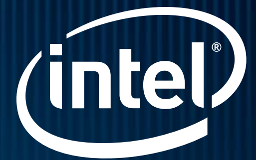
THE PARALLEL UNIVERSE



Issue 6
March 2011

Parallelizing
Intel® Integrated Performance Primitives Functions Using Intel® Cilk™ Plus and Intel® Threading Building Blocks

by Walter Shands

Code Tips
Intel® Array Building Blocks

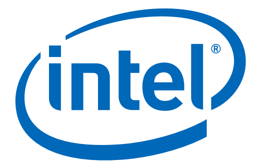
by Zhang Zhang

Letter from the Editor

by James Reinders

Walter Shands
Technical Consulting Engineer

INTEL® PARALLEL STUDIO HAS GONE EXTREME



Achieve unprecedented performance with Intel® Parallel Studio XE

From the makers of Intel® VTune™ Performance Analyzer and Intel® Visual Fortran Compiler comes the ultimate all-in-one performance toolkit.

Victoria Gromova
Technical Consultant Engineer

INTEL® PARALLEL STUDIO XE

Intel® Parallel Studio XE combines Intel's industry-leading C++ and Fortran compilers; libraries; and error-checking, security, and profiling tools into a single tool suite that helps high-performance computing and enterprise developers maximize application performance, security, and reliability.

Intel® C++ Studio XE offers the same benefits for developers who only need the C++ compiler.

Intel Parallel Studio XE includes three next-generation revisions of industry-leading products:

- **Intel® Composer XE**
  Optimizing compilers and high-performance libraries
- **Intel® Inspector XE**
  Powerful thread and memory error checker
- **Intel® VTune™ Amplifier XE**
  Advanced performance profiler

Save with suites

When purchased as part of a suite, all components can be had for a significant savings. For details on buying tools individually, visit http://software.intel.com/en-us/articles/buy-or-renew/.

Already own the software?

If you currently own Intel® software tools, you could be eligible for special upgrade pricing. For details, visit http://software.intel.com/en-us/articles/intel-xe-product-comparison/#upgrade.

Rock your code. **Rock your world.**

Download free 30-day trials of Intel® Software Development Products at **www.intel.com/software/products/eval**.

For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**
© 2011, Intel Corporation. All rights reserved. Intel the Intel logo, and VTune are trademarks of Intel Corporation in the U.S. and other countries.
*Other names and brands may be claimed as the property of others.

CONTENTS

Letter from the Editor

PBB, XE, Cambrian Explosion, and the Art of Computer Programming, BY JAMES REINDERS ... 4

James Reinders, lead evangelist and director of Intel® Software Development Products, discusses the challenge of choice when determining which products and programming models make sense for your programming needs.

Parallelizing Intel® Integrated Performance Primitives Functions Using Intel® Cilk™ Plus and Intel® Threading Building Blocks, BY WALTER SHANDS ... 6

The parallel models in Intel® Parallel Building Blocks easily integrate into existing applications, help preserve investments in existing code, and speed development of parallel applications.

Intel® Array Building Blocks Code Tips, BY ZHANG ZHANG ... 12

Assist the runtime system in generating high-performance code and develop a sound understanding of the Intel® Array Building Blocks API to help avoid errors.

Success Story Roundup ... 20

Envivio, The Creative Assembly, and Altair share how they employed Intel® Software Development tools to achieve increased performance, productivity, and reliability.

Subscribe today: *The Parallel Universe* is a free quarterly magazine. Click the link below to sign up for future issue alerts and to share the magazine with friends.

© 2011, Intel Corporation. All rights reserved. Intel, the Intel logo, Cilk Plus, Intel Core, and Intel VTune are trademarks of Intel Corporation in the U.S. and other countries. *Other names and brands may be claimed as the property of others.

LETTER FROM THE EDITOR

PBB, XE, CAMBRIAN EXPLOSION, AND THE ART OF COMPUTER PROGRAMMING

I like choices because they provide options. For computer programming, like a mechanic, we look for the right tools for the job. This issue of The Parallel Universe magazine helps educate us to select those tools for the job by educating us on programming methods and tools. Articles in this issue highlight parallel programming using the latest Intel® XE tools by showing results and tips using programming models and tools.

As tasty as a Smörgåsbord may be, some people find too many choices distracting. I have noticed that when I am with a group at a Chinese restaurant, with 185 numbered entrees on the menu, many people are slow to choose. I have also noticed that decisions are faster if someone makes a recommendation or if there is a "special of the day" highlighted.

If you are hungry for multicore programming, let me recommend the Intel® Parallel Studio XE product and the Intel® Parallel Building Blocks.

If you have a bigger appetite that includes clusters of processors, let me recommend the Intel® Cluster Studio product and the Intel® Message Passing Interface (MPI) Library.

I received some very nice feedback on the articles in the last issue that helped detail these recommendations. If you missed it, I'd encourage you to go back and read Issue 5 as well at http://i.cmpnet.com/ddj/go-parallel/assets/Parallel_Mag_Issue5.pdf.

In this issue, we'll further explore my recommendations by showing more of what these products and programming models are capable of in practice. The aim is to help you determine when these products and programming models make sense for your programming needs. We can't cover everything, but hopefully you'll find something in these pages that helps you.

We have solutions that extend the most used programming languages and programming environments.

As I've mentioned before, computer hardware has never been more complex. The challenge with programming models and tools is to make this complexity manageable by being high level enough to be usable, while still offering control that is sufficiently low level to make the results useful.

Despite our best efforts, the number of solutions can look overwhelming. A colleague of mine called this the "Cambrian Explosion for Parallel Programming." Can we pick the winners?

I think so, because I think we are actually pretty far into the "Cambrian Explosion" (it started with programming languages in general in the 1950s). The winners of this latest era need to extend the winners of the last explosion. We are not starting over. Should we? I don't think so, and I'll leave elaborating on that for another day.

The most popular solutions, a.k.a. "the winners," will be those that extend the most used programming languages.

That explains why Intel® Parallel Building Blocks and the Intel® Parallel Studio XE tools focus on C, C++, and Fortran developers, and enjoy a high degree of popularity as a result. We have solutions that extend the most used programming languages and programming environments.

That said, we still have a lot to learn together.

The articles in this issue will help by exploring programming models and tools to share success and offer tips.

I'm taking feedback on what additional articles, and perhaps a book, would be useful to you. Feel free to drop me a note at james.r.reinders@intel.com if you have some suggestions. I have enjoyed hearing from many of you in the past year. Keep the comments coming!

Enjoy!

JAMES REINDERS
Portland, Oregon
March 2011

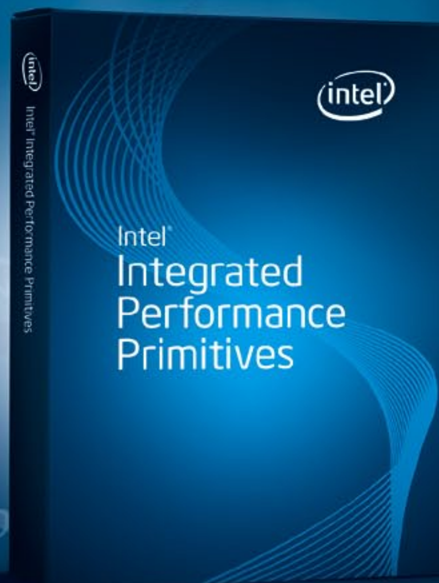
For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

James Reinders is Chief Software Evangelist at Intel Corporation. His articles and books on parallelism include Intel Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism, which has been translated into Japanese, Chinese, and Korean. Reinders is also widely interviewed on the subject of parallelism.

Parallelizing Intel® Integrated Performance Primitives Functions

Using Intel® Cilk™ Plus and Intel® Threading Building Blocks

by Walter Shands



Intel® Integrated Performance Primitives (Intel® IPP) is an extensive library of multicore-ready, highly optimized software functions for multimedia, data processing, and communications applications. Intel® IPP is included in Intel® Composer XE, which is a component of Intel® Parallel Studio XE; and Intel® C++ Composer XE, which is a component of Intel® C++ Studio XE.

Walter Shands
Technical Consulting Engineer

Intel IPP is available in single-threaded and multithreaded versions that can be linked either statically or dynamically. Software developers sometimes prefer to use the single-threaded versions of the library. There are various reasons for this, one of which is to control the number of threads in an application that uses Intel IPP. For instance, if the threaded Intel IPP library is used and an Intel IPP threaded function is called by the application, the Intel IPP threading subsystem will create a number of software threads equal to the number of hardware threads available on the system. If the application code is threaded at a higher level, the total number of threads running in the application when an Intel IPP function is called could exceed the available hardware threads. This can cause increased thread synchronization overhead.

Even so, it is sometimes important to thread individual Intel IPP functions when using the single-threaded versions of the Intel IPP library. This can be the case when an application that has been threaded and tuned for a certain number of hardware threads is run on a system with many more cores. An Intel IPP function call that was not a hotspot on a small number of cores can emerge as a hotspot when the original CPU-intensive portion of the application is spread across many more cores.

It turns out that Intel Composer XE and Intel C++ Composer XE also have tools to add threading to single-threaded Intel IPP functions; these tools are called Intel® Parallel Building Blocks (Intel® PBB). Intel PBB is not the name of a product but a term for a collection of models (or tools) that assist developers with implementing parallelism. Since they share the same foundation, you can mix and match the models that suit your unique parallel implementation needs. These models easily integrate into existing applications, help preserve investments in existing code, and speed development of parallel applications. They consist of Intel® Cilk™ Plus, Intel® Threading Building Blocks (Intel® TBB), and Intel® Array Building Blocks (Intel® ArBB).

Two of these building blocks, Intel TBB and Intel Cilk Plus, provide parallel algorithms and keywords, respectively, to thread existing code. In the following paragraphs, I'll describe how to use these to write wrappers to provide threading to Intel IPP functions.

Intel TBB is a C++ template library solution that can be used to enable general parallelism. It is for C++ developers who write general-purpose loop and task parallelism applications. It includes scalable memory allocation, load-balancing, work-stealing task scheduling, a thread-safe pipeline and concurrent containers, high-level parallel algorithms, and numerous synchronization primitives.

Intel TBB and lambda functions can be used to thread Intel IPP functions on an as-needed basis. The basic idea is to wrap the Intel IPP function call with an Intel TBB parallel algorithm such as parallel_for and use a lambda function to simplify the code. The Intel TBB template function parallel_for parallelizes a for loop by breaking the iteration space into chunks, and running each chunk on a separate thread.

Here is an example of an Intel IPP function that has been threaded using Intel TBB and a lambda function. **Figure 1.**

The Intel IPP function ippiSet_8u_C1R sets the elements of an array of eight bit unsigned integers, which typically represents a piece of an image, to a specific value. The piece of the image to work on is called the region of interest (ROI). The Intel IPP functions with ROI support are distinguished by the presence of an R descriptor in their names.

parallel_for works with a template class provided by Intel TBB called blocked_range<T>, which describes a one-dimensional iteration space over type T. This is essentially the range of data that will be broken up and run on individual threads.

ippiSet_8u_C1R takes a value, the pointer to the start of the ROI, the distance in bytes between lines in the ROI, and a structure that describes the size of the ROI as parameters. By passing the ROI height in lines to the blocked_range template, we have instructed the parallel_for statement to break up the iteration space based on ROI lines.

Since the blocked_range breaks up the iteration space into subspaces, we have to adjust the ippiSet_8u_C1R input ROI size to

```
inline
void inl_ippiSet_8u_C1R(const Ipp8u value, Ipp8u* pDstROI, int dstStep, IppiSize _roisize)
{
  parallel_for( blocked_range<int> (0,_roisize.height), [value, pDstROI, dstStep, _roisize ] (const blocked_
range<int> &r) {
        IppiSize roisize = {_roisize.width, r.end() - r.begin()};
        ippiSet_8u_C1R( value, &pDstROI[ dstStep * r.begin()], dstStep, roisize);
  }, auto_partitioner());
}
```

Figure 1

For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

the size of the ROI subspace that is being worked on. We must also move the input pointer to the ROI array to point to the subspace ROI by providing the appropriate index into the ROI array.

By using a lambda function as the second parameter to parallel_for we can avoid writing an STL-style function object outside the parallel_for statement and put all of the code in the parallel_for. In addition, the wrapper function inl_ippiSet_8u_C1R is made inline to attempt to avoid the cost of an extra function call.

The Intel TBB wrapper provides a reliable, portable, and scalable parallel version of the operation. The Intel IPP function provides low-level optimizations based on the processor's available features such as Streaming SIMD Extensions (SSE) and other optimized instruction sets.

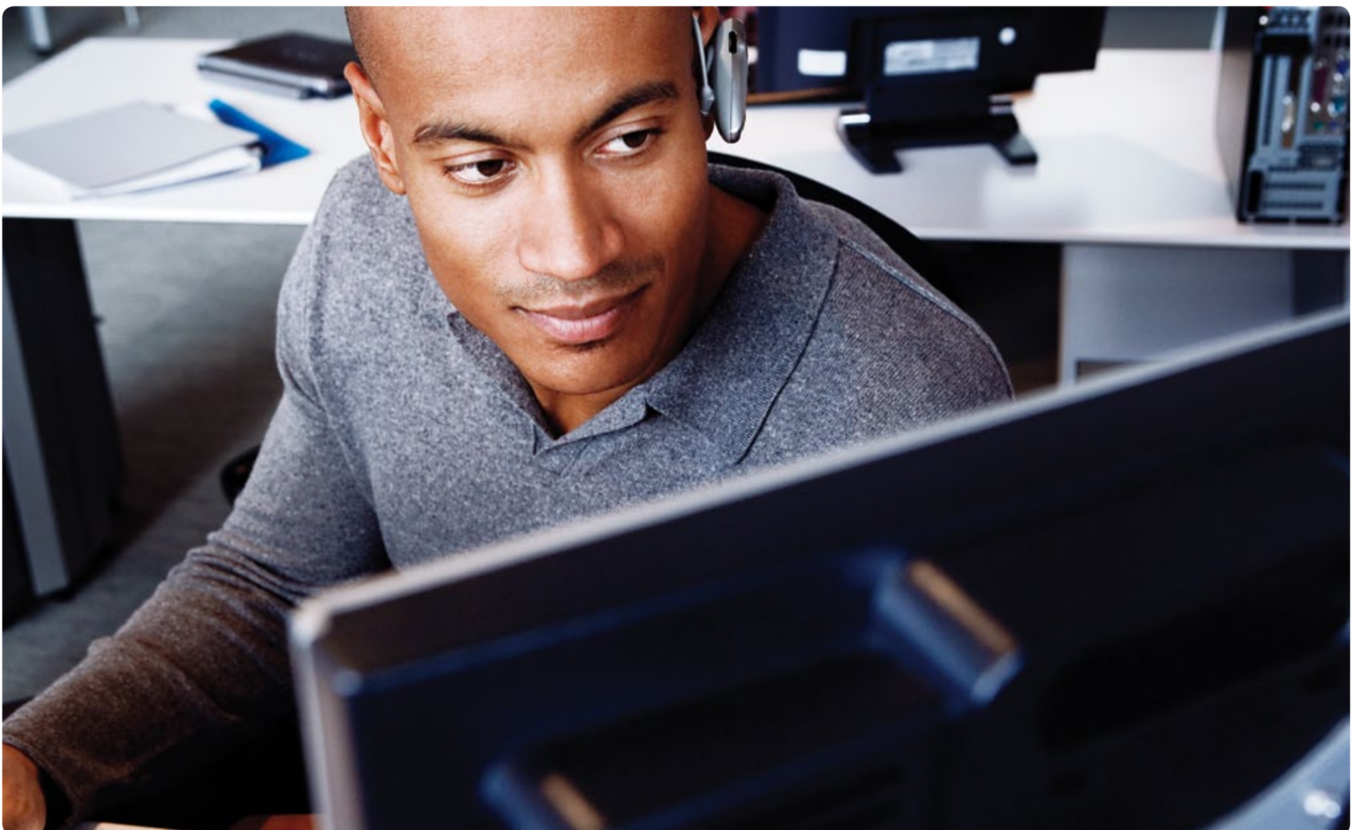
Intel Cilk Plus is an extension to C and C++ implemented by the Intel® C++ Compiler, offering a quick, easy, and reliable way to add threading to applications. Intel Cilk Plus is for C++ software developers who write simple loop and task parallel applications. It offers superior functionality by combining vectorization features with high-level loop-type data parallelism and tasking.

A function wrapper constructed using Intel Cilk Plus can be used instead of an Intel TBB wrapper to thread the Intel IPP call.

Here is an example of the same Intel IPP function threaded using Intel Cilk Plus. **Figure 2.**

In this example, the keyword cilk_for is used to parallelize the original for loop so it scales across multiple cores. A cilk_for loop is a replacement for the normal C++ for loop that permits loop iterations to run in parallel.

In the cilk_for loop the data from the ROI is processed one line at a time so the ROI height is set to one. However, Intel Cilk Plus will divide up the work to be done in the loop so that several lines can be processed by the same thread. Like Intel TBB, Intel Cilk Plus uses a work-stealing task scheduler that ensures load balancing and prevents oversubscription of available hardware threads.



BLOG highlights

Using the Intel® Threading Building Blocks Graph Community Preview Feature: An Implementation of Dining Philosophers

BY MICHAEL VOSS

Intel® Threading Building Blocks (Intel® TBB) Version 3 Update 5 introduced the class graph as a Community Preview (CP) feature. There is an introductory post that provides an overview of the class and the nodes that can be used with it. You can download the open-source version of this release at www.threadingbuildingblocks.org and are encouraged to provide feedback about the graph via the forum. In a previous post, I provided an example that created a simple message graph. In this post, I describe a more complicated example that highlights some interesting features of the API.

This example will demonstrate:

- How to use the graph's run function
- How to mix explicit puts with explicit edges
- The non-greedy nature of the join_node

In this post, I'll provide an implementation for the Dining Philosophers' problem shown below. In this problem, several philosophers are sitting together at a table. Each philosopher needs to both think and eat, but can only do one of these at a time.

The method of using Intel TBB or Intel Cilk Plus wrappers to externally thread Intel IPP functions can be used for many other Intel IPP functions with appropriate adjustments to parameters passed to the function. Intel IPP functions are thread safe, so as long as input data is protected there is no danger of deadlocks or data races when the same Intel IPP function is called from different threads.

This is because Intel IPP functions do not use internal global variables; state changes cannot occur when the same Intel IPP function called on one thread is then called from a different thread with different parameters.

In addition, threading wrappers can be used with the Intel IPP threaded library if parallelization inside the library is disabled beforehand by using the Intel IPP ippSetNumThreads API call with a parameter of 1.

Furthermore, Intel TBB and Intel Cilk Plus constructs can be nested. For instance, you can use an Intel Cilk Plus cilk_for call inside a call to Intel TBB parallel_for and vice versa.

Finally, Intel Composer XE and Intel C++ Composer XE are available on both Linux* and Microsoft Windows* platforms. If you are on a Microsoft Windows platform and using Microsoft Visual Studio*, there is an easy way to add Intel TBB and Intel IPP support to your application. In the case of Microsoft Visual Studio* 2010, edit the project properties, select the **Intel Performance Libraries** item under **Configuration Properties**, and then select Yes in both the Intel Integrated Performance Primitives and Intel Threading Building Blocks drop-down list boxes. Then include the Intel IPP and Intel TBB headers in your application code.

Intel Composer XE and Intel C++ Composer XE combine optimized compilers with high-performance libraries, advanced vectorization, and Intel Parallel Building Blocks, speeding and simplifying threading and performance on Linux and Microsoft Windows with the same code base. Because Intel PBB technologies in Intel Composer XE and Intel C++ Composer XE are compatible with the Intel IPP library, developers can use them to create tailored software solutions that scale across multiple cores. **FIgure 3.** ☐

For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

```
inline
void inlc_ippiSet_8u_C1R(const Ipp8u value, Ipp8u* pDstROI, int dstStep, IppiSize _roisize)
{
        cilk_for( int i=0; i < _roisize.height; i++) {
                IppiSize roisize = {_roisize.width, 1};
                ippiSet_8u_C1R( value, &pDstROI[ dstStep * i], dstStep, roisize);
        };
}
```

Figure 2

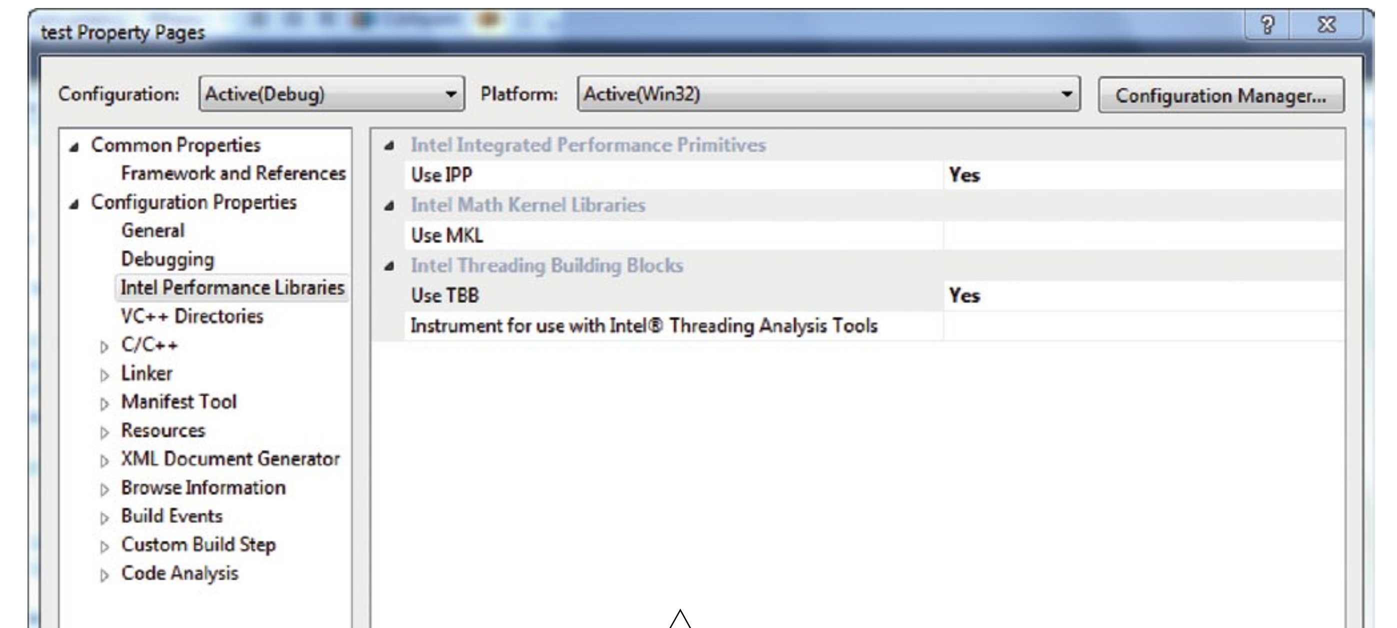


Figure 3

For more information, see the links below:

Intel® Parallel Studio XE—http://software.intel.com/en-us/articles/intel-parallel-studio-xe

Intel® Composer XE and Intel C++ Composer XE—http://software.intel.com/en-us/articles/intel-composer-xe

Intel® Parallel Building Blocks—http://software.intel.com/en-us/articles/intel-parallel-building-blocks

Intel® Cilk™ Plus—http://software.intel.com/en-us/articles/intel-cilk-plus

Intel® Integrated Performance Primitives—www.intel.com/software/products/ipp

Intel® Threading Building Blocks—www.intel.com/software/products/tbb

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

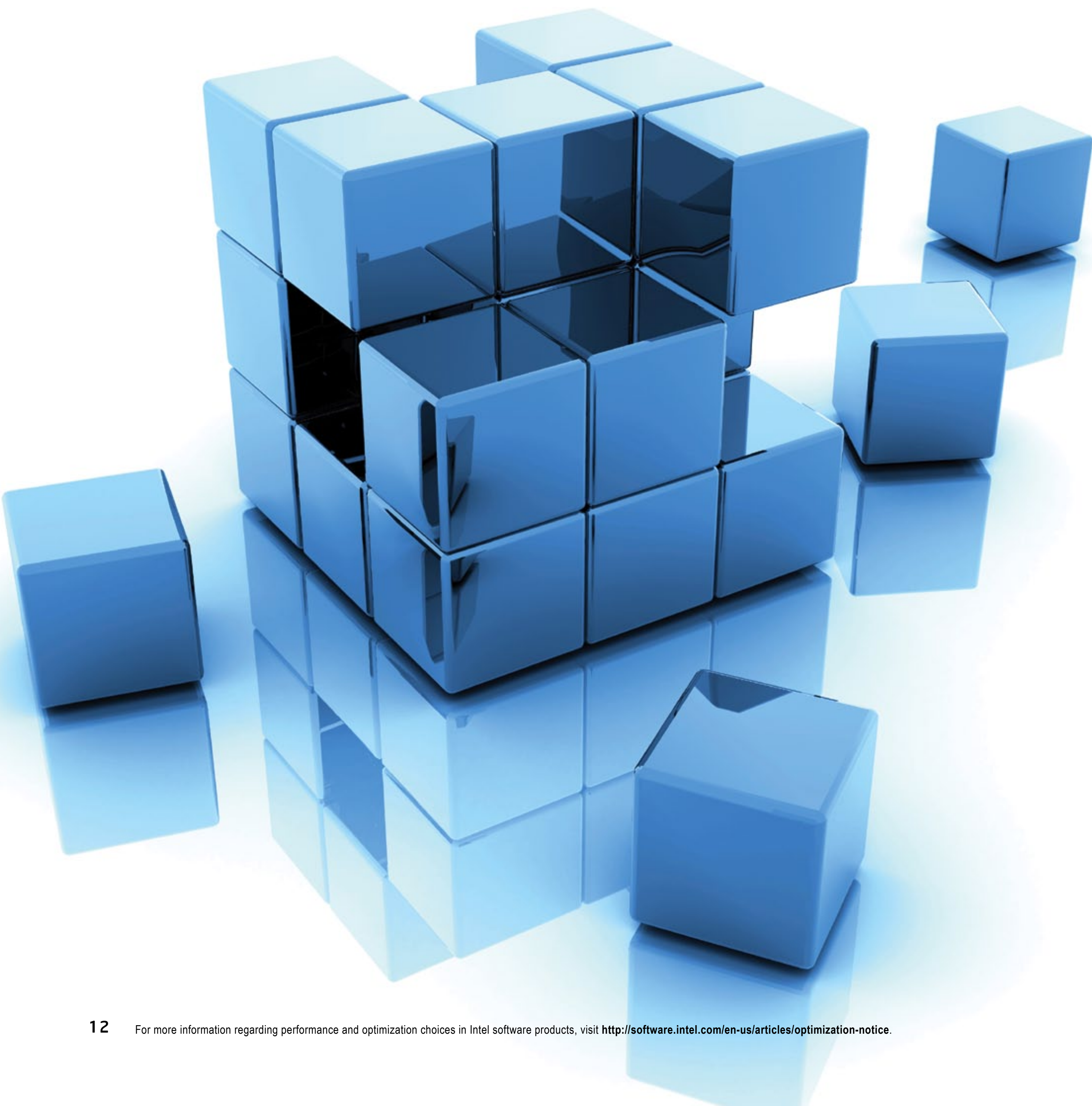
The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: **http://www.intel.com/design/literature.htm**

INTEL® ARRAY BUILDING BLOCKS CODE TIPS

by Zhang Zhang



For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

Intel® Array Building Blocks (Intel® ArBB) is a parallel programming solution designed specifically for dynamic data parallelism. It is the newest component of Intel® Parallel Building Blocks (Intel® PBB), which currently also includes Intel® Threading Building Blocks (Intel® TBB) and Intel® Cilk™ Plus.

What makes Intel® ArBB unique is a runtime system that is capable of doing retargetable and dynamic compilation. The runtime system includes a virtual machine (VM) with a JIT compiler that can adapt to the target architecture by generating parallelized, vectorized, and optimized code dynamically. On top of this VM, Intel ArBB is implemented as an embedded language whose syntax is implemented as a C++ library API compatible with the ISO C++ standard. The programming model imposed by this API supports structured, deterministic, and race-free data parallelism.

Developers programming in Intel ArBB focus on algorithms. They do not need to deal with low-level architectural details, such as SIMD instructions, thread management, and intrinsic functions, to get good performance. The runtime system takes care of those. However, there are certain things that a programmer can do to assist the runtime system in generating high-performance code. There are also areas in the Intel ArBB API that need to be well-understood in order to avoid errors. This article summarizes these tips and explains them using code examples.

CODE TIP 1:
Expressing Parallelism

The first question a programmer who wants to code an algorithm using Intel ArBB asks is, "How do I express parallelism?" Designed to be a language suited for data parallelism, Intel ArBB allows parallelism to be expressed at a high level in two ways.

1. Using vector operations directly on data collections

A vector operation is a function that takes containers as parameters. Parallelism is achieved when using these operations because computations on the elements of a container are either independent or have known parallel implementation strategies. As an example, consider the computation of the Euclidean distances between two points in an m-dimensional space. The distance between these points is calculated using the formula:

$$d = \left(\sum_{i=1}^{m} | x_i - y_i |^2 \right)^{1/}$$

Suppose we also have n points. In this case, the result will include n distances.

One way to code this is to use 2-D dense containers to represent a collection of points, each of which requires multiple coordinates. The width of the 2-D dense containers can be n, the number of points, while the height can be the dimensionality m of the points. **Figure 4** shows the code. The distance function reads in and operates on two containers (X and Y) as a whole, and writes output to another container (d) as a whole.

2. Applying elemental functions to data collections

An elemental function (i.e., a kernel function) is written to operate on individual data elements, as opposed to data collections. An elemental function can be invoked, in parallel, on all elements in a container by

```
std::size_t length = N, dim = M;
dense<f64, 2> vec_X(length, dim), vec_Y(length, dim);

void distance(dense<f64, 2> X, dense<f64, 2> Y, dense<f64>& d) {
    dense<f64, 2> sqdiff = (X ñ Y) * (X ñ Y);
    dense<f64> s = add_reduce(sqdiff, 1);
    d = sqrt(s);
}
```

Figure 4. Use vector operations to express parallelism.

using the arbb::map operator to map the function across the container. **Figure 5** shows an example of computing the Mandelbrot set in the complex plane. The kernel function evaluates a quadratic polynomial for an individual complex number. Then, the arbb::map operator takes a pointer to this kernel function as argument, mapping it across all the elements of the 2-D dense containers D and C used. Note that the "_for" used here is a serial control flow construct acting on one element of the input array, not a parallel for.

CODE TIP 2:

Programming for Performance

The Intel ArBB virtual machine performs many types of optimizations during code generation. All are transparent to the user. However, a set of best practices can be adopted by users to allow more optimization opportunities and to assist the virtual machine in generating efficient code.

1. Use arbb::expect_size to achieve good memory allocation and cache-friendly data distribution.

Intel ArBB has a segregated memory management model in which Intel ArBB container objects are allocated and managed in a memory space separated from regular C++ objects. This is for both data safety and performance reasons. In cases where the size of a container is known before the container is passed to a function, the runtime can pre-allocate the memory and can slice and distribute the data in a way that leads to efficient cache usage, vectorization, and multithreading. All that is needed is for the user to communicate the size expectation to the JIT compiler using arbb::expect_size. Listing 3 shows examples of giving shape expectations for 1-D, 2-D, and 3-D dense containers.

The arbb::expect_size statements must be the first statements in a function that takes parameters of Intel ArBB containers. Sizes must be given for each dimension of a container. Sizes can be specified as integer constants, C++ variables, or expressions that evaluate to an integer. Note that sizes can be computed at runtime, but the size used to construct an Intel ArBB closure and the actual size used when the closure is invoked must match.

```
arbb::expect_size(1d_dense, width);
arbb::expect_size(2d_dense, width, height);
arbb::expect_size(3d_dense, width, height, depth);
```

Figure 6. Give shape expectations.

2. Prefer memory managed by Intel ArBB to avoid unnecessary data copying.

There are two ways to create a container: Bind an Intel ArBB handle to an existing C++ array, or let Intel ArBB create it entirely in the managed memory space. The first approach makes it easier to work with existing C++ data, but it requires the runtime to both copy the data into the managed memory space before an operation, and copy the data back into the original space at the end of an operation. The extra data movements do have performance impacts, and they may also require extra synchronization.

On the other hand, using Intel ArBB managed memory allows the associated data to reside in the runtime memory space during the entire lifetime of the container. No data copying between the managed memory space and the regular memory space takes place unless explicitly specified. Also, the runtime can better manage alignment requirements (which may, in fact, be hardware platform specific). **Figures 7 and 8** show examples of both approaches.

```
static const int max_count = 1000;

// kernel function
void mandel(i32& d, std::complex<f32> c) {
    i32 i;
    std::complex<f32> z = 0.0f;
    _for (i = 0, i < max_count, i++) {
        _if (abs(z) >= 2.0f) {
            _break;
        } _end_if;
        z = z*z + c;
    } _end_for;
    d = i;
}
// use map() to invoke the kernel function
void doit(dense<i32, 2>& D, dense<std::complex<f32>, 2> C) {
    map(mandel)(D,C);
}
// from within C++ code
call(doit)(dest, pos);
```

Figure 5. Use the map operator to apply an elemental function to each item in a data collection.

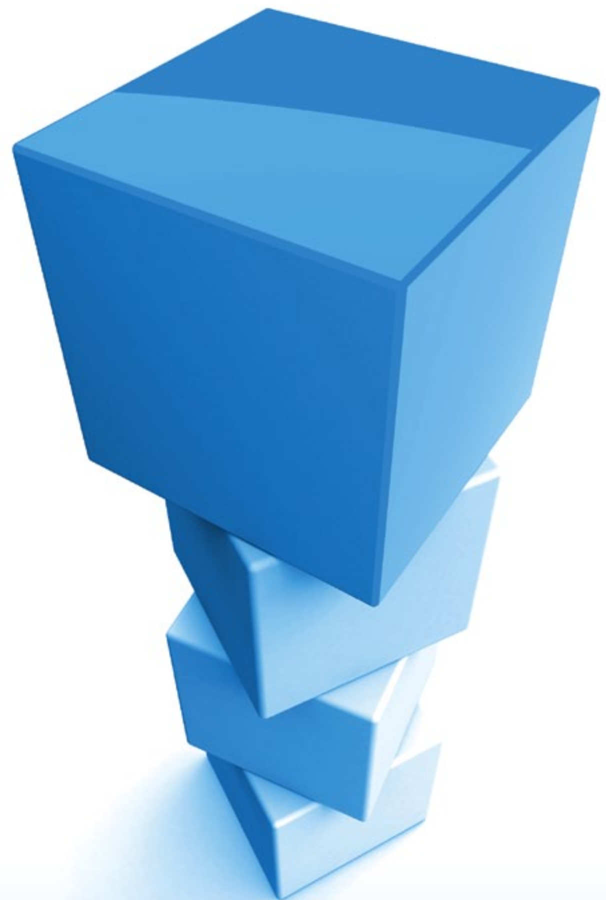
http://software.intel.com/en-us/articles/optimization-notice

```
dense<f32> a(1024), b;
// init 'a' using write range
range<f32> ra = a.write_only_range();
for (size_t i = 0; i < ra.size(); ++i) {
    ra[i] = . . .;
}

// copy in of 'a'
call(fun1)(a, b);
// NO copy out of 'b'

// NO copy in of 'a'
call(fun2)(a, b);
// NO copy out of 'b'

// get data out of 'b' when we need it
const_range<f32> rb = b.read_only_range();
std::vector<float> result(rb.size());
std::copy(rb.begin(), rb.end(), result.begin());
```

Figure 7. Prefer memory managed by ArBB.

```
dense<f32> a, b;
bind(a, c_arr, 1024);
bind(b, c_brr, 1024);
// copy in of 'a'
call(fun1)(a, b);
// sync; copy out of 'b'
// copy in of 'a'
call(fun2)(a, b);
// sync; copy out of 'b
```

Figure 8. The bind interface is convenient but may result in extra copies.



3. Use collective operations sparingly.

Collective operations such as reductions, scans, gathers, scatters, and permutations are necessary for implementing many algorithms. However, collective operations typically introduce irregularities into the computation flow that may require synchronization. Synchronization can significantly reduce chances for the runtime to do optimizations across a big chunk of code. The rules of thumb for using collective operations are:

- They should be used sparingly and only if necessary.
- If possible, move a collective operation to the beginning or the end of the function that invokes it.

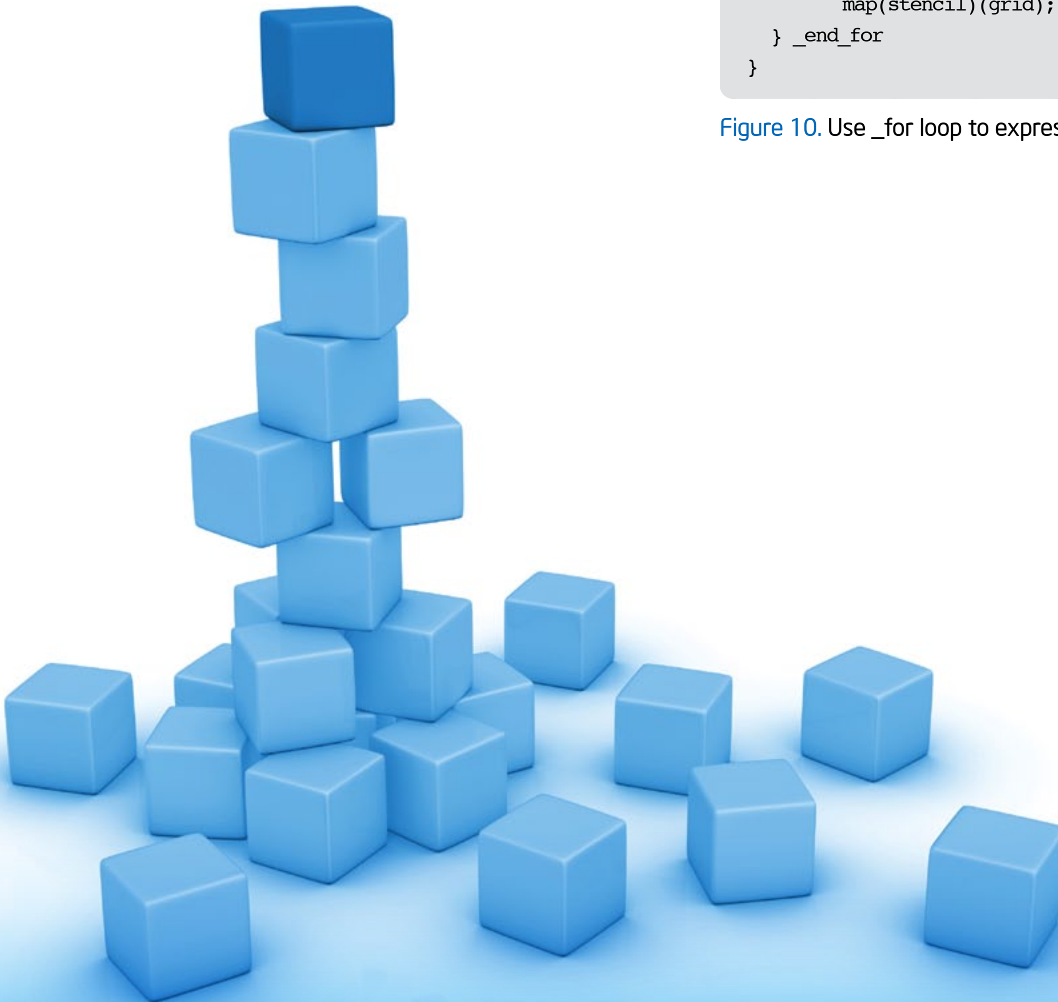
4. Use the most specific function to avoid unnecessary data dependencies.

To help the runtime generate efficient code, programmers should avoid writing code that introduces unnecessary data dependencies. This means two things. First, if an operation can be accomplished in one step using a specialized operator, then it should not be done using multiple steps with more general operations. Second, intermediate copies of containers should be avoided as much as possible. In particular, don't unnecessarily reuse a temporary variable, especially if you are only updating part of a collection. Introduce a new temporary for logically disjoint operations.

Figure 9 shows three equivalent ways to restrict values in a container to a range. The first two ways each need two statements. However, the first way introduces shadow intermediate copies of the container, which are only partly updated, and the second way has unnecessary serial dependencies. By contrast, the third way achieves the same effect using one specialized statement and has a better chance to yield optimal code.

```
dense<f32> x = ..;

// dep. between 'x' copies
x = select(x < 0.0f, 0.0f, x);
x = select(x > 255.f, 255.f, x);

// dep. between 'x' copies
x = max(0.0f, x)
x = min(255.f, x);

// Optimal
x = clamp(x, 0.0f, 225.f)
```

Figure 9. Avoid unnecessary data dependencies and use the most specialized operation available.



CODE TIP 3:

Use _for Properly for Serial Control Flow

Users sometimes have a false impression that the _for loop is parallelized. In fact, the _for loop, and all other loop constructs provided by Intel ArBB, are serial loops. Iterations of a _for loop are executed sequentially. Intel ArBB does not auto-parallelize _for loops. In other words, the _for loop cannot be used to express data parallelism. Refer to Code Tip 1 to see how to properly express parallelism in Intel ArBB.

The _for loop is designed to express serially dependent iterative computation. This is the case in which a computation must be done incrementally, with the current step depending on the result of the previous step. A good example would be an iterative stencil computation (like the one found in heat dissipation), as shown in **Figure 10**. In this code, each stencil-based update step is parallelized through the use of the arbb::map operator. But the updating must be done multiple times in a sequence in order to compute the solution over time.

```
void apply_stencil(dense<f64, 2>& grid, i32 iterations)
{
  _for(i32 i = 0, i < iterations, ++i)
  {
        map(stencil)(grid);
  } _end_for
}
```

Figure 10. Use _for loop to express serial iterative computation.

For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

CODE TIP 4:
Use Intel ArBB Expressions for Intel ArBB Control Flow

The control flow statements (_for, _while, _do_until, _if/_else, etc.) in Intel ArBB must be constructed using condition expressions, just like their counterparts in C++. However, the condition expressions must be Intel ArBB types. It is a common error to use a conditional variable of C++ type. For example, **Figure 11** shows the wrong way and the correct way of constructing a _for statement. To understand this, remember that statements with C++ types are executed once and for all at the JIT time (i.e., capture time). The effect of these statements is "backed" into the generated code and will not change during the subsequent Intel ArBB execution.

```
_for(int i = 0, i < iterations, ++i) { // WRONG
  . . .
} _end_for

_for(i32 i = 0, i < iterations, ++i) { // CORRECT
  . . .
} _end_for
```

Figure11. Do not use C++ type for condition variables.

CODE TIP 5:
Properly Exit from Intel ArBB Control Flow

Another common misuse related to Intel ArBB control flows is to use a return statement to jump out of a _if/_else branch or to break out from an Intel ArBB loop. Using return statements inside Intel ArBB control flow statement is not supported. This leads to runtime errors since the Intel ArBB control flow statements are not properly nested. Fortunately, there are easy solutions for this situation.

For loop constructs (_for, _while, and _do/_until), Intel ArBB provides the _break statement that can be used to properly terminate a loop. For _if/_else branches, users need to restructure the code to avoid the need of a return statement. Examples in **Figure 12** depict both the _break statement and the proper way of jumping out of a branch.

BLOG highlights

TBB 3.0 and processor affinity

BY ANDREY MAROCHKO

A week ago I started telling you about a couple of new, helpful features in the TBB 3.0 Update 4 task scheduler, and we talked about the support for processor groups—an extension of Win32 API available in 64-bit edition of Windows* 7. The main purpose of processor groups is to extend Win32 capabilities to allow applications to work with more than 64 logical CPUs. By its nature, this functionality is pretty close to the classic processor affinity concept, essentially turning a flat processor affinity model into a hierarchical one. First, you affinitize a thread to a processor group, and then you can apply a conventional affinity mask to tie the thread to a subset of CPUs in the current group.

Such an obvious relation between the two concepts reminded me of one idea that TBB team kicked around some time ago. Truth be told, that idea was pushed upon us by the issue one of our customers faced when trying to fuse TBB and MPI on a NUMA system. Their MPI processes were running on multicore NUMA nodes, and they wanted to parallelize computation inside the MPI process by means of TBB. The problem they stumbled upon was that TBB, when initialized by default, instantiated its worker thread pool in accordance with the total system concurrency, disregarding the number of cores on the current NUMA node. Naturally, this resulted in huge oversubscription and performance plunge.

Fortunately, in this particular case the developers had full control over the TBBfied part of the code, so they were able to resolve the issue by specifying desired amount of threads via tbb::task_scheduler_init constructor. But what if your code relies on task scheduler auto-initialization? Or, even worse, you do not even suspect that TBB is present because it is internally used by a third party component, which most probably initializes it by default (only if it is not configurable through a fancy API)?

```
// Break out of a loop using _break
_for(int i = 0, i < iterations, ++i) {
  . . .
  _break;
} _end_for

// This leads to runtime error
_if(x == 0) {
  return;
} _else {
  . . .
} _end_if

// This is OK
_if(x != 0) {
  . . .
} _end_if
```

Figure 12. Properly exit from Intel ArBB control flow statements.

CODE TIP 6:
Only Inspect the Values of Intel ArBB Variables in Debug Mode

An Intel ArBB variable is an opaque handle that will hold a value only when the generated code is executed. It does not have any value at "capture" (i.e., JIT) time. Therefore, using printf or std::cout statements to inspect Intel ArBB variable values inside functions invoked from inside a "call" does not work because these statements are executed only at JIT time when they don't have values (see **Figure 13**). The proper way to inspect the value of an Intel ArBB variable is to use a debugger, or to use printf or std::cout statements with the environment variable ARBB_OPT_LEVEL being set to O0. These are equivalent since both debugging and ARBB_OPT_LEVEL=O0 put the execution into "emulation mode." In this mode, Intel ArBB operations are emulated using regular C++ execution rather than JIT compilation. This mode is just for debugging purposes and so does not provide performance benefits. After inspecting the values, ARBB_OPT_LEVEL should be switched back to O2 or O3 to get performance benefits. **Figure 13.**

CODE TIP 7:
Implicit Type Conversion Is Not Supported

C/C++ programmers are often surprised at first that Intel ArBB does not perform automatic type conversions for Intel ArBB types. For example, **Figure 14** shows a function definition that tries to multiply a container of floating points by an integer factor. If it were written for C/C++ types, then the compiler would automatically "promote" the integer factor to a float pointing value. However, code in Intel ArBB is required to have an explicit cast for the integer factor.

Intel ArBB intentionally does not support implicit type conversion because it is expensive in the context of vectorization (since this may require lane changes, generations of intermediate values, and the like), and it is error prone. Basically, implicit casts would invisibly introduce extra and potentially expensive operations, and the philosophy of Intel ArBB is that any expensive operation should be explicit so it can be controlled. Programmers can always do an explicit cast as shown. Both C-type and C++ casts work, but the C++ cast (as shown in the example) is generally preferred. Note that this rule also applies to constants, so constants should have the correct type. For example, a single-precision, floating-point value for "one" should be written as 1.0f, not 1.0. The latter has a double-precision type. **Figure 14.**

```
void arbb_func(i32 var) {
  . . .
  printf (ì%dî, var);           // Does not work in O2, O3
  printf (ì%dî, value(var));    // Does not work in O2, O3
  . . .
}
```

Figure 13. This use of printf does not work in O2 and O3 modes.

```
void arbb_func(i32 factor, dense<f64>& vec) {
  vec = vec * factor;                           // WRONG
  vec = vec * std::static_cast<f64>(factor);    // CORRECT
}
```

Figure 14. Intel ArBB does not support implicit casting.

 For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

CODE TIP 8:

Be Aware of the Indexing Conventions for Multidimensional Intel ArBB Containers

Multidimensional containers have row-major order layout in memory. However, the indexing conventions are different for Intel ArBB and C/C++. Specifically, a multidimensional container in Intel ArBB (that is, a 2-D or 3-D dense container) is always indexed in the order of width, height, and depth. This has implications in at least the following situations:

- When declaring or binding a multidimensional container that requires the size of each dimension, width is specified before height and height before depth. You can think of this as an x, y, z order. See **Figure 15** for examples.
- When accessing an individual element using indices, the column index goes before the row index, and then the page index follows after that. See **Figure 16** for an example.
- When accessing a neighboring element using the arbb::neighbor operator, the column offset goes before the row offset, followed by the page offset. See **Figure 17** for examples.

```
void stencil(f64& e) {
  f64 N = neighbor(e, 0, -1);
  f64 S = neighbor(e, 0, +1);
  f64 W = neighbor(e, -1, 0);
  f64 E = neighbor(e, +1, 0);
  . . .
}
```

Figure 17. Access neighboring cells in a 2-D dense.

Summary

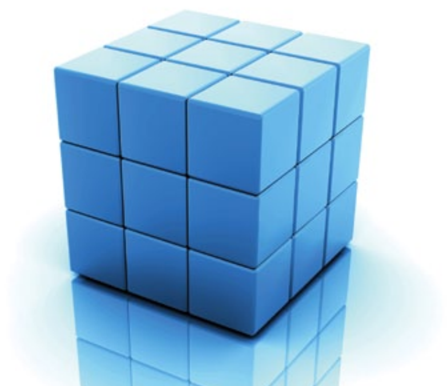
Intel ArBB offers an interface that makes it easy to write code to operate in parallel over collections of data. The syntax allows structured, readable, and maintainable code, and decreases development time. The runtime provides performance portability and scalability across SIMD units, multiple cores, and even accelerator devices. A good understanding of the syntax and proper usage can help us harness the full potential of productivity and performance. ☐

```
double c_mat[NROW][NCOL];

dense<f64, 2> matA(NCOL, NROW);   // Same shape as c_mat

dense<f64, 2> matB;
bind(matB, &c_mat[0][0], NCOL, NROW);// Note the order of NCOL and NROW
```

Figure 15. Declare and bind a multidimensional container.

```
void arbb_func(dense<f64, 2>& m) {
  . . .
  f64 e = m(2, 3);
  . . .
}
```

Figure 16. Access an individual element of a 2-D dense.



For more information about Intel ArBB programming techniques, see the links below:

Intel® ArBB Knowledge Base: http://software.intel.com/en-us/articles/intel-array-building-blocks-kb/all/1/

Intel® online forum: http://software.intel.com/en-us/forums/intel-array-building-blocks/

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: **http://www.intel.com/design/literature.htm**

Helping ensure the best video quality and performance

Intel® Parallel Studio XE helps Envivio create safe and secured code

envivio®

Company: **Envivio Inc.**
Industry: **Video Compression**

"Having such a tool this early in the development stage frees the validation from trivial bug reports and gives our engineers the opportunity to code more efficiently from the very beginning of the product cycle."

Jean Kypreos
ADVANCED VIDEO PROCESSING TEAM MANAGER
Envivio

Company

Any video—Anytime, Anywhere. Envivio's goal is to make the world's video content universally enjoyable by all viewers, on any device, across any network, at any time. Since its founding in 2000 as an inventor of video encoding technology, Envivio has amassed dozens of patents, pioneered video-over-IP methods, and continually leads with support for emerging technology. Today Envivio* solutions deliver millions of content streams to hundreds of different styles of mobile phones, set top boxes, and PC platforms, on behalf of content owners, telecomm operators, cable and satellite companies, and mobile service providers in every market in the world. Using codecs from standard H.264 through the Elite and Extreme codecs developed in the Envivio labs, Envivio optimizes the viewing experience for every screen, on every network. Deployment and support are backed by a global staff that is familiar with local technical, regulatory, and market guidelines encountered around the world. Envivio is headquartered in San Francisco, and has offices in Rennes, France; Beijing; Tokyo, and Singapore. For more information, visit www.envivio.com.

Mission

To grow the quality of the 4Caster C4* product to achieve the best video quality, best performance, and fastest time to market

Product

4Caster C4 is a real-time transcoding application that supports one channel of high-definition or multiple channels of standard-definition encoding for IPTV, Internet TV encoding up to VGA resolution, and 3G Mobile TV encoding.

Challenge

As code size grows, a simple IDE is no longer enough to properly follow large sections of code. Larger code generates more bugs of an increased complexity, compromising reliability and time to market.

Impact

The likelihood that a bug will return to the encoding team has been lowered, while both developer ramp-up time and time to market have improved.

 For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

ROUNDUP

Why Envivio's products benefit from utilizing parallelism

Depending on the formats, several cores are required to perform a single encoding. In particular, for high-definition content repurposing from MPEG2 to H.264, a single core is not enough to achieve the encoding task in real time. In that case, both multicore and multiprocessor (DP) technologies are required to provide the minimum horsepower for the application.

For standard-definition content, a high level of video quality is required for the broadcast market segment. This quality is obtained using heavy-processing algorithms while keeping the real-time constraint. At the same time, in order to keep a modular level of density (i.e., 2/4 SD channels per platform), a parallelization (2/4 cores per channels) schema is mandatory to take advantage of the platform capabilities.

The density problematic is induced by the economic equation of the BOM cost: more channels per platform lead to smaller COGS. In Envivio applications, the H.264 codec is the most "time-critical" software component. Particular care is taken in the parallelization balance of this module.

Results

Using the static security analysis (SSA) capability in Intel® Parallel Studio XE to analyze sample code, all the falsely inserted bugs were detected. SSA was then introduced into the process with a real test case: the encoding core of the 4Caster C4. Based on the results of the analysis, some sections of unused but executed code were removed, thus improving performance. Sections of unused but executed code were removed, improving performance.

Even if no examples of potential memory corruption were found, many areas of the code saw improved safety, adding error management and tracking the potential null pointer errors. Special care was taken with the in-and-out interface as the last test for SSA allowed the front end of the encoder to be secured. Misuse in the product of this critical module will result in immediate detection.

On its first launch, SSA found approximately 300 errors, most related to potential pointer usage. Five critical bugs were discovered, in addition to three minor bugs and approximately 50 vulnerabilities. As a result, some detected areas were rewritten, most of the dead code was eliminated, and many critical pointers were secured.

Globally, more reliable code was delivered to the integrators, minimizing the possibility that a bug will return to the encoding team.

How Intel® Software Development Products assisted

The SSA capability in Intel Parallel Studio XE was used to analyze the core module of the product: the H264 encoder. It was set to its maximum sensitivity, helping reduce the floating validation time, while improving product quality and facilitating on-time delivery.

SSA also offers more resilience to the code. This protection promoted developer confidence in the code, making it safe to connect one piece of software to the other. Safe and secured code will return errors immediately, preventing long, demanding debugging sessions. Used in the early stages of development, SSA fixed bugs before they appeared, saving time and resources. By combining with Intel® Inspector XE, also in the early stage of development, greater dynamic error resilience was achieved.

The SSA tool was easy to set up, launch, and use. And because results are displayed within Intel Inspector XE, there was only a single interface to master. In addition, access to the source code was fast, and problems could be found and fixed within minutes.

SSA is fully integrated with the compiler, and results could be manipulated in many ways. The user-friendly interface and filters enabled the developers to focus on a category of problems, while the documentation clarified tricky and advanced coding issues.

Enabling an immersive and realistic PC game experience

The Creative Assembly relies on Intel® Software Development Products to conquer threading and graphics challenges in new Napoleon: Total War game

Company: **The Creative Assembly**
Industry: **Gaming**

"Two things drew us to Intel® TBB. First and foremost was the relative ease with which it could be integrated into the existing serial code, and second was the speed at which programmers can learn to use it."

Yuri O'Donnell
SENIOR ENGINE CODER
The Creative Assembly

Company

Founded in 1987, The Creative Assembly is a leading computer games developer based in Sussex in the UK, and is best known for the Total War* series of strategy games. Napoleon: Total War* is the fifth major release in the franchise and, like its predecessors, is a PC-only release. The company has also had major successes in the console market with Spartan: Total Warrior* and Viking: Battle for Asgard*. From 2003 to 2005, the Total War* engine was used for the BBC game show, Time Commanders, which won international acclaim. In 2005, The Creative Assembly was acquired by SEGA. For more information, visit www.creative-assembly.co.uk.

Mission

Marry real-time strategy battles with the slower, turn-by-turn tactical overview of a campaign map to create realistic, immersive games people want to play.

Product

Napoleon: Total War is the first in an all-new story-driven branch of The Creative Assembly's multi-award-winning Total War RTS franchise. Napoleon: Total War will keep the franchise's genre-leading 3D battles on land and sea. The turn-based campaign is split into three different story-driven campaigns, telling the story of the rise and fall of Napoleon Bonaparte through his most famous battles.

Challenge

Harness the power of multicore PCs to support development of an immersive and realistic game environment.

Impact

The Creative Assembly programmers can scale to as many cores as provided, which, for strategy games, frees them from being CPU bound, while future-proofing the engine.

Why The Creative Assembly's products benefit from utilizing parallelism

The more processing and graphics power that can be devoted to a game, the better it looks and the greater the sense of immersion. The heavy customization of individual details in Napoleon: Total War was made possible by multithreading techniques. Although it will run on a single-core processor, the game engine has been coded with a dual-core or greater target in mind, and is split into two parts: logic and rendering. These routines are further subdivided into individual "mini-task" algorithms, such as sea geometry refreshes and pathfinding. The idea is that no one process holds up another and the game can take advantage of available hardware.

Results

Napoleon: Total War achieved a more realistic user experience and significantly increased the fidelity of the visual experience by adding greater animation, all while delivering lower system requirements than its predecessor. The team is confident that it would not have been able to ship on time without Intel® Threading Building Blocks (Intel® TBB).

The efficiency of the engine means that the battlefields and naval encounters in Napoleon can be vast and heavily populated, with individual AI routines for every unit. Napoleon includes battles with 10,000 to 20,000 men, each one created with a level of detail comparable to that of games with only 10 to 20 characters on screen. One of the key new tools for Napoleon is a custom unit builder, which allows designers to add details to the models. Napoleon represents the most advanced incarnation of the Total War series, and yet boasts lower system requirements than its predecessor.

How Intel® Software Development Products assisted

The Creative Assembly team drew heavily on the experience of Intel® engineers and Intel® tools when designing their current gaming engine.

Intel® Thread Checker helped validate every memory operation undertaken by The Creative Assembly team. By notifying them of any simultaneous access to shared memory, race conditions could be identified and remedied. Even more useful were the Intel TBB libraries, which developers used to parallelize their code. Not only was it relatively easy to integrate into existing serial code, the programmers could quickly learn how to use it. Within days it was yielding performance benefits.

One of the advantages of targeting the PC, rather than writing for a multiplatform audience, was that Intel TBB could be completely integrated into the engine, saving an enormous amount of time and money. The Creative Assembly avoided having to write its own job queue system, which saved effort on development, testing, and debugging. The team is confident that it would not have been able to ship on time without Intel TBB.

In order to reduce the system requirements, the team turned to Intel® Graphics Performance Analyzers. Developers got the timing of each API call and saw how long it took on the GPU. This enabled them to modify the render state and determine the performance impact. The tool's real-time analysis of the rendering pipeline helped identify bottlenecks, and spot instances of overdraw in any scene.

Creating a new standard in virtual crash testing

Altair advances frontal crash simulation with help from Intel® Software Development Products

Company: **Altair Engineering, Inc.**
Industry: **Manufacturing and Industrial**

"This breakthrough delivers the missing link for CAE-driven design in vehicle safety. This, combined with our new hybrid solver approach, has enabled us to eliminate the turnaround time bottleneck inherent to virtual crash testing. Now, multi-disciplinary optimization for crash, durability, and NVH will be able to provide valuable input to the design process."

Dr. Uwe Schramm
CTO FOR HYPERWORKS
Altair Engineering, Inc.

Company

Altair Engineering, Inc. empowers client innovation and decision making through technology that optimizes the analysis, management, and visualization of business and engineering information. Privately held, with more than 1,300 employees, Altair has offices throughout North America, South America, Europe, and Asia-Pacific. With a 25-year track record for product design, advanced engineering software, and grid-computing technologies, Altair consistently delivers a competitive advantage to customers in a broad range of industries. Built on a foundation of design optimization, performance data management, and process automation, HyperWorks is an enterprise simulation solution for rapid design exploration and decision making. To learn more, please visit www.altair.com and www.altairhyperworks.com.

Mission

Write the first simulation code to solve a full vehicle frontal crash simulation, with more than 1 million elements in less than five minutes.

Product

RADIOSS is a next-generation finite element analysis (FEA) solver for linear and non-linear simulations. It can be used to simulate structures, fluids, fluid-structure interaction, sheet metal stamping, and mechanical systems. This robust, multidisciplinary solution allows manufacturers to maximize durability, noise and vibration performance, crashworthiness, safety, and manufacturability of designs in order to bring innovative products to market faster.

Challenge

Use state-of-the-art hybrid programming mixing different parallelization techniques to achieve more scalability and deliver optimal performance for very large number of processors.

Impact

Altair improved customer satisfaction by exceeding customer performance and timeline requirements.

For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

Why Altair's products benefit from utilizing parallelism

Virtual crash tests are one of the most time-consuming tasks in the automotive development process. Altair wished to demonstrate the feasibility of a new analysis process that drastically reduces the simulation time needed for virtual crash testing. In terms of the product lifecycle, it sought to reduce both prototyping costs and time to market, two key competitive advantages for Altair customers.

Massively parallel programming enables very good scalability using domain decomposition techniques and an MPI communication library; such scalability tends to decrease as the number of processors increases and the amount of data to compute decreases.

Altair used state-of-the-art hybrid programming mixing different parallelization techniques (MPI and OpenMP) to achieve more scalability and deliver optimal performance at very large number of processors. To accomplish their goal, Altair leveraged the optimization made possible by Intel® compilers, libraries, and tools to sustain the required efficiency. There exists no real alternative to achieve the necessary level of performance.

Results

The five-minute goal was successfully exceeded (294s achieved using 1024 cores [128 MPI x 8 OpenMP) and overall performance was increased by 10x. As a result of hybrid programming, scalability of the code demonstrated up to 1024 cores. In addition, a new numerical algorithm, called Advance Mass Scaling (AMS), decreased computational costs.

Altair enjoyed improved customer satisfaction by exceeding current customer performance and timeline requirements by enabling faster design variant evaluations and less time-consuming design sensitivity and robustness analysis. For the market, this advancement can be considered a disruptive approach by cutting simulation time from the hours previously required to minutes, making possible new processes and advancements in virtual crash testing such as finer meshes, integration of better material laws with rupture, optimizations, and scatterings, etc. By decreasing the delay of virtual crash testing, both time to market and costs can be drastically decreased.

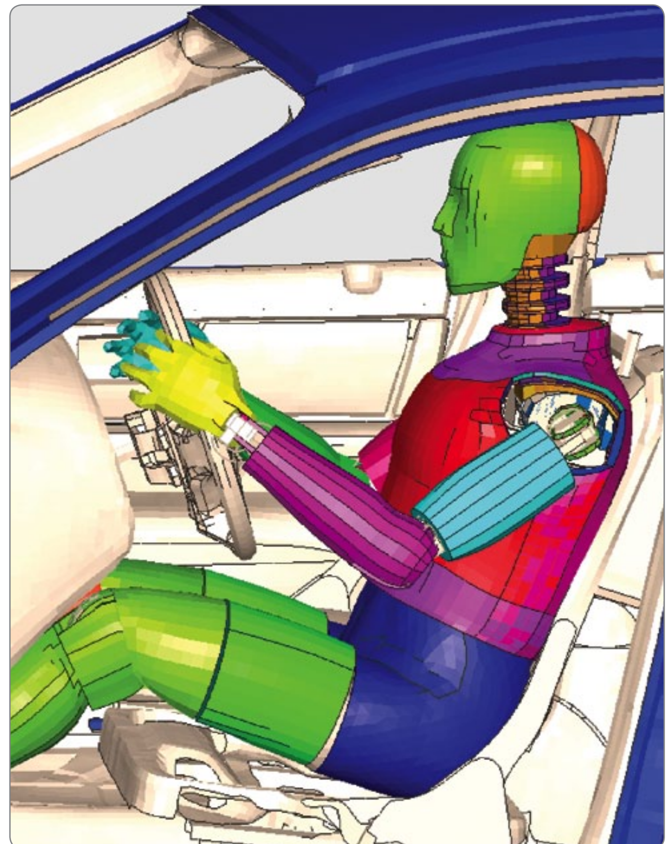
How Intel® Software Development Products assisted

To meet its speed, scalability, and performance objectives, Altair incorporated a range of Intel Software Development tools, including the following:

- Intel® Compilers
- Intel® VTune™ Analyzer (now called Intel® VTune Amplifier™ XE)
- Intel® Cluster Toolkit
- Intel® Trace Analyzer and Collector
- Intel® MPI Library

Altair and Intel were also able to leverage the Intel® Xeon® processor 5500 series-based clusters and deliver substantial performance improvement for crash simulations. This improvement will have long-term benefits for the industry.

Finally, Intel provided technical help for running and optimization, access to computing resources, and marketing support.



Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

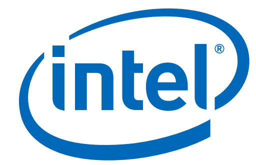
Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

 For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**.

INTEL® HIGH-PERFORMANCE TOOL SUITES



Achieve unprecedented application performance and reliability

Whatever your development goal, Intel has the high-performance tool suite to meet your needs.

Werner Krotz-Vogel
Technical Marketing Engineer

For high-performance computing and enterprise developers

Intel® Parallel Studio XE

Intel® Parallel Studio XE combines Intel's industry-leading C++ and Fortran Compilers; libraries; and error-checking, security, and profiling tools into a single tool suite that helps high-performance computing and enterprise developers maximize application performance, security, and reliability. Intel® C++ Studio XE offers the same benefits for developers who only need the C++ Compiler.

For MPI developers on high-performance clusters

Intel® Cluster Studio

(Formerly Intel® Cluster Toolkit Compiler Edition)

Intel® Cluster Studio is an integrated software package of best-in-class cluster tools that includes Intel® C++ and Intel® Fortran Compilers, optimized performance libraries, Intel® Trace Analyzer and Collector, Intel® MPI Library, and Intel® MPI Benchmarks to help increase MPI cluster performance on Linux and Windows.

Save with suites

When purchased as part of a suite, all components can be had for a significant savings. For details on buying tools individually, visit **http://software.intel.com/en-us/articles/buy-or-renew/**.

Already own the software?

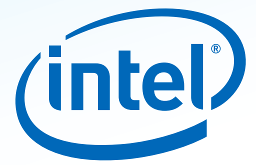
If you currently own Intel® software tools, you could be eligible for special upgrade pricing. For details, visit **http://software.intel.com/en-us/articles/intel-xe-product-comparison/#upgrade**.

Rock your code. **Rock your world.**

Download free 30-day trials of Intel® Software Development Products at **www.intel.com/software/products/eval**.

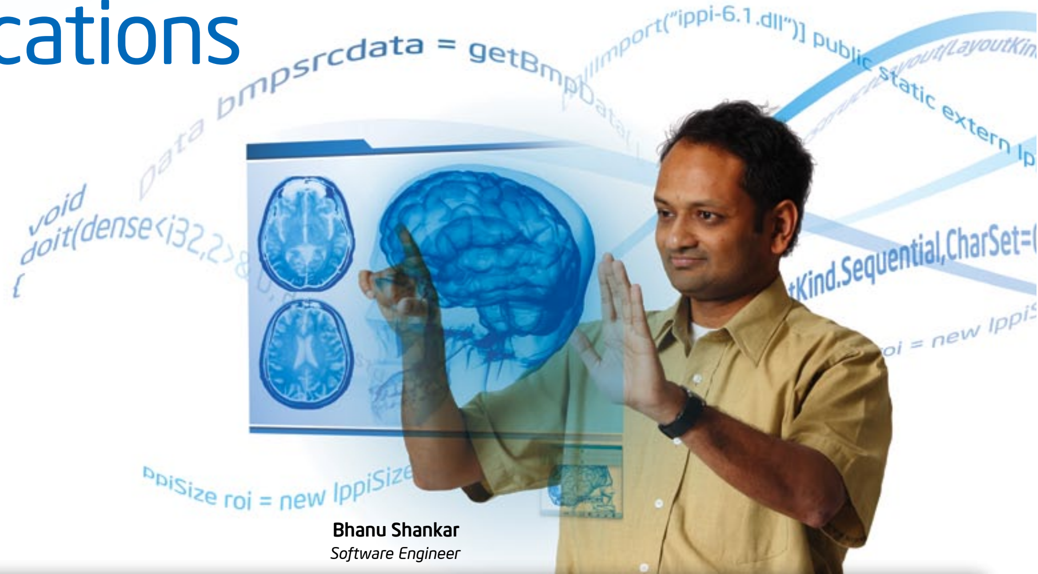
For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**
© 2011, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries. *Other names and brands may be claimed as the property of others.

INTEL® HIGH-PERFORMANCE TOOL SUITES



Create, debug, and boost performance for serial and parallel applications

Whatever your development goal, Intel has the all-in-one tool suite to meet your needs. Find the right suite for you below.



Bhanu Shankar
Software Engineer

Essential Performance

Intel® Parallel Studio

Intel® Parallel Studio provides a comprehensive tool suite that includes an innovative threading assistant, optimizing compiler and libraries, memory and threading error checker, and threading performance profiler for Microsoft Visual Studio* C/C++ developers that want to take advantage of multicore.

Advanced Performance

Intel® Parallel Studio XE

Intel® Parallel Studio XE combines Intel's industry-leading C++ and Fortran Compilers, libraries; and error-checking, security, and profiling tools into a single tool suite that helps high-performance computing and enterprise developers maximize application performance, security, and reliability.

Distributed Performance

Intel® Cluster Studio

(Formerly Intel® Cluster Toolkit Compiler Edition)

Intel® Cluster Studio is an integrated software package of best-in-class cluster tools that includes Intel® C++ and Intel® Fortran Compilers, optimized performance libraries, Intel® Trace Analyzer and Collector, Intel® MPI Library, and Intel® MPI Benchmarks to help increase MPI cluster performance on Linux and Windows .

Save with suites

When purchased as part of a suite, all components can be had for a significant savings. For details on buying tools individually, visit http://software.intel.com/en-us/articles/buy-or-renew/.

Already own the software?

If you currently own Intel® software tools, you could be eligible for special upgrade pricing. For details, visit **http://software.intel.com/en-us/articles/intel-xe-product-comparison/#upgrade**.

Rock your code. **Rock your world.**

Download free 30-day trials of Intel® Software Development Products at **www.intel.com/software/products/eval**.

For more information regarding performance and optimization choices in Intel software products, visit **http://software.intel.com/en-us/articles/optimization-notice**

© 2011, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries. *Other names and brands may be claimed as the property of others.