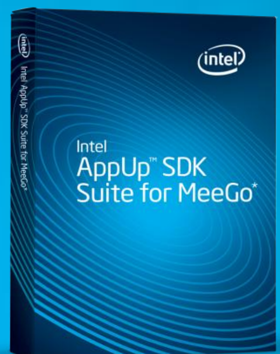
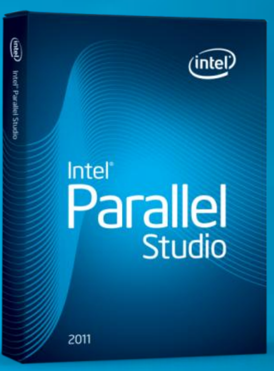
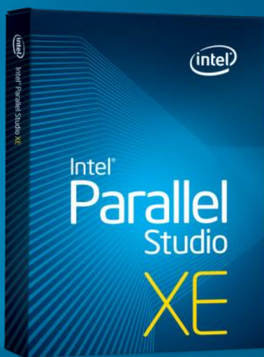


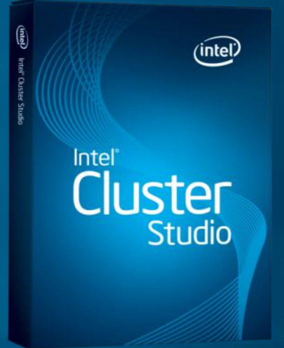
Efficient
Performance



Essential
Performance



Advanced
Performance



Distributed
Performance

Intel® Parallel Studio XE

Hans Pabst, Developer Product Division

Facing the Multicore-Challenge II

September 28-30, 2011

Motivation: Performance

"Parallel hardware needs parallel programming"

Programmability!

Hardware Potential

Application Performance

Parallelism

No Parallelism

GHz Era

Multicore Era

Manycore Era

Performance

Time

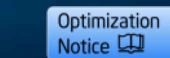
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Intel® Parallel Studio XE

Phase	Tool	Usage	Benefit
Compilation Debugging	Intel® Composer XE	C/C++ and Fortran Compiler, Performance Libraries and parallel programming models	Strong step towards higher performance (ad-hoc and in the future), additional robustness and safety
Verification Correctness	Intel® Inspector XE	Debugging (memory access and thread usage) for better code / application quality	Higher productivity, early or continuous quality assurance (GUI+CLI)
Analysis Tuning	Intel® VTune™ Amplifier XE	Profiler to inspect hardware events (counter), scalability etc.	Avoids work based on guesses, combines ease of use with deep insight (GUI+CLI)

Powerful compilers, verification and performance analysis.

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



Familiar Programming

Source

Compilers
Libraries
Parallel Models

Multicore

Many-core

Cluster

One source base, tuned to many targets.

Multicore CPU

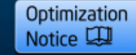
Multicore CPU

Intel® MIC Architecture Co-Processor

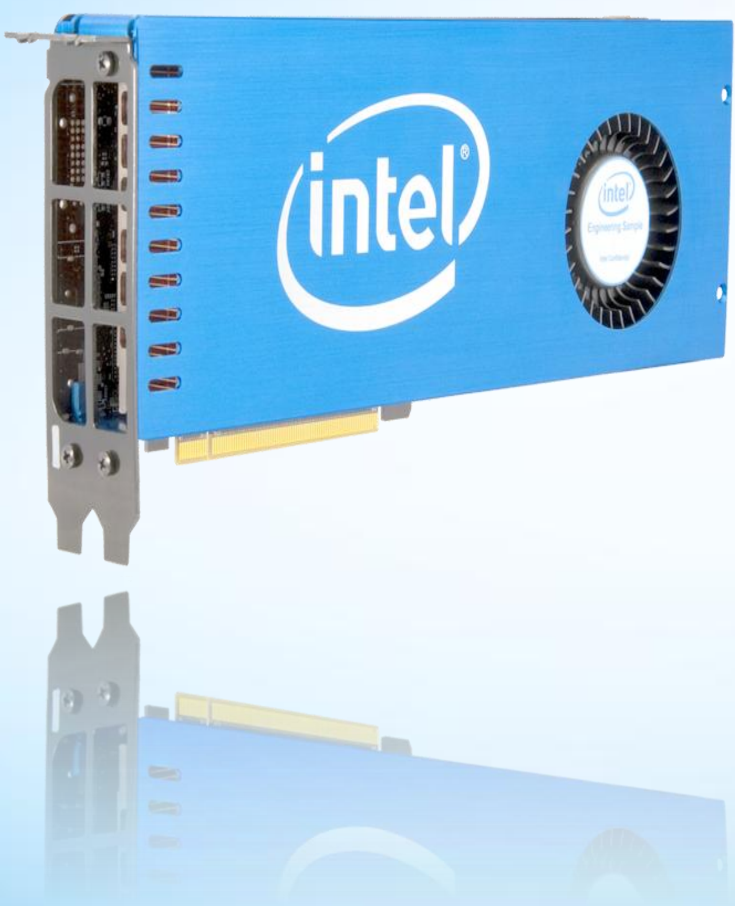
Multicore Cluster

Multicore and Many-core Cluster

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

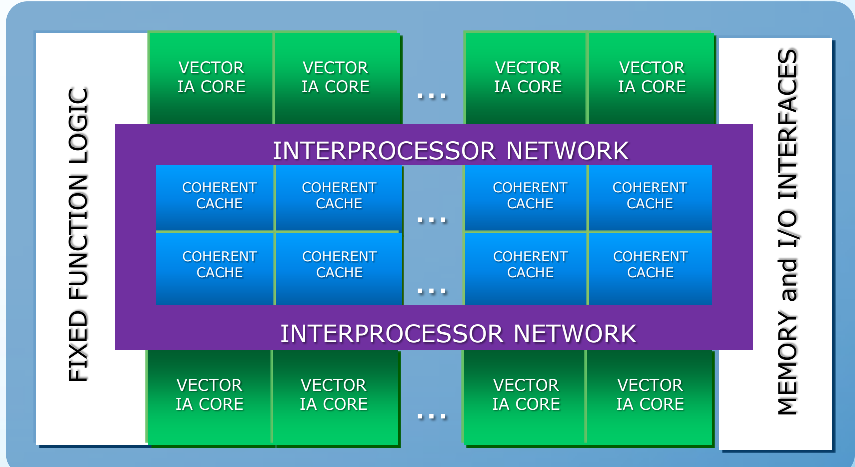


Intel® Many Integrated Core (MIC) Co-Processor Architecture



Knights Ferry Software Development Platform

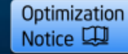
- Up to 32 cores, 128 threads
- 512-bit SIMD support
- Fully coherent cache
- Up to 2 GB GDDR5 memory
- Latest Intel SW developer products



First Intel® MIC product

- Codenamed “Knights Corner”
- Planned for production on Intel’s 22 nm 3D Tri-Gate transistor technology

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



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	Widely used C++ template library for parallelism	Intel® Integrated Performance Primitives	Message Passing Interface (MPI)	Intel® Concurrent Collections
		Intel® Math Kernel Library	OpenMP*	Offload Extensions
			Coarray Fortran	Intel® Array Building Blocks
Open sourced	*Open sourced*		OpenCL*	Intel® SPMD Parallel Compiler
Also an Intel product	*Also an Intel product*			

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

Performance Optimization Steps

Shown steps enable to scale forward to many-core co-processors.

Baseline
Recompilation of the existing code.

Intel® Compiler
- Performance comparison with other compilers.

Intel® Libraries
Identify fixed functionality and employ optimized code, threads, and (with Intel® MKL) multiple nodes.

Intel® MKL
- Statistics (VSL)
- BLAS
- etc.

Intel® IPP
- Multi-media
- etc.

Multithreading
Achieve scalability across multiple cores, sockets, and nodes.

Intel® Compiler
- Auto/guided par.
- Intel Cilk Plus
- OpenMP*

Intel® Threading Building Blocks

Intel® Cluster Studio
- Cluster tools
- MPI

Intel® ArBB VM
- Unified model for SIMD and threads

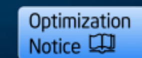
Vectorization
Make use of SIMD extensions, e.g. Intel® AVX.

Intel® Compiler
- Optimization hints
- Intel Cilk Plus
  - #pragma simd
  - array notation
  - elemental fn.
- Intrinsics
- Assembler

Intel® ArBB VM
- Unified model for SIMD and threads

The order of the steps is suggested to be based on a performance analysis.

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



Intel® Compiler

Intel® C/C++ Compiler Version 12.1
Intel® Fortran Compiler Version 12.1

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

Key Files Supplied with Compilers

source /opt/intel/composerxe/bin/compilervars.sh intel64

source /opt/intel/composerxe/bin/compilervars.sh ia32

Windows*

- Intel compiler
  - `icl.exe, ifort.exe`: C/C++ compiler, Fortran compiler
  - `icl.cfg, ifort.cfg`: Default compiler options
  - `compilervars.bat`: Setup command window build environment (C/C++ and Fortran)
- Linker driver
  - `xilink.exe`: Invokes `link.exe`
- Intel include files, libraries
- Re-distributable files
  - `<install-dir>\ComposerXE-2011\redist\`

Linux*, Mac OS* X

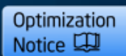
- Intel compiler
  - `icc, ifort`: C/C++ compiler, Fortran compiler
  - `compilervars.(c)sh`: Source scripts to setup the complete compiler/debugger/libraries environment (C/C++ and Fortran)
- Linker driver
  - `xild`: Invokes `ld`
- Intel include files, libraries
- Intel debugger
  - `idbc` (Command Line) Debugger, `idb` (GUI) Debugger (Linux* only)

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

Optimization Notice

Common Optimization Switches

	Windows*	**Linux***
Disable optimization	/Od	-O0
Optimize for speed (no code size increase)	/O1	-O1
Optimize for speed (default)	/O2	-O2
High-level loop optimization	/O3	-O3
Create symbols for debugging	/Zi	-g
Multi-file inter-procedural optimization	/Qipo	-ipo
Profile guided optimization (multi-step build)	/Qprof-gen /Qprof-use	-prof-gen -prof-use
Optimize for speed across the entire program	/fast (same as: /O3 /Qipo /Qprec-div- /QxHost)	-fast (same as: -ipo –O3 -no-prec-div -static -xHost)
OpenMP 3.0 support	/Qopenmp	-openmp
Automatic parallelization	/Qparallel	-parallel

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


Code Generation Options

{L&M} -x<extension> {W}: /Qx<extension>

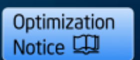
- Targeting Intel® processors - specific optimizations for Intel® processors
- Compiler will try to make use of all instruction set extensions up to and including <extension>; for Intel® processors only!
- Processor-check added to main-program
- Application will not start (will display message), in case feature is not available

{L&M}: -m<extension> {W}: /arch:<extension>

- No Intel processor check; does not perform Intel-specific optimizations
- Application is optimized for and will run on both Intel and non-Intel processors
- Missing check can cause application to fail in case extension not available

{L&M}: -ax<extension> {W}: /Qax<extension>

- Multiple code paths – a 'baseline' and 'optimized, processor-specific' path(s)
- Optimized code path for Intel® processors defined by <extension>
- Baseline code path defaults to –msse2 (Windows: /arch:sse2); can be modified by –m or –x (/Qx or /arch) switches

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


Code Generation Options

The default is –msse2 (Windows: /arch:sse2)

- Activated implicitly for –O2 or higher
- Implies the need for a target processor with Intel® SSE2
- Use –mia32 (/arch:ia32) for 32-bit processors without SSE2 (e.g. Intel® Pentium™ 3) to adjust baseline code path

Special switch –xHost (Windows: /QxHost)

- Compiler checks host processor and makes use of latest instruction set extension available
- Avoid for builds being executed on multiple, unknown platforms

Multiple extensions can be used in combination:
–ax<ext1>,<ext2> (Windows: /Qax<ext1>,<ext2>)

- Can result in more than 2 code paths (incl. baseline code path)
- Use –mia32 (/arch:ia32) for 32-bit processors without SSE2 (e.g. Intel® Pentium™ 3) to adjust baseline code path

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

NUMA

- OpenMP options (Intel linker driver)

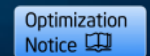
  -par-affinity=KMP_SETTING

  -Qpar-affinity:KMP_SETTING

  KMP_SETTING, e.g. “granularity=thread,compact”

  (similar to KMP_AFFINITY environment option)

- What else?
  - Linux*: “first touch rule”, libnuma, numactrl, etc.
  - Windows*: VirtualAlloc + “first touch”

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


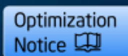
What is Vectorization?

- Hardware
  - SIMD registers
    SSE: 128 bit
    AVX: 256 bit
    MIC: 512 bit
  - SIMD instructions
    Intel® SSE
    Intel® AVX
    …
- Programming (SW)
  - Auto-vectorization
  - Hinting vectorization
  - Explicit vectorization

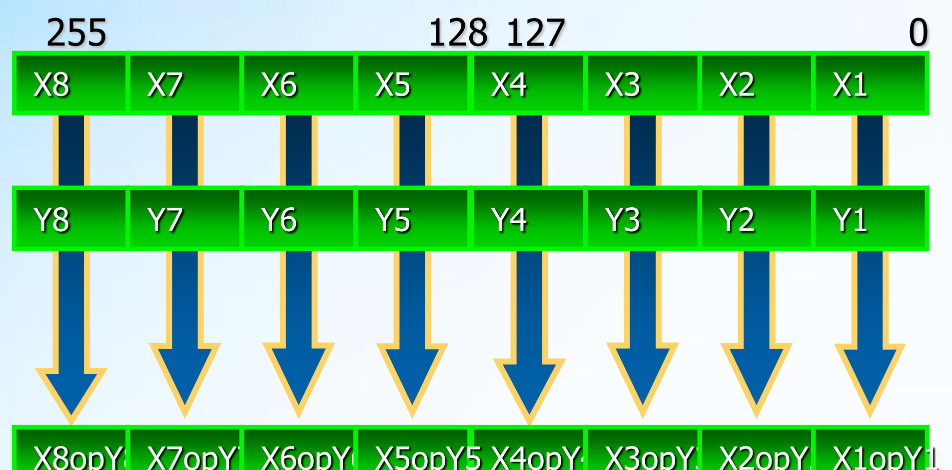
Level	Description
Grid*	Group of clusters communicating through internet
Cluster*	Group of computers communicating through fast interconnect
Node	Group of processors communicating through shared memory
Socket	Group of cores communicating through shared cache
Core	Group of functional units communicating through registers
Hyperthreads	Group of thread contexts sharing functional units
Superscalar	Group of instructions sharing functional units
Pipeline	Sequence of instructions sharing functional units
Vector	Single instruction using multiple functional units

* Vectorization is completely non-interfering with MPI processes/threads. Provides up to 8x (AVX) using SP.

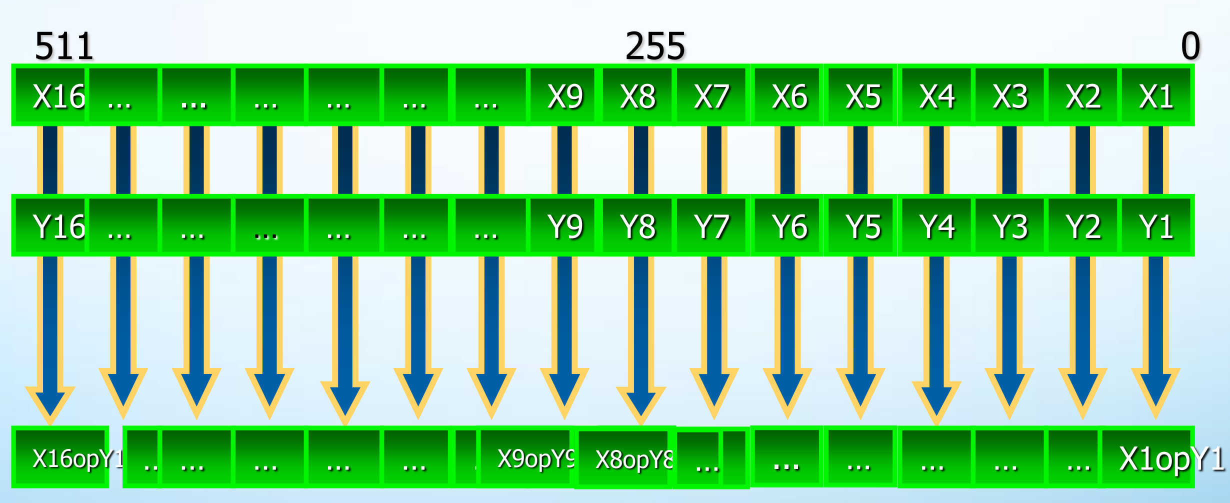
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.



Extending In-Core Data Parallelism…



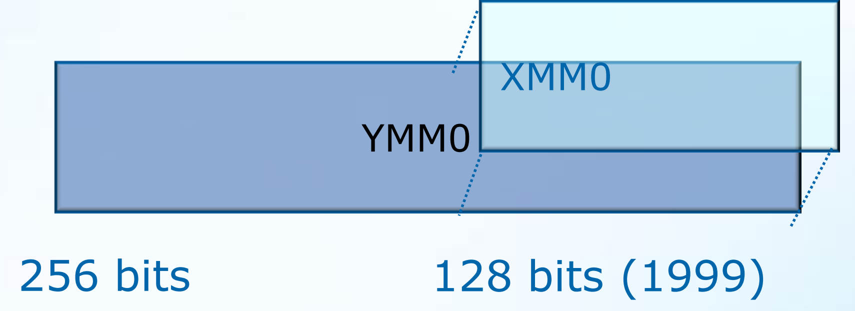
Intel® AVX
Vector size: 256bit
Data types: 32 and 64 bit floats
VL: 4, 8, 16
Sample: Xi, Yi 32 bit int or float



Intel® MIC arch.
Vector size: 512bit
Data types:
32 and 64 bit ints
32 and 64bit floats
(some support for
16 bits floats)
VL: 8,16
Sample: 32 bit float

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

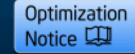
Intel® AVX Key Features



Key Feature	Benefit
Wider Vectors – Increased from 128 to 256 bit – Two 128-bit load ports	Up to twice the FLOP/s (peak) compared to SSE; good power efficiency
Enhanced Data Rearrangement – broadcast, masked loads – permute data	Organize, access and pull only necessary data more quickly and efficiently
Three and four Operands – Non-destructive syntax – AVX 128, and AVX 256	Fewer register copies, better register use for both vector and scalar code
Flexible support for unaligned memory access	More opportunities to fuse load and compute operations
Extensible new opcode (VEX)	Code size reduction

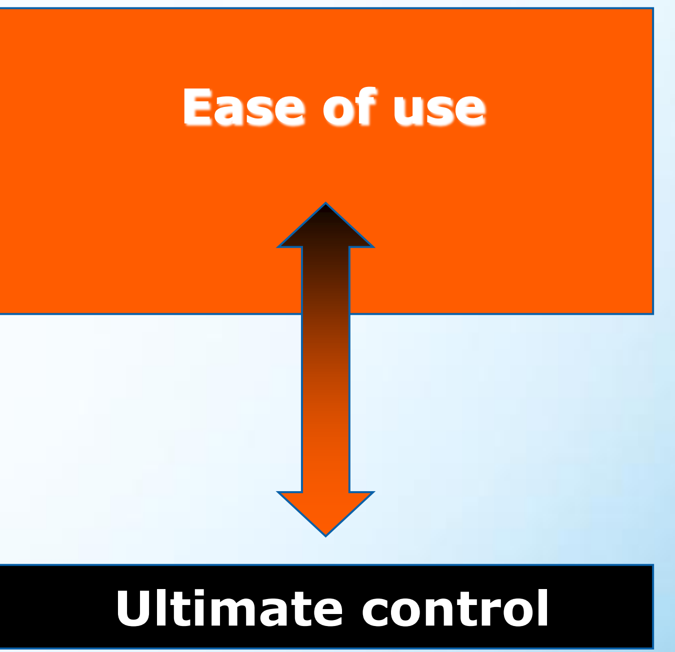
AVX 128 vs. AVX 256: the upper part of the register is zeroed out (zeroing idioms).

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



How to Vectorize?

Fully automatic vectorization
Built-in vectors (explicit)
Auto vect. hints (#pragma ivdep, …)
SIMD intrinsic class (F32vec4 add)
Vector intrinsic (mm_add_ps())
Assembler code (addps)



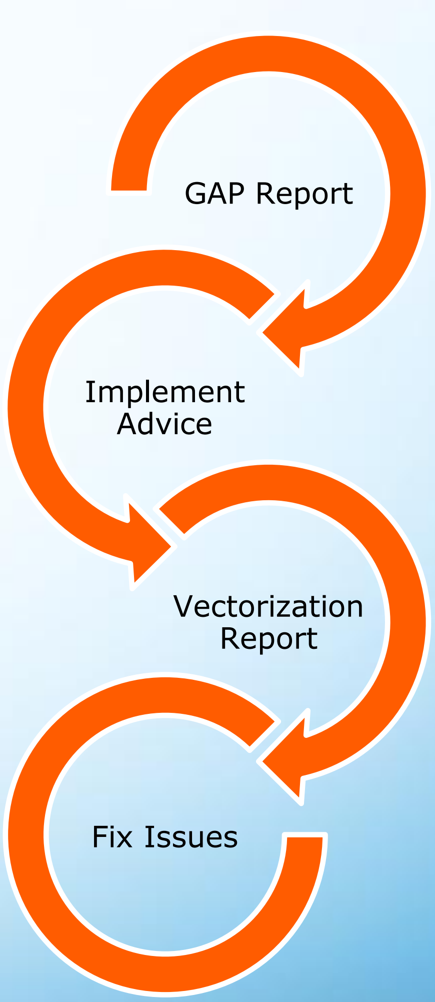
Ultimate control

Auto-vect., explicit constructs (built-in vect., hints)
- Multiple auto-generated code paths possible (-ax, /Qax)
- Forward-scaling

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

How to achieve auto-vectorization?

- Guided Auto-Parallelization (GAP)
  (Compiler-as-a-tool)
  - User/advice-oriented terminology
- Vectorization report
  - Compiler terminology
  - Complete
- Fix vectorization blockers
  - May require user-mandated vectorization
  - Adjust the code, e.g. combine two loops



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

Compiler as a Tool

Application Source C/C++/Fortran → Compiler → Application Binary + Opt Reports → Performance Tools → Identify hotspots, problems

Application Source + Hotspots → Compiler in advice-mode → Advice messages

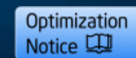
GAP

-guide=n
/Qguide:n

Modified Application Source → Compiler (extra options) → Improved Application Binary

Simplifies programmer's effort in application tuning, and avoids compiler jargon.

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



Vectorization Example

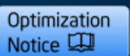
```
void mul(NetEnv* ne, Vector*
    rslt
  Vector* den, Vector* flux1,
  Vector* flux2, Vector* num
{
  float *r, *d, *n, *s1, *s2;
  int i;
  r=rslt->data;
  d=den->data;
  n=num->data;
  s1=flux1->data;
  s2=flux2->data;

  for (i = 0; i < ne->len; ++i)
    r[i] = s1[i]*s2[i] +
    n[i]*d[i];
}
```

GAP Messages (simplified):

1. Use a local variable to store the upper-bound of loop at line 29 (variable: ne->len) if the upper-bound does not change during execution of the loop
2. Use "#pragma ivdep" to help vectorize the loop at line 29, if these arrays in the loop do not have cross-iteration dependencies: r, s1, s2, n, d

-> Upon recompilation, the loop will be vectorized

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



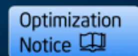
Diagnostic Level of Vectorization Switch
L&M: -vec-report<N> W: /Qvec-report<N>

N	Diagnostic Messages
0	No diagnostic messages; same as not using switch and thus default
1	Report about vectorized loops– default if switch is used but N is missing
2	Report about vectorized loops and non-vectorized loops
3	Same as N=2 but add add information on assumed and proven dependencies
4	Report about non-vectorized loops
5	Same as N=4 but add detail on why vectorization failed

Note:

- In case inter-procedural optimization (-ipo or /Qipo) is activated and compilation and linking are separate compiler invocations, the switch needs to be added to the link step

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



Vectorization Report

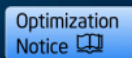
- Provides details on vectorization success & failure
  - L&M: -vec-report<n>, n=0,1,2,3,4,5
  - W: /Qvec-report<n>, n=0,1,2,3,4,5

```
35:     subroutine fd( y )
36:     integer :: i
37:     real, dimension(10), intent(inout) :: y
38:     do i=2,10
39:        y(i) = y(i-1) + 1
40:     end do
41:     end subroutine fd
```

```
novec.f90(38): (col. 3) remark: loop was not vectorized: existence of
vector dependence.
novec.f90(39): (col. 5) remark: vector dependence: proven FLOW
dependence between y line 39, and y line 39.
novec.f90(38:3-38:3):VEC:MAIN_:  loop was not vectorized: existence of
vector dependence
```

Software & Services Group
Developer Products Division

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

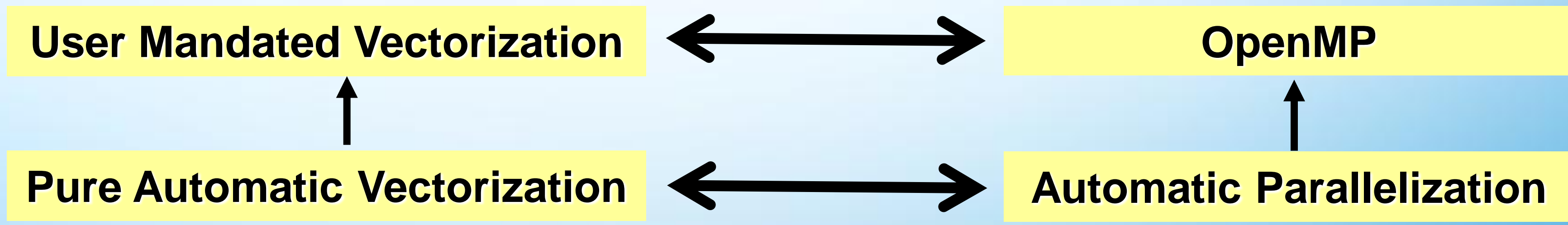


User-Mandated Vectorization

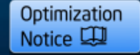
User-mandated vectorization is based on a new **SIMD Directive**

- The SIMD directive provides additional information to compiler to enable vectorization of loops (at this time only inner loop)
- Supplements automatic vectorization but differently to what traditional directives like IVDEP, VECTOR ALWAYS do, the SIMD directive is more a command than a hint or an assertion: The compiler heuristics are completely overwritten as long as a clear logical fault is not being introduced

Relationship similar to OpenMP versus automatic parallelization:



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



SIMD Directive Notation

C/C++: #pragma simd [clause [,clause] …]

Fortran: !DIR$ SIMD [clause [,clause] …]

Without any additional clause, the directive enforces vectorization of the (innermost) loop

Example:

```
void add_fl(float* a, float* b, float* c, float* d, float* e, int n)
{
  #pragma simd
  for (int i=0; i<n; i++)
    a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
}
```

Without the SIMD directive, vectorization will fail (too many pointer references to do a run-time overlap-check).

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

Example*: Matrix-Matrix Multiplication

Row-major matrices (C++) multiplied using a tri-loop implementation.

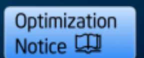
Options W	Options L	Time [s]	Note
/Od	-O0	30.3	
/O1	-O1	9.6	
/O2 /Qvec-	-O2 -no-vec	2.1	
/O3 /Qvec-	-O3 -no-vec	1.6	
/O2	-O2	1.0	
/O3	-O3	0.8	Implies IPO per module
/O3 /Qipo	-O3 -ipo	< 0.8	Inter-module IPO
/fast	-fast	< 0.8	Code specific to compiler host
PGO		0.3	Profile-guided optimization
/fast /Qparallel	-fast -parallel	0.1	[-no-opt-matmul -Qopt-matmul-]!!!

* Remember, this is an example; call an optimized library function to perform standard functionality!
* Evaluated using Intel® Core™ i7, 4-Core 64-bit, 1.6 GHz

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

Vectorization and UDTs

- User-defined types (UDTs)
  - Contribute to expressiveness of algorithms
  - Explicit vectors with hardcoded lengths (e.g. float4) should not be mixed up with primitives for explicit vectorization
- Performance considerations
  - Array of Structure (AoS)
  - Structure of Arrays (SoA)

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


Intel® Composer XE

- Product homepage
  http://software.intel.com/en-us/articles/intel-composer-xe/
- Fortran compiler benchmarks
  http://polyhedron.com/compare0html

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

Intel® Cilk™ Plus

Intel Cilk Plus, est. 2010

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

Cilk Plus Overview

C/C++ Language Extension
Fork-join Task Parallelism

Task Parallelism	Data Parallelism
• Work-stealing • Reducers • Three keywords	• Array sections/slices • Elemental functions • Vectorization

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

History

- MIT Cilk
  - Over 15 years
- Cilk Arts, Inc. (acquired by Intel in 2009)
  - Extended Cilk to C++
- Intel Cilk Plus
  - Simplified build and programming experience
- Open source Cilk
  - Supported by GCC 4.7

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

Getting Started

#include <cilk/cilk.h>

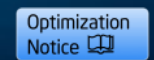
- _Cilk_spawn → cilk_spawn
- _Cilk_sync → cilk_sync
- _Cilk_for → cilk_for

- What else?

  Nothing else. Everything is within the compiler.

#include <cilk/cilk_stub.h>: cilk_spawn/cilk_sync → nothing, cilk_for → for

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



Example: Quicksort Algorithm

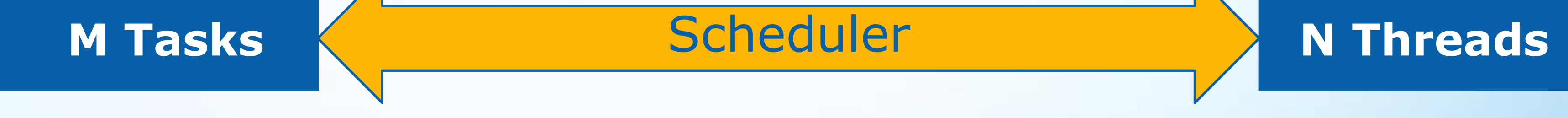
```
void qsort(int* begin, int* end)
{
  if (begin != end) {
    int* pivot = end - 1;
    int* middle = std::partition(begin, pivot,
      std::bind2nd(std::less<int>(), *pivot));
    std::swap(*pivot, *middle);
    cilk_spawn qsort(begin, middle);
    qsort(middle + 1, end);
    cilk_sync;
  }
}
```

(std::bind2nd is deprecated in C++11)

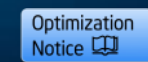
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Cilk Spawn and Sync

Spawning	Stealing
Cheap (3-5x cost of a function call)	Higher cost (locks, barriers etc.)
Often no stealing required	Rare for mostly balanced work
Fork, asynchronous call	Join, sync.



- Forward-scaling (compared to manual threading)
- Simple addition to the code, non-intrusive
  - Easy revertible to serial execution
  - Serial equivalence
- Automatically balanced (scheduler)

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


Cilk For

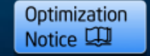
- Features
  - Looks like a normal for loop (type, condition etc.)

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

    Type: incl. iterators over contiguous region
    Condition: for example inequality operation
  - Iterations complete before program continues
  - Iterations may execute in parallel

- Requirements
  - Iterations need to be (order-) independent
  - Only a single induction variable

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



Cilk For vs. Spawn

```
for (int x = 0; x < n; ++x) { cilk_spawn f(x); } // Perf.!
cilk_for (int x = 0; x < n; ++x) { f(x); }        // Perf.!
```

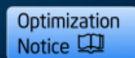
→ Same semantic, but different performance

Implementation

0 - 7
spawn
continuation
0 - 3
4 - 7
spawn
continuation
spawn
continuation
0 - 1
2 - 3
4 - 5
6 - 7
0 1 2 3 4 5 6 7

Work per iteration can be little, and therefore the steal overhead eventually becomes significant.

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



Implicit Syncs

```
void f() {
  cilk_spawn g();
  cilk_for (int x = 0; x < lots; ++x) {
    ...
  }
  try {
    cilk_spawn h();
  }
  catch (...) {
    ...
  }
}
```

- At end of a cilk_for body (does not sync g())
- Before entering a try block containing a sync
- At end of a try block containing a spawn
- At end of a spawning function

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

Reducers

- What is a reducer?

  C++: cilk::reducer_op_add<int> sum(3); // initial view

- When continuation is stolen…
  - An "identity view" of the reducer is used to continue
  - Cilk sync will merge all views according to the reducer

- Hyperobject library

  reducer_list_append, reducer_list_prepend, reducer_max, reducer_max_index, reducer_min, reducer_min_index, reducer_opadd, reducer_ostream, reducer_basic_string, …

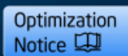
  (Write own reducers by implementing cilk::monoid_base and cilk::reducer)

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

Example: Reduction

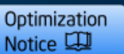
```
int accumulate(const int* array,
               std::size_t size)
{
  reducer_opadd<int> result(0);
  cilk_for (std::size_t i = 0; i < size; ++i) {
    result += array[i];
  }
  return result.get_value();
}
```


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



Example: Data Race

```
int accumulate(const int* array,
               std::size_t size)
{
  int result = 0;
  cilk_for (std::size_t i = 0; i < size; ++i) {
    result += array[i];
  }
  return result;
}
```


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


Adjusting Behavior

- Runtime system API*

```
__cilkrts_set_param("nworkers","5");
int __cilkrts_get_nworkers(void);
int __cilkrts_get_worker_number(void);
int __cilkrts_get_total_workers(void);
```

- Pragmas

```
#pragma cilk grainsize = expression
    cilk_for (…)
```

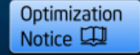
- Serialization (elision)

```
/Qcilk-serialize or -cilk-serialize
```

- Environment variables

```
CILK_NWORKERS
```


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

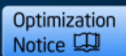


Intel® Cilk™ Plus

Elemental Functions and Array Sections

Intel Cilk Plus, est. 2010

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



Overview

- Elemental functions (“kernels”)
  - Properties apply to guarantee vectorization
  - Control flow is supported (masked exec.)

- Array notation (sections/slices)
  - [start:size], or
  - [start:size:stride]
  - [:] → everything*

For example: y[0:10:10] = sin(x[20:10:2]);
z[0:10][10:10] // 100 elements

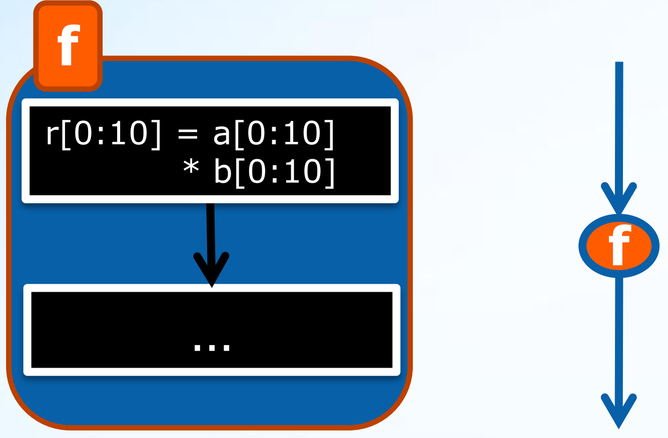
F90-style notation: [begin:end], or [begin:end:stride]
* only works for array shapes known at compile-time

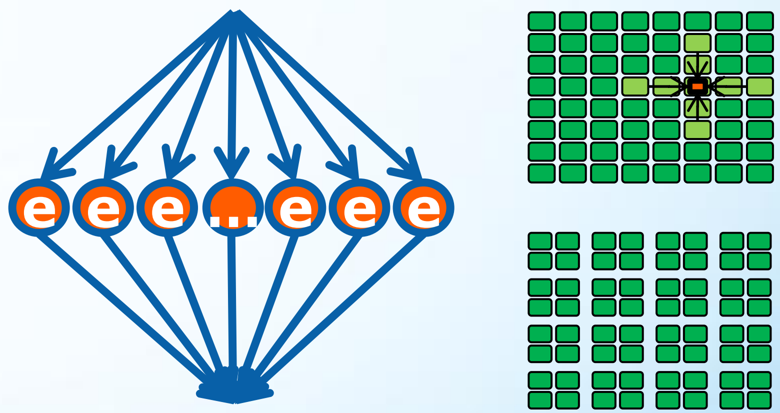
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Vector Processing vs. Elemental Functions

Vector Processing

Elemental Processing



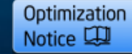


More natural for gather, scatter, synchronization, …

"Kernel"

Elemental processing is naturally embedded into a more general vector processing context allowing e.g. scatter etc.

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

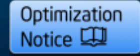


Operators

- Most C/C++ operators work with array sections
  - Element-wise operators (rank and size must match) `a[0:10] * b[4:10]`
  - Scalar expansion `a[10:10] * c`
- Assignment and evaluation
  - Evaluation of RHS before assignment `a[1:8] = a[0:8] + 1`
  - Parallel assignment to LHS `^ temp!`
- Gather and scatter*

```
a[idx[0:1024]] = 0
b[idx[0:1024]] = a[0:1024]
c[0:512] = a[idx[0:512:2]]
```

* Direct instruction support on certain HW targets.

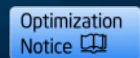
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.


Example: Element-wise Addition (Array Section)

- Explicit construct to *express* vectorization
- Compiler assumes no aliasing of pointers (similar to #pragma simd)

```
void sum(const int* begin, const int* begin2,
         std::size_t size, int* out)
{
  out[0:size] = begin[0:size] + begin2[0:size];
}
```


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



Example: Element-wise Addition (Array Section, Elemental/Kernel Function)

```
__declspec(vector)
void kernel(int& result, int a, int b)
{
  result = a + b;
}


void sum(const int* begin, const int* begin2,
         std::size_t size, int* out)
{
  kernel(out[0:size],
      begin [0:size],
      begin2[0:size]);
}
```


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

Example: Element-wise Addition (Array Section, Kernel Function, Cilk For)

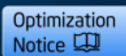
```
__declspec(vector)
void kernel(int& result, int a, int b)
{
  result = a + b;
}


void sum(const int* begin, const int* begin2,
         std::size_t size, int* out)
{
  cilk_for (std::size_t i = 0; i < size; ++i) {
    kernel(out[i], begin[i], begin2[i]);
  }
}
```


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

Example: Element-wise Addition (No Threading, Explicit Vector Length)

```
__declspec(vector)
void kernel(int& result, int a, int b)
{
  result = a + b;
}


void sum(const int* begin, const int* begin2,
         std::size_t size, int* out)
{
  for (std::size_t i = 0; i < size; i += 8) {
    kernel(out[i:8], begin[i:8], begin2[i:8]);
  }
}
```

The amount of data needs to be a multiple of eight. Allows to handle the remainder in a customized way.

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


Reductions

- Generic reductions

```
result __sec_reduce(initial, a[:], function-id)
void __sec_reduce_mutating(reduction, a[:], function-id)
```

- Built-in reductions

```
__sec_reduce_add(a[:])
__sec_reduce_mul(a[:])
__sec_reduce_all_zero(a[:])
__sec_reduce_all_nonzero(a[:])
__sec_reduce_any_nonzero(a[:])
__sec_reduce_min(a[:])
__sec_reduce_max(a[:])
__sec_reduce_min_ind(a[:])
__sec_reduce_max_ind(a[:])
```


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

Other Operators

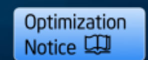
- Shift operators

```
b[:] = __sec_shift(a[:], signed shift_val, fill_val)
b[:] = __sec_rotate(a[:], signed shift_val)
```

→ Stencil pattern!

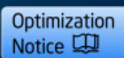
- Cast-operation for array dimensionality, e.g.

```
float[100] → float[10][10]
```


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


Example: Matrix-Vector Multiplication

```
void mxm(double* result,
  const double* matrix, const double* vector,
  std::size_t nrows, std::size_t ncols)
{
  cilk_for (std::size_t i = 0; i < nrows; ++i) {
    const std::size_t start = i * ncols;
    result[i] = __sec_reduce_add(
      matrix[start:ncols] * vector[0:ncols]);
  }
}
```


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


Intel® Cilk Plus

- Language specification and ABI
- Open source announcement

  http://software.intel.com/en-us/articles/intel-cilk-plus-open-source/
  http://www.cilkplus.org/

- GCC status
  - Intel Cilk Plus runtime
  - GCC patch/branch

- Documentation, articles, and user forum

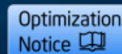
  http://software.intel.com/en-us/articles/intel-c-composer-xe-documentation/
  http://software.intel.com/en-us/articles/intel-cilk-plus-support/
  http://software.intel.com/en-us/forums/intel-cilk-plus/

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

Intel® Threading Building Blocks

Intel TBB, est. 2006

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



TBB Overview

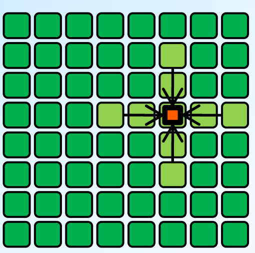
C++ Template Library
Task and Data Parallelism

Tasks and Scheduler	Low-Level Primitives	High-Level Primitives
• Load balancing • Work-stealing	• Synchronization • Par. constructs	• Patterns • Algorithms
• Efficient	• Efficient	• Productive

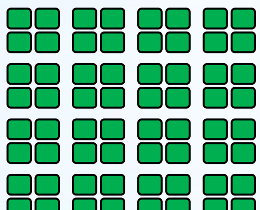
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Parallel Patterns

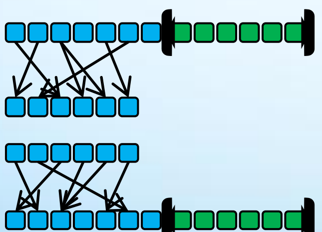
- Stencil



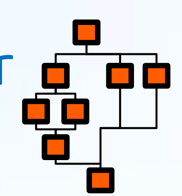
- Partition



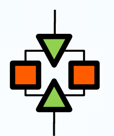
- Gather/scatter



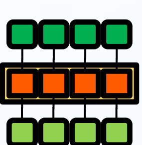
- Superscalar sequence



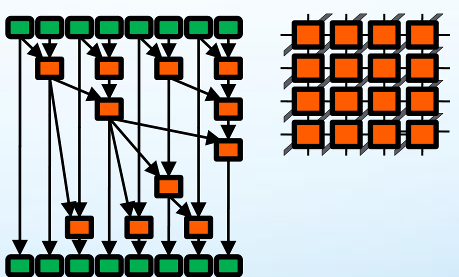
- Speculative selection



- Map



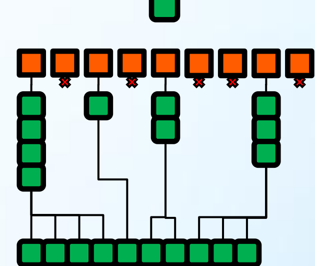
- Scan and Recurrence



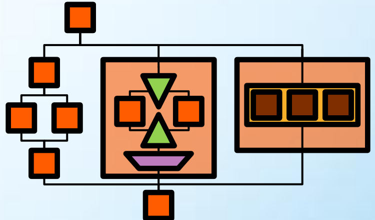
- Pipeline

- Reduce

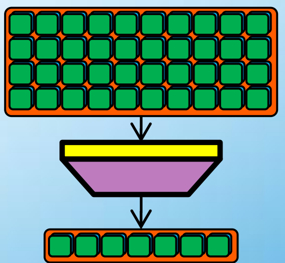
- Pack & Expand



- Nest



- Search & Match



http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/2011Update/tbbxe/Design_Patterns.pdf

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

Example: Getting Started

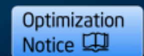
```
int main(int argc, char* argv)
{
  using namespace tbb;
  tbb::task_scheduler_init init;

  tick_count start = tick_count::now();
  // ...
  tick_count end = tick_count::now();
}
```


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

Example: Allocators

```
typedef float T; // SP
typedef tbb::cache_aligned_allocator<T> A;

std::vector<T, A> buffer(1024);
```

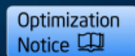

Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.



C++11 (C++0x)

- Parallel programming
  - Core language
    - Lambda expressions, type inference, etc.
    - Range-based “for” loops
    - Memory model, TLS, atomics
  - Standard template library (STL)
    - Threads, synchronization (lock objects)
- Compiler support
  - GNU g++ Compiler V4.5
  - Intel® C++ Compiler V12
  - Microsoft* C++ Compiler V16 (2010)

Criterion for compiler support: λ-expressions

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


Example: Element-wise Addition

Syntactical advantages with C++11 (λ)

```
void sum(const int* begin, const int* begin2,
         std::size_t size, int* out)
{
  tbb::parallel_for(std::size_t(0), size,
    [=](std::size_t i) {
      out[i] = begin[i] + begin2[i];
    }
  );
}
```

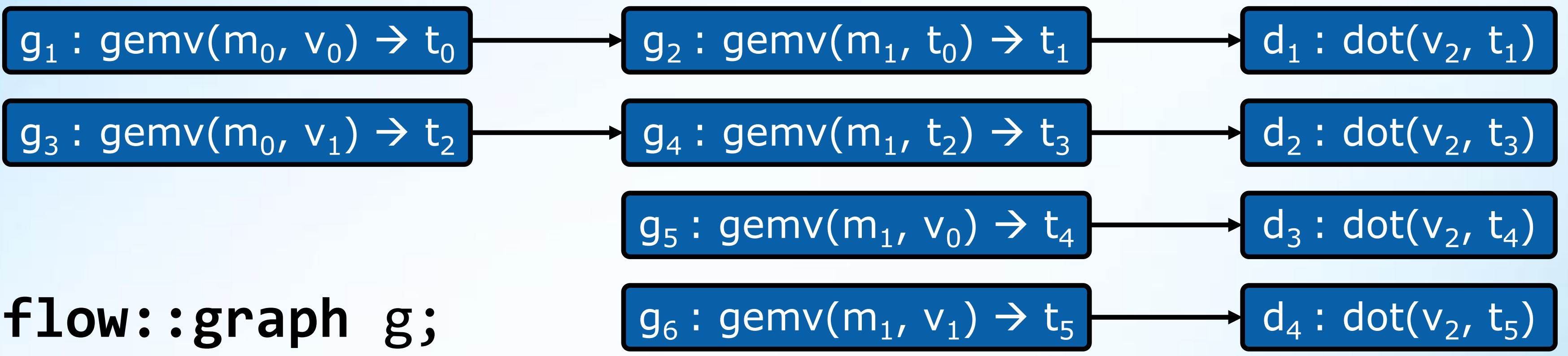

Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Scheduler

- Task-based parallelization
  - Abstraction from manual thread usage (system progr.)
  - Scheduling policy can be separated (balanced work)
  - Grain/block size to fit into CPU cache

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

Example: Expression Evaluation (flow::graph)



- **Idea*** extract parallelism within an expression
  - Use **`flow::function_node`** objects to perform an action
  - Use **`flow::make_edge`** to connect the nodes

* Note, “gemv” and “dot” are not required to be (data–)parallel.

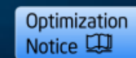
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

Example: Expression Evaluation (flow::graph)

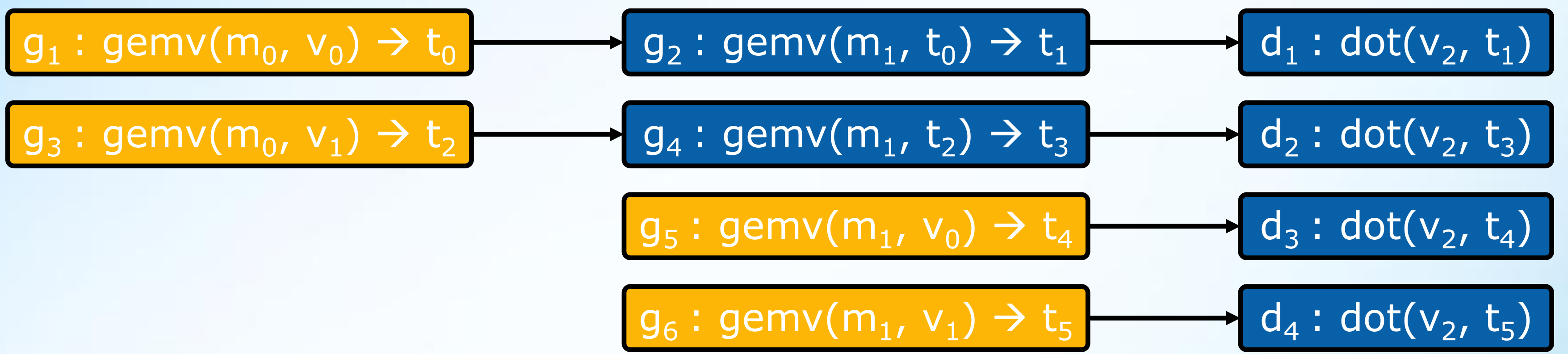
g_1 : gemv(m_0, v_0) → t_0 → g_2 : gemv(m_1, t_0) → t_1 → d_1 : dot(v_2, t_1)

```
flow::graph g;

flow::function_node<in_gt, out_gt>  g1(g, /*g1 args*/);
flow::function_node<in_gt, out_gt>  g2(g, /*g2 args*/);
flow::function_node<in_dt, out_dt>  d1(g, /*d1 args*/);

flow::make_edge(g1, g2);
flow::make_edge(g2, d1);
```

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



Example: Expression Evaluation (flow::graph)

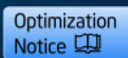



```
g1.try_put(/*g1 input*/);
g3.try_put(/*g3 input*/);
g5.try_put(/*g5 input*/);
g6.try_put(/*g6 input*/);
g.wait_for_all();
```

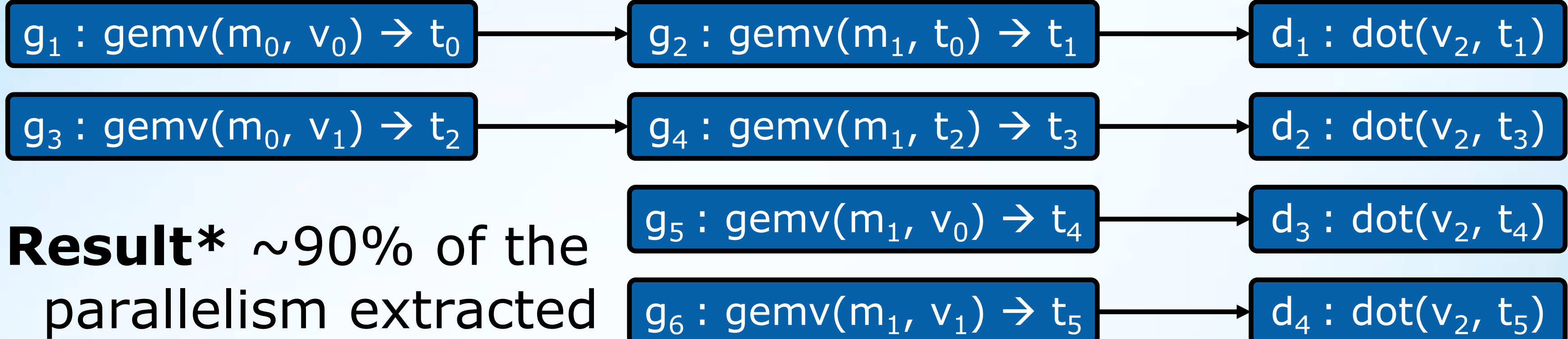
More ideas:

- Use **`flow::broadcast_node`** to express feeding mult. nodes ($m_0 \rightarrow$ g1/g3, $m_1 \rightarrow$ g5/g6)
- Overlap multiple evaluations by delaying **`wait_for_all()`**

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



Example: Expression Evaluation (flow::graph)



Result* ~90% of the parallelism extracted

```
Intel Composer XE 2011  IA-32 Visual Studio 2008
Expression Evaluation (serial)
 Time 1: 1.54824 ms
 Time 2: 1.56808 ms
 Time 3: 1.5474 ms
 Time 4: 1.56472 ms
 Time 5: 1.58763 ms
Minimum: 1.5474 ms
Results: -1531.1 544.084 -12.1451 40.8885

Expression Evaluation (flow::graph)
 Time 1: 0.964368 ms
 Time 2: 0.837537 ms
 Time 3: 0.851505 ms
 Time 4: 0.843683 ms
 Time 5: 0.846197 ms
Minimum: 0.837537 ms
Results: -1531.1 544.084 -12.1451 40.8885

Comparison
Short Path: 0.772724 ms
Graph Time: 0.837537 ms -> 92%
```

* Intel® Core 2 Duo T9600 @ 2.80 GHz

Summary

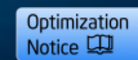
- Static…dynamic dependencies (can be created at runtime)
- Fine…coarse grained tasks (low overhead)
- Easy way to run non-threaded (foreign) functionality but to exploit multicore and many-core

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

Other Parallel Algorithms

- Loop parallelization
  - parallel_for and parallel_reduce
  - parallel_scan
- Parallel Algorithms for Streams
  - parallel_do
  - parallel_for_each
  - pipeline
- Parallel function invocation
  - parallel_invoke
- Parallel Sort
  - parallel_sort

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



Intel® Threading Building Blocks

- Open source portal
  http://threadingbuildingblocks.org/

- Documentation, articles, and user forum
  http://software.intel.com/en-us/articles/intel-c-composer-xe-documentation/
  http://software.intel.com/en-us/articles/intel-threading-building-blocks-kb/all/
  http://software.intel.com/en-us/forums/intel-threading-building-blocks/

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

A Good Parallel Programming Model

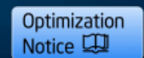
Easy to use

Produces software that is safe

Makes my code go faster

Co-exists with other programs

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



Questions?

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

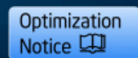
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

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



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 © 2011. Intel Corporation.

http://intel.com/software/products

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