Cilk™ Plus in GCC

GNU Tools Cauldron 2012

Balaji V. Iyer
Robert Geva and Pablo Halpern
Intel Corporation

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

Presentation Outline

- Introduction
- Cilk Plus components
- Implementation
- GCC Project Status
- The Next steps and Community Contribution

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

Why Parallel Programming?

- Most computers today contain multiple cores.
  - More power efficient to use multiple compute elements than a monolithic processor with high clock rates.
- Vector units and SIMD instructions are present in most modern microprocessors
  - AVX (Intel), NEON (ARM), MIPS-3D (MIPS), Altivec (Motorola), Tilera (tilepro), Tensilica (xtensa), …
- Future Trends:
  - Transistor densities continue to increase
  - High throughput while consuming less power
  - Complex handheld devices are emerging ➔ low-energy

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

Parallel Programming

- Parallel programming is necessary to fully utilize today's processors
- The application must keep the processors busy while achieving high-performance and consuming less power.
- Parallel programming can be hard!
  - Combining threads and locks introduces errors & performance issues
  - Programming with threads is tedious and *non-expressive*
  - Programming directly with threads often leads to undesirable non-determinism
- The vectorizer may need help from programmer for C/C++ programs due to pointer aliasing

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

Cilk Plus

The Intel ® Cilk™ Plus specification is a set of C/C++ extensions for programming multicore vector processors that:

- Incorporates vector and task level parallelism
- Describes the semantics of a parallel program so that the compiler can apply thread-level and vector-level parallelism
- Has been applied to GCC

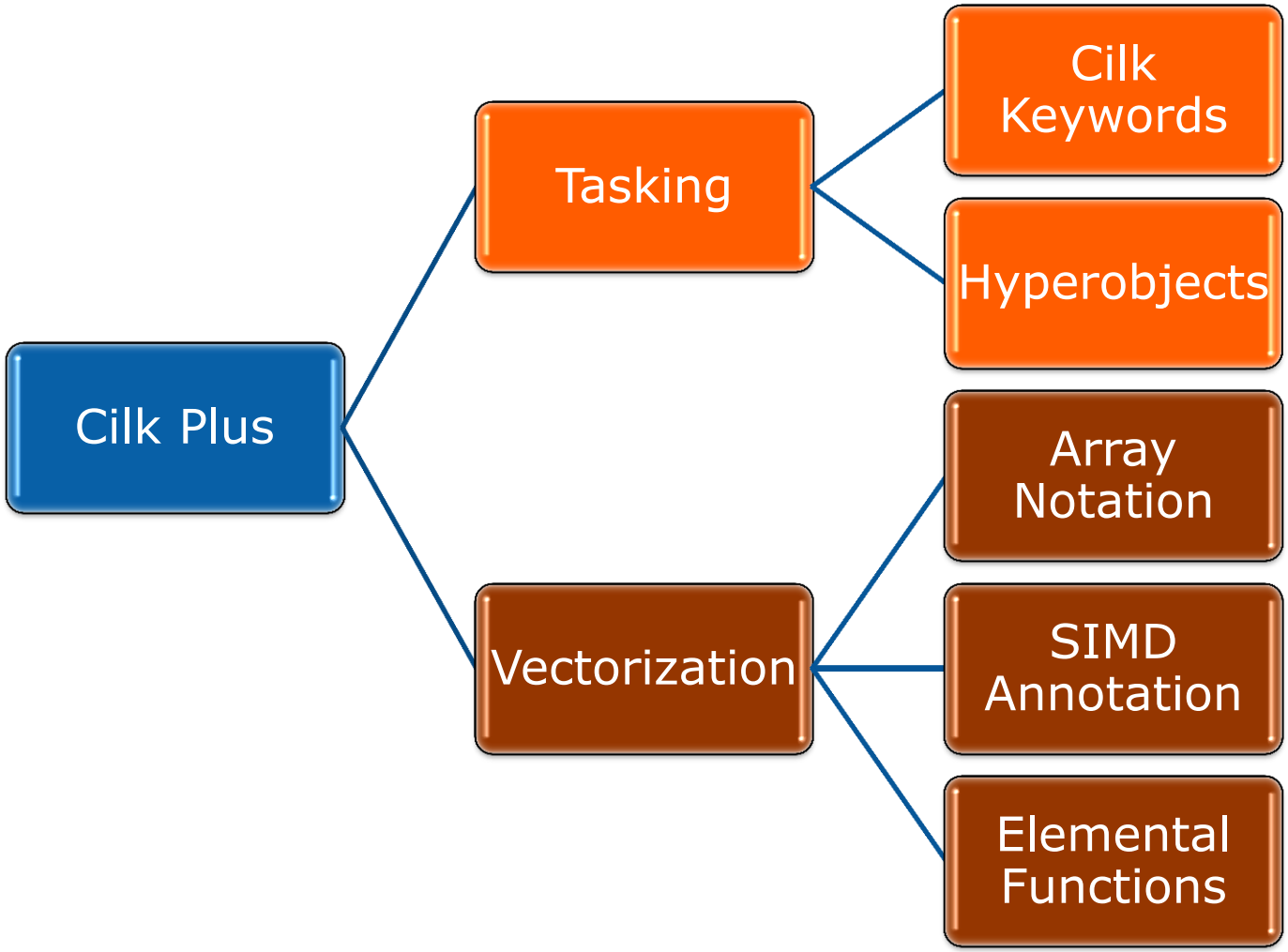
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Cilk Plus Implementation

- Fully implemented in Intel's compiler (ICC)
- A branch in GCC was opened in August 2011
- All features of initial ABI have been implemented in Cilk Plus GCC

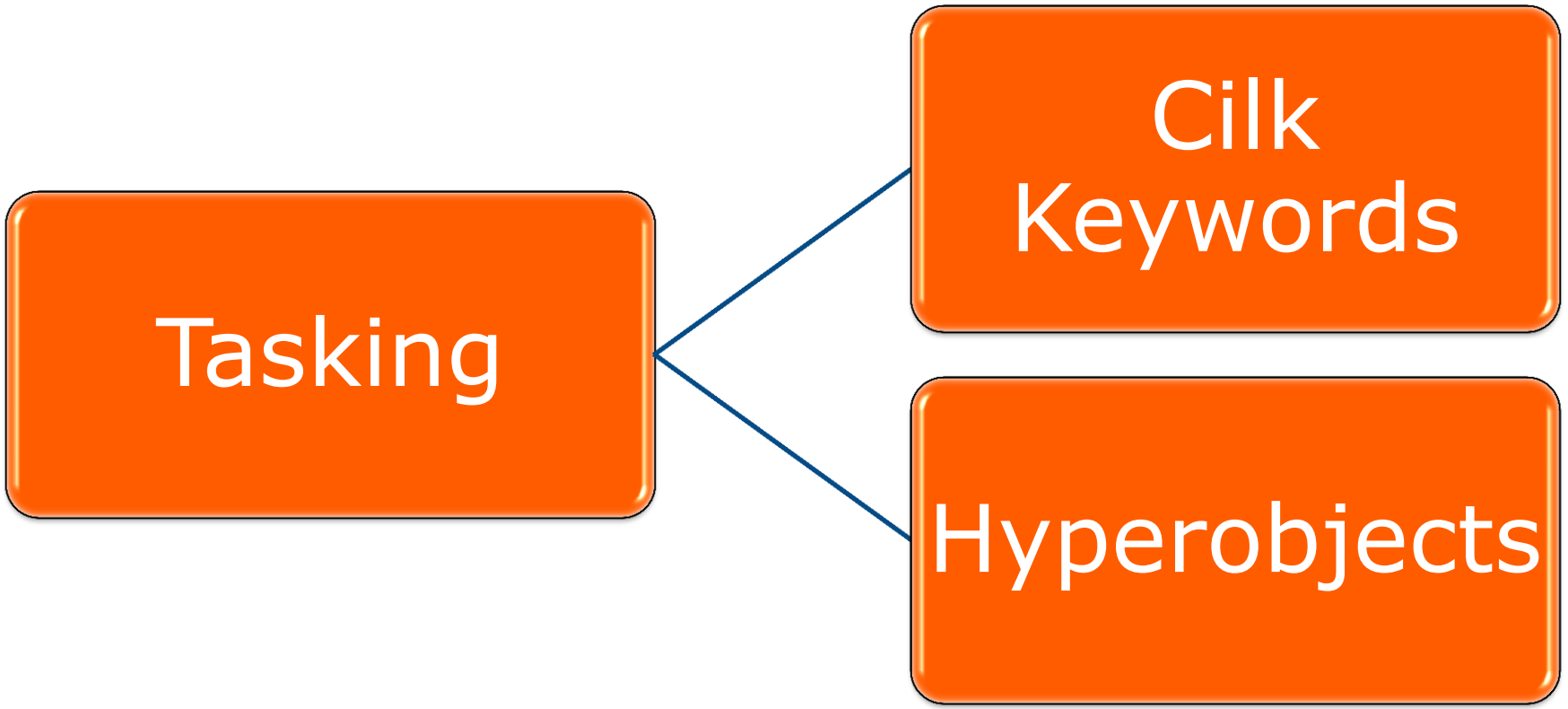
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Cilk Plus Components



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

Cilk Plus Tasking Components

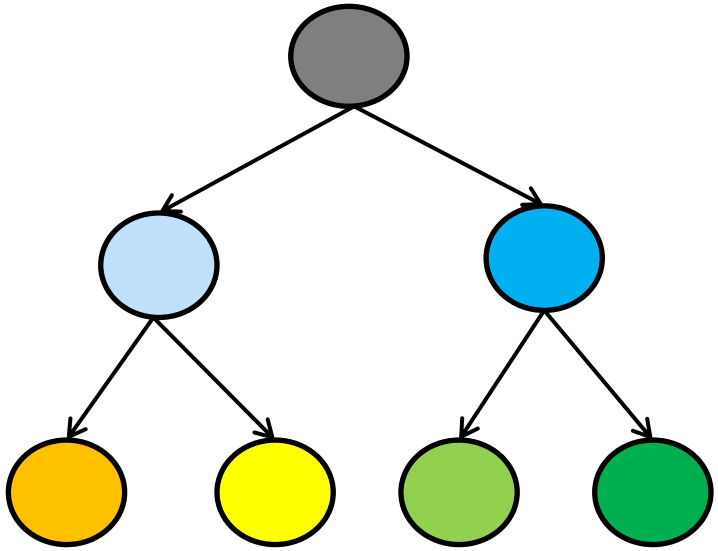


Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Tree Walk Example

```
int tree_walk(node *n)
{
    int a = 0, b = 0;
    if (n->left)
      a = tree_walk(n->left);
    if (n->right)
      b = tree_walk(n->right);
    int c = f(n->value);

    return a + b + c;
}
```



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

Parallel Tree Walk

```
#include <cilk/cilk.h>
int tree_walk(node *n)
{
    int a = 0, b = 0;
    if (n->left)
      a = cilk_spawn tree_walk(n->left);
    if (n->right)
      b = cilk_spawn tree_walk(n->right);
    int c = f(n->value);
    cilk_sync;
    return a + b + c;
}
```


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

Parallel For Loop

```
for (int i = start; i < finish; i += stride)
 { /* Body of loop uses i */ }
f();
```

Iterations can execute in parallel.

All iterations must complete before f() executes

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

cilk_for Loop

```
cilk_for (int i = start; i < finish; i += stride)
    { /* Body of loop uses i */ }
f();
```

- Runtime uses dynamic load-balancing
- Iterations must be independent -- compiler can apply data-parallel optimizations such as vectorization.
- Loop control variable can be any random access iterator.

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

Interaction with Cilk Runtime

Cilk Runtime is responsible for thread creation

- **`cilk_spawn`** gives the runtime *permission* to continue before the called function (child) returns.
  - Low cost (5x to 10x cost of a function call)
  - Code is *processor oblivious*: the number of cores is not specified.
  - If no available resources, then child executes serially.
  - A work-stealing scheduler may *steal* the continuation in the caller and run it asynchronously.
- **`cilk_for`** gives the runtime *permission* to run iterations in parallel
- **`cilk_sync`** does not cause any thread to stall
  - A worker thread just finds other work to steal.
  - No global barrier is implied.

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

Reducer Hyperobjects

- "Traditional" reduction on a parallel for loop:

```
long a[sz];
cilk::reducer_opadd<long> sum(0);
cilk_for (std::size_t i = 0; i < sz; ++i)
    sum += a[i];
```

Warning: reducer_opadd<float> would not be fully deterministic!

Parallel accesses each get their own "view" of sum

- Generalized reduction for *any code* executing in parallel:

```
cilk::reducer_list_append<int> lst;
void tree_walk2(node *n) {
  if (n->left) cilk_spawn tree_walk2(n->left);
  if (n->right) cilk_spawn tree_walk2(n->right);
  lst.push_back(f(n->value));
}
```

Final list has same order as for serial execution!

- You can define your own reducer types.

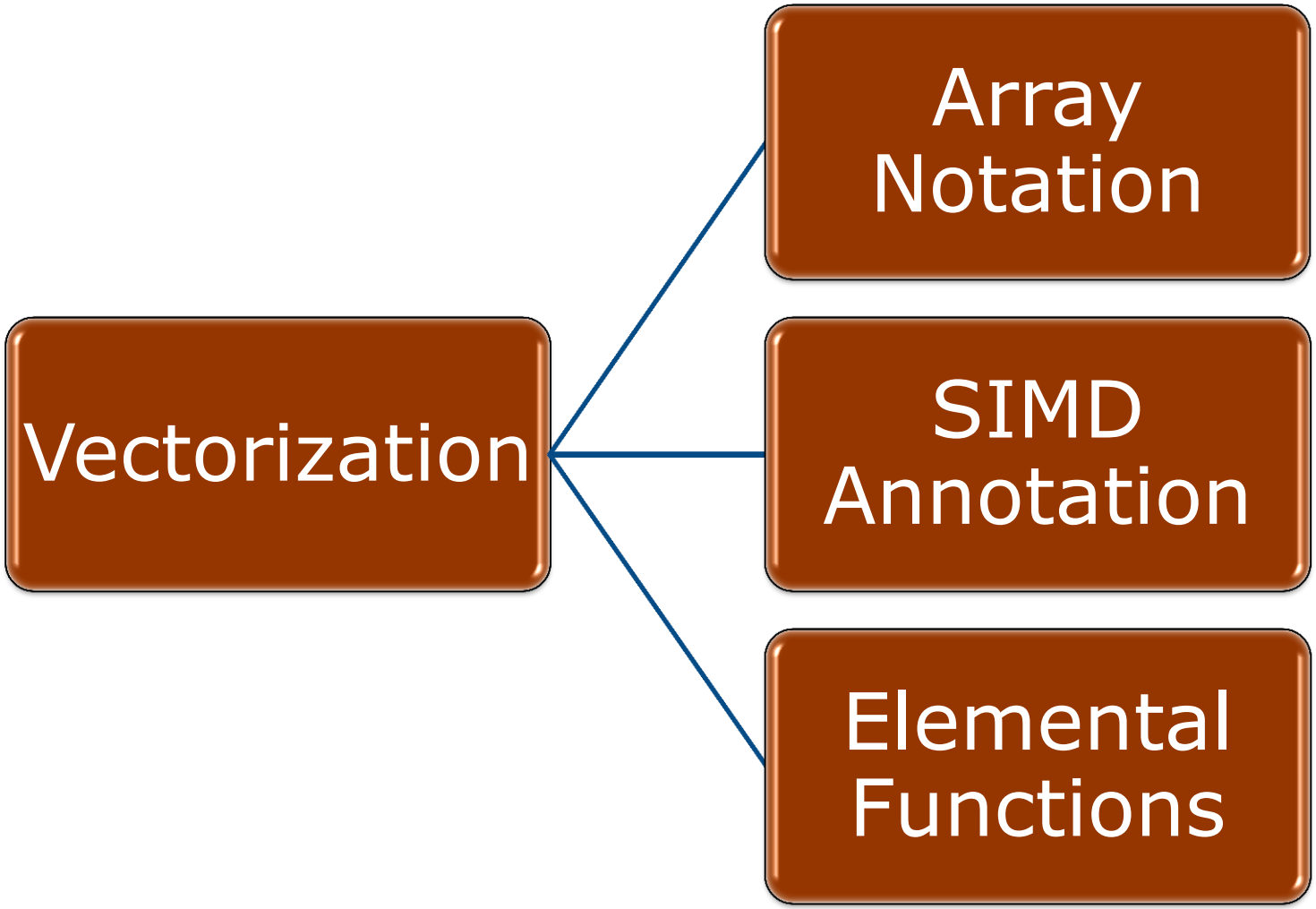
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Tasking Summary

- Using 3 simple keywords a serial program can be converted to a parallel program
- The user need not worry about the processor architecture.
  - It is the runtime's job!
- Hyperobjects help achieve serial semantics in a parallel program

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

Cilk Plus Vectorization Components



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

Array Notation

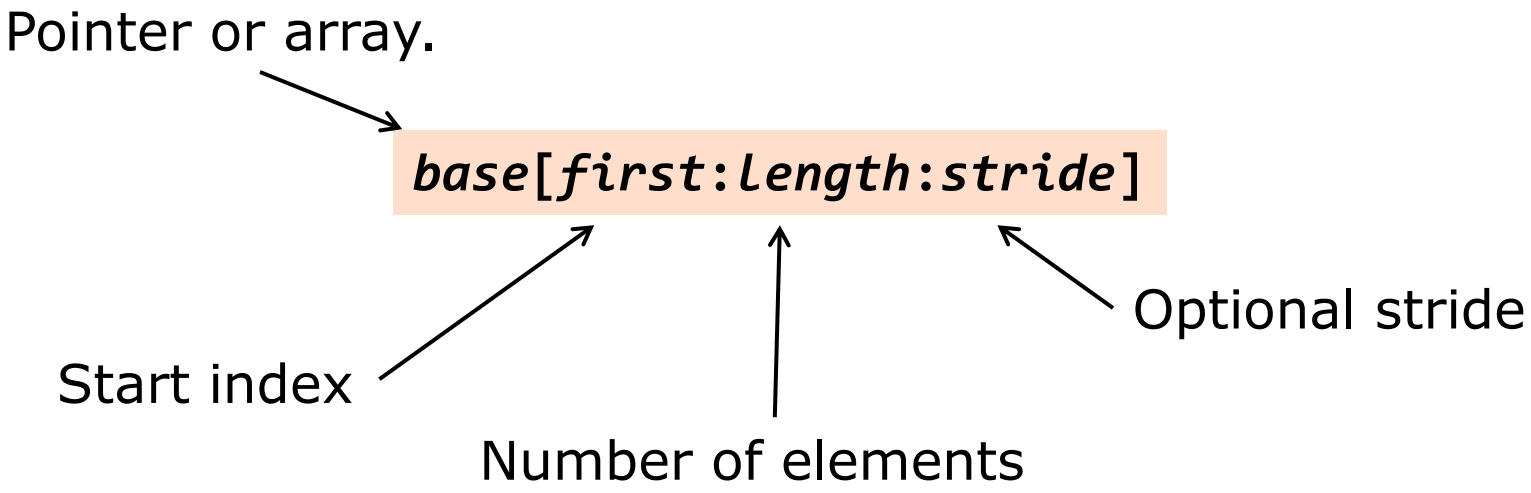
- Let programmer specify parallel intent
  - Give license to the compiler to vectorize

```
// Set y[i] ← y[i] + a * x[i] for i∈[0..n)
void saxpy (float a, float x[], float y[], size_t n)
{
    y[0:n:1] += a * x[0:n:1];
}
```

It also works on pointers

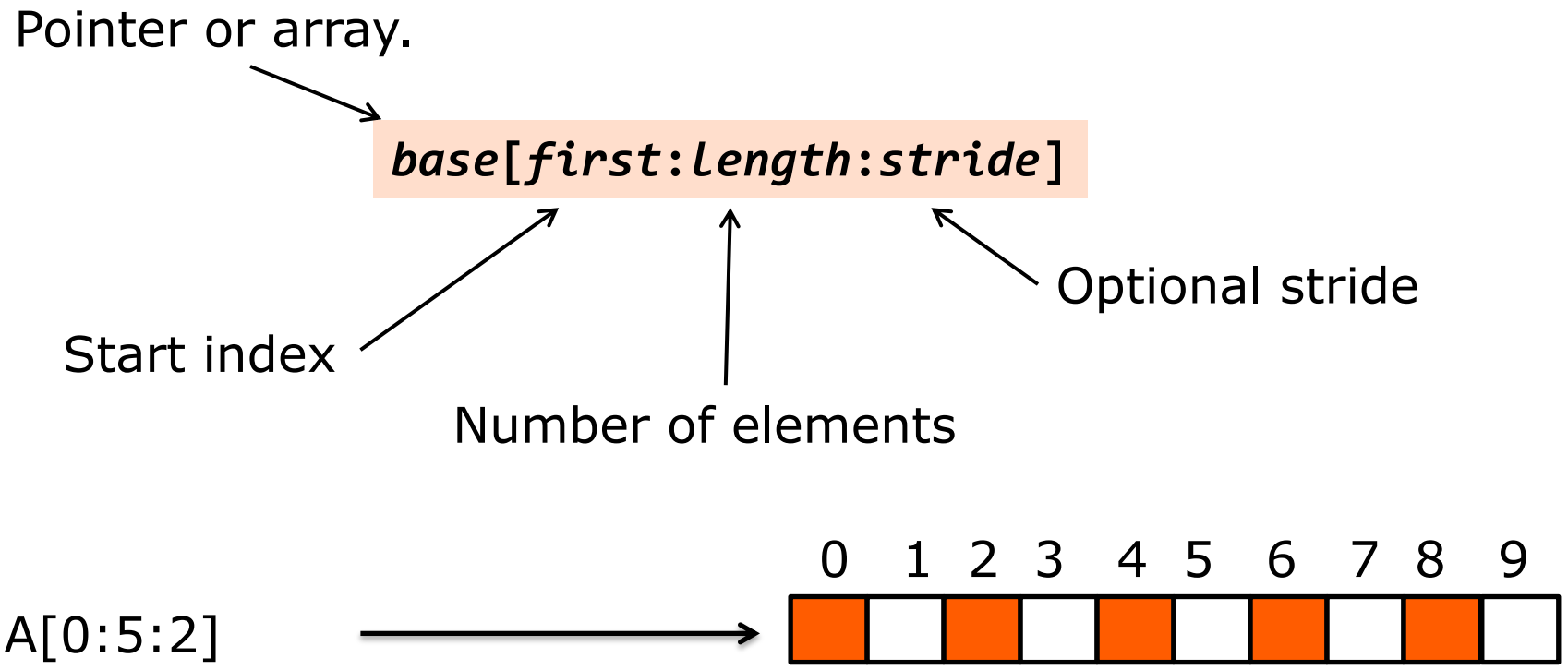
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Array Section Triplet



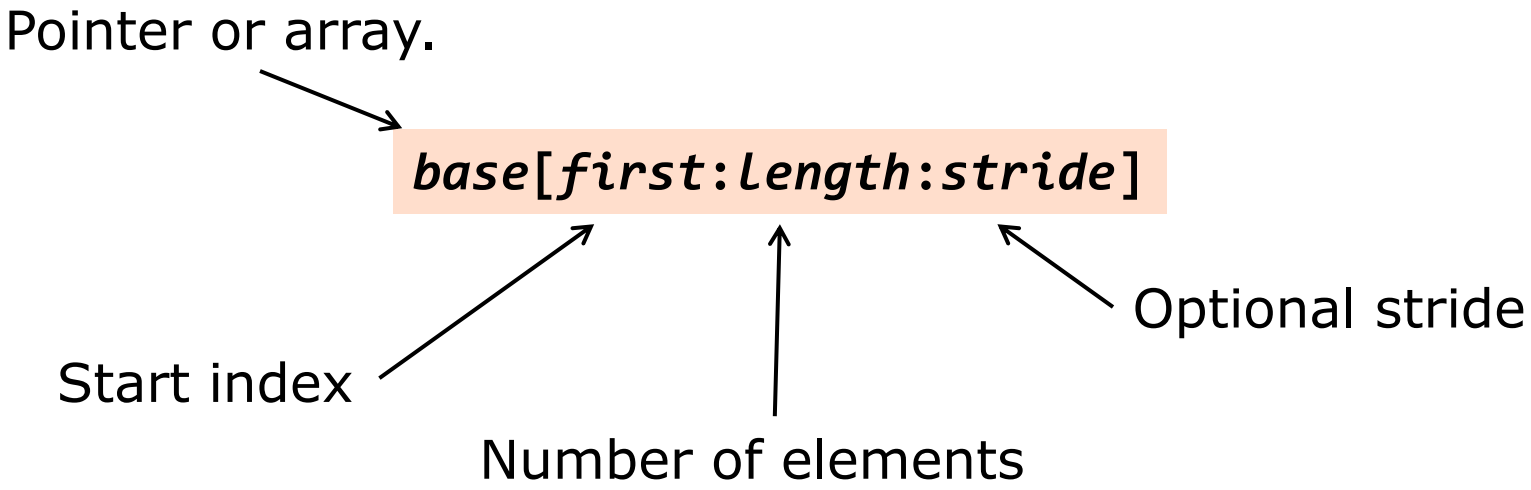
Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

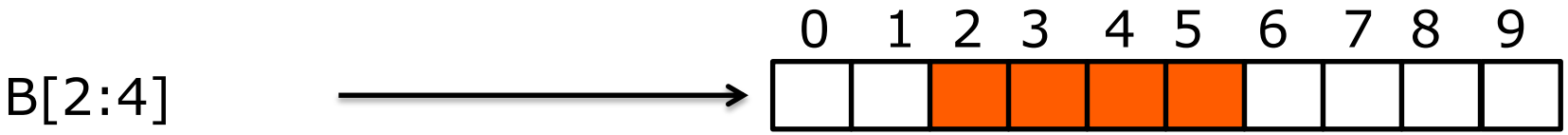
Array Section Triplet



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

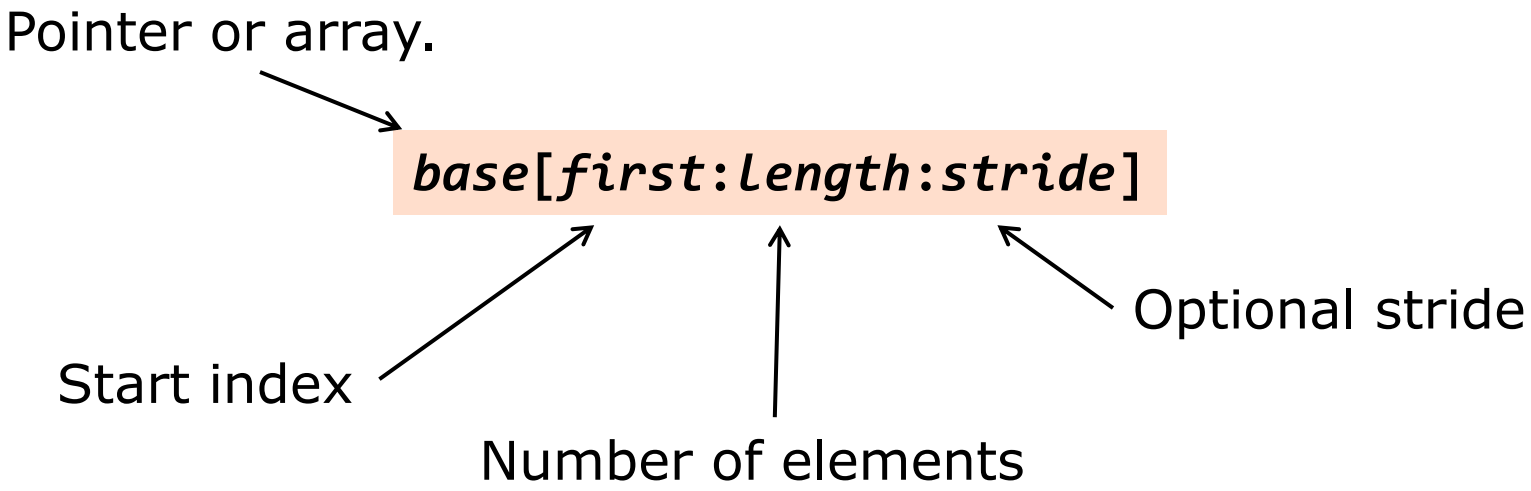
Array Section Triplet

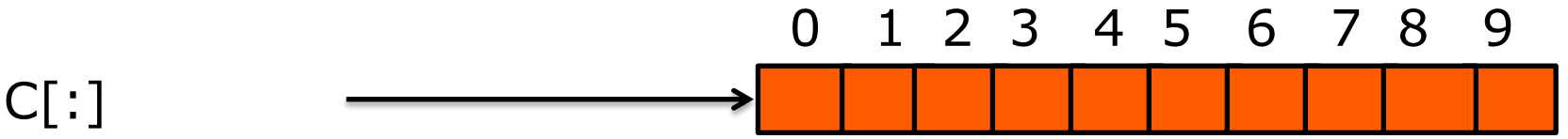




Copyright© 2011, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Optimization Notice

Array Section Triplet





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

Array Notation Decomposition

```
int A[10], *B;
A[1:5:2] = B[1:5];
```

```
for (i = 0; i < 5; i++)
  A[(i*2) + 1] = B[(i*1) + 1];
```

```
int A[10];
A[:] = 5;
```

```
for (i = 0; i < 10; i++)
  A[i] = 5;
```

```
int *B;
B[:] = 5;
```

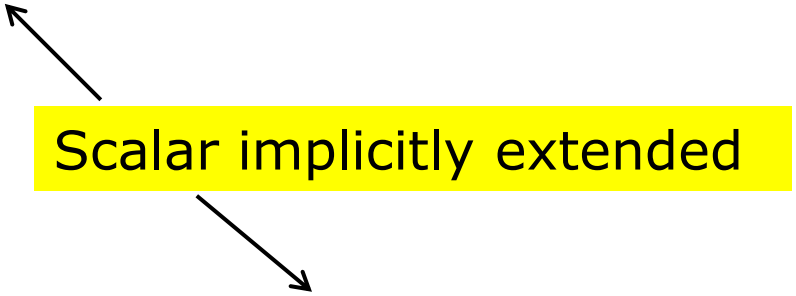
```
error!! one must specify the
start and length of access for
pointers!
```


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

More Examples

- Update m×n tile with corner [i][j].

```
Vx[i:m][j:n] += a*(U[i:m][j+1:n]-U[i:m][j:n]);
```



- Function call

```
theta[0:n] = atan2(y[0:n],1.0);
```

- Gather/scatter

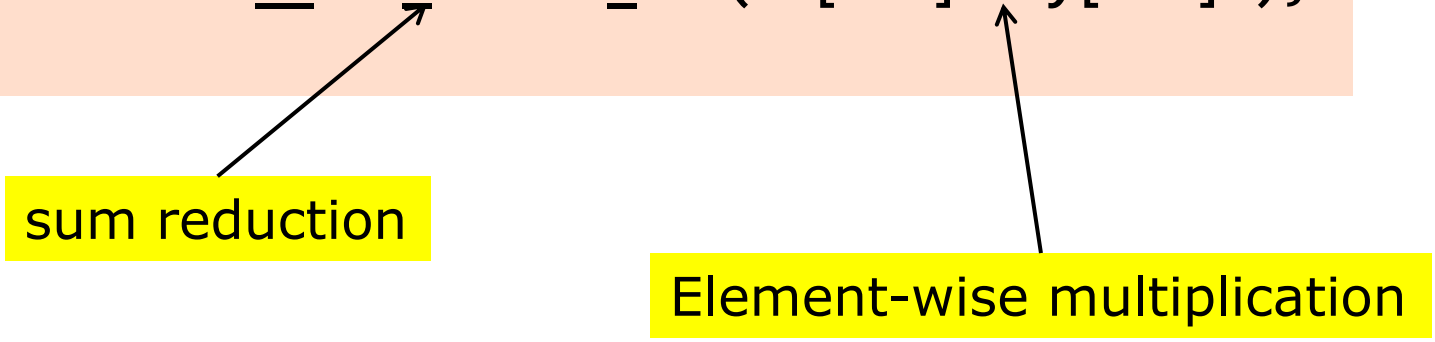
```
w[0:n] = x[i[0:n]];
y[i[0:n]] = z[0:n];
```


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

Built-in Reduction Functions

- Built-in reduction operation for common cases
  +, *, min, index of min, etc.
- User-defined reductions allowed too

```
float dot( float x[], float y[], size_t n )
{
    return __sec_reduce_add( x[0:n] * y[0:n] );
}
```



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

Array Notations with Conditionals

- Array notation can be used within conditionals.
- A vectorizing compiler can generate a mask that allows vector computations based on the condition.

```
int array_abs (int *A, int N)
{

  if (A[0:N] < 0)
    {
      A[0:N] = A[0:N] * -1;
    }
}
```

```
int array_abs (int *A, int N)
{
for (i = 0; i < N; i++)
  if (A[i] < 0)
    {
      A[i] = A[i] * -1;
    }
}
```


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

SIMD Annotation

- Loop annotation informs the compiler that vectorized loop will have same semantics as serial
- Used to break dependencies that user knows are unnecessary.
- Additional clauses for reductions and other vectorization guidance (borrowed from OpenMP*)

```
int func (int *p, int *q) {
#pragma simd
  for (int ii = 0; ii < 10000; ii++)
    {
      *(a+ii) = *(p+ii) + *(q+ii);
    }
}
```


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

How do we vectorize these scenarios?

- File 1 (main.c)

```
extern int my_add (int, int);
for (int i = 0; i < 10000; i++)
  {
     z[i] = my_add (x[i], y[i]);
  }
```

This for-loop cannot be vectorized due to a function call inside it…

- File2 (add.c)

```
int my_add (int x, int y)
{
   return x + y;
}
```

….but this function call can operate on vector registers & return a vector value while achieving correct result

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

Elemental Functions

- A way to allow vectorization across function boundaries
- The user implements a **scalar version** of the function with an appropriate annotation
- The compiler creates a vector version of the function
- Clauses available to describe the target processor and the behavior of the function parameters
- A good application: Writing function libraries

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

Elemental Functions - Example

- Defining an elemental function:

```
                          double option_price_call_black_scholes(
    double S, double K, double r, double sigma, double time)
{
    double time_sqrt = sqrt(time);
    double d1 = (log(S/K)+r*time)/(sigma*time_sqrt) +
        0.5*sigma*time_sqrt;
    double d2 = d1-(sigma*time_sqrt);
    return S*N(d1) - K*exp(-r*time)*N(d2);
}
```

- Invoking the elemental function:

```
for (i = 0; i < N; i++)
  call[i] = option_price_call_black_scholes(S[i],K[i],r,
                                                sigma, time[i]);
```

- Array notations are also allowed

```
call[:] = option_price_call_black_scholes(S[:], K[:], r, sigma, time[:]);
```


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

Vectorization Summary

- Array notation provides a trivial method to write parallel code
- SIMD annotation provides simple pragmas that can be used to vectorize loops by breaking outside dependency
  - Code is portable across compilers
- Elemental function allows vectorization across function boundaries.

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

Presentation Outline

- Introduction
- Cilk Plus Tasking components
- Cilk Plus Vector components
- **Implementation**
- GCC Project Status
- The Next steps and Community Contribution

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

Performance Status

- Overall, Cilk Plus GCC performs as expected on several internal benchmarks
- We noticed one problem: aobench
  - ICC compiled executable was significantly faster than the gcc-compiled one.
  - Upon investigation we found a 2 way nested loop that ICC vectorized and GCC just emitted scalar code.

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

AO bench: Ambient Occlusion Renderer

- Small program for benchmarking real-world floating point performance.
- Case study for combining task and data parallelism in Cilk Plus.
- Parallelized in 1 day using Cilk Plus
- Original Serial code: http://code.google.com/p/aobench/

http://software.intel.com/en-us/articles/data-and-thread-parallelism/

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

AO Bench Kernel

- Aobench has a huge 2-way nested for-loop that was the hottest part of the code.

```
Compute ambient occlusion ()
{
  compute an ortho-basis for intersect pt normal ()
  for number of samples wide
    {
      for number of samples high
        {
          compute random ray using cosines, sines
          and square roots.
          occlusion += intersection of scene objects ()
        }
    }
}
```


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

AO Bench Analysis

- When we inlined this function and some of its callees, several dependencies (due to temporary assignments) in the code disappear and the loops can be vectorized.

- We hand-vectorized the code and we were able to get about half the difference back

- Remaining difference due to vector transcendentals
  - Sine and cosines were used extensively in the for-loop.
  - GCC was not able to insert vector version of these functions

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

Cilk Plus Status Summary

- Cilk Plus is a great set of language extensions to extract both vector-level and task-level parallelism
- Cilk Plus GCC is a branch of GCC live-sources.
- The runtime has been released as free software and is available with Cilk Plus GCC.
- All features of initial Cilk Plus ABI (ABI 0) have been implemented.
  - Update to the most recent ABI (ABI 1) is in progress.
- Proposed for inclusion in the next C++ Standard
- Looking forward to have Cilk Plus adopted into trunk

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

Community Contributions

- Contributions from GCC community are welcome and encouraged.
- We would like adoption and feedback
- Wish list:
  - Better vectorization support
  - Optimize existing procedures.
  - Ports to other architectures

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

Cilk Plus Related Websites

- Cilk Plus Website: www.cilkplus.org
- Language Specification and ABI:
  http://software.intel.com/en-us/articles/intel-cilk-plus-specification/
- Cilk Plus GCC Branch:
  - svn://gcc.gnu.org/svn/gcc/branches/cilkplus

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

Acknowledgements

- Prof. Charles Leiserson
- H. J. Lu for helping us get a Cilkplus branch
- Jeff Law for helping me get GCC write access and valuable comments about the Cilk runtime.
- GCC reviewers who helped us review our patches and help fix problems.
- Many Intel engineers

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

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

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

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 © 2012. 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.