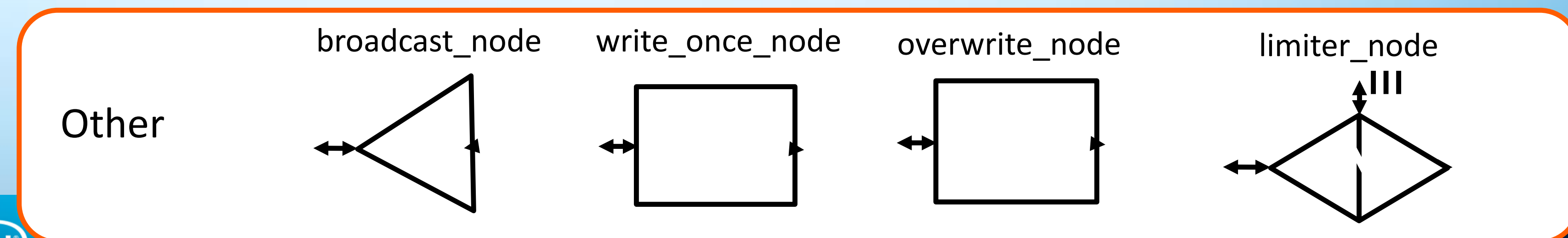
Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

The Flow Graph has several Node Types:



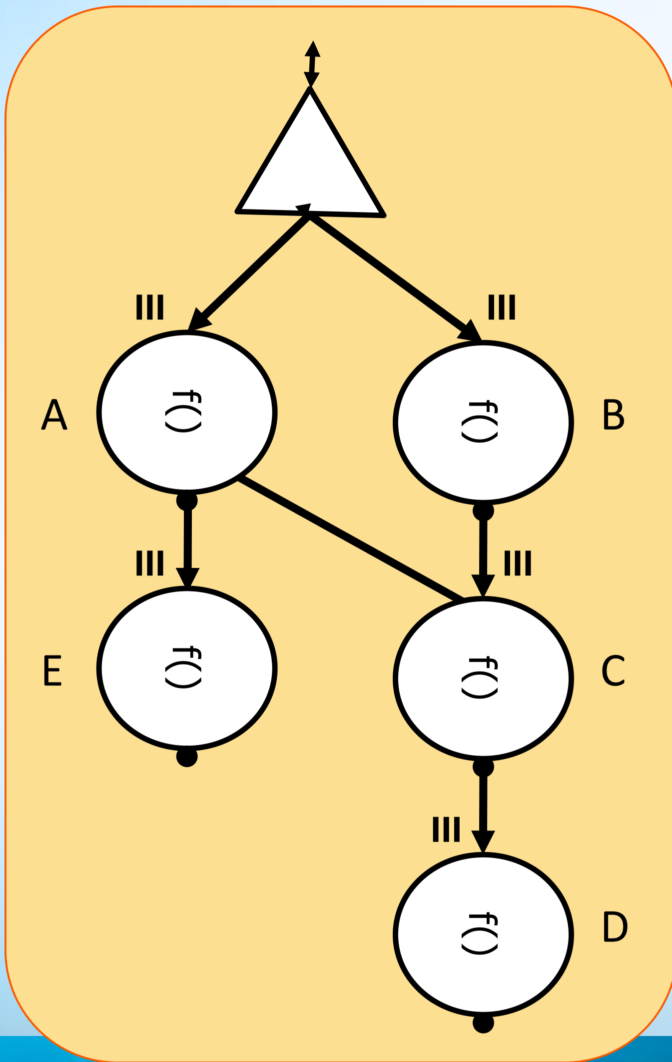






Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Tiny Example:
A dependence graph




```
struct body {
  std::string my_name;
  body( const char *name ) :
    my_name(name) {}
    void operator()( continue_msg ) const {
   printf("%s\n", my_name.c_str());
}
};


int main() {
  graph g;
  broadcast_node< continue_msg > start;
  continue_node< continue_msg > a( g, body("A") );
  continue_node< continue_msg > b( g, body("B") );
  continue_node< continue_msg > c( g, body("C") );
  continue_node< continue_msg > d( g, body("D") );
  continue_node< continue_msg > e( g, body("E") );
  make_edge( start, a );
  make_edge( start, b );
  make_edge( a, c );
  make_edge( b, c );
  make_edge( c, d );
  make_edge( a, e );
  for (int i = 0; i < 3; ++i ) {
    start.try_put( continue_msg() );
    g.wait_for_all();
  }
  return 0;
}
```

Copyright© 2015, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Legal Disclaimer

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.

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, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.
*Other names and brands may be claimed as the property of others.

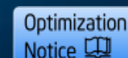
Copyright © 2014. Intel Corporation.

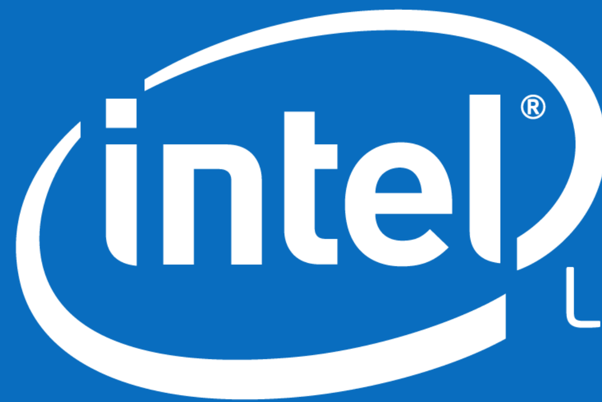
Optimization Notice

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 brands and names are the property of their respective owners.




intel®
Leap ahead™