

***Multithreading Image Analyzer Engine for faster Image detection***

***Multithreading  
Image Analyzer Engine  
for faster Image  
detection***

Version 1.1  
Date 20/04/07

## **Multithreading Image Analyzer Engine for faster Image detection**

### **Disclaimers**

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 by visiting [Intel's Web Site](#). Copyright ©

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, visit [Intel Performance Benchmark Limitations](#)

Hyper-Threading Technology requires a computer system with an Intel® Pentium® 4 processor supporting Hyper-Threading Technology and an HT Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See <http://www.intel.com/info/hyperthreading/> for more information including details on which processors support HT Technology.

[BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, 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, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey 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 © 2007, Intel Corporation. All rights reserved.

## **Table of Contents**

<u>Introduction:</u> .....	4
<u>Tools to address challenges of multithreaded programming:</u> .....	4
Intel compilers: .....	4
Intel® VTune™ Performance Analyzer: .....	4
Intel® Thread Checker: .....	5
Intel® Thread Profiler: .....	5
Intel® Performance Libraries: .....	5
<u>Image Analyzer: Application overview:</u> .....	6
<u>Problem Identification phase:</u> .....	6
<u>Fixing Memory related issues:</u> .....	7
<u>Making the application Multi-core ready:</u> .....	7
<u>Final Performance Summary:</u> .....	8
<u>Conclusion:</u> .....	9
<u>Contributors:</u> .....	9

## ***Multithreading Image Analyzer Engine for faster Image detection***

### **Introduction:**

Now that our industry has moved to multi-core processors, the nature of software development has taken a turn. Near-term gains in application performance will come primarily from threading applications. Many in the developer community expect threaded applications running on multi-core platforms to be catalysts that open new market opportunities. For software companies, threading can be a disruptive technology that causes some companies to gain competitive advantages. Threading in 2006 and beyond is an opportunity to deliver exceptional value through innovative, high performance products and services that differentiates their business from competitors. For corporate developers, threading is the key to getting the most value from investments in client and server infrastructures. Multi-threaded applications running on multi-core platforms will get the work done more efficiently and in less time, which can move their business along at a much faster pace.

### **Tools to address challenges of multithreaded programming:**

#### **Intel compilers:**

The most popular Intel tools—the C++ and FORTRAN compilers—lead the industry in their support of OpenMP\* extensions ([www.openmp.org](http://www.openmp.org)), which help implement parallelism. The advantages of using OpenMP instead of threading by hand are similar to the benefits of using C++ rather than assembly language: programs are easier to maintain and debug, and they provide better performance. Intel compilers can also find parallelism opportunities automatically in sequential code, reducing development labor. Over time, Intel compilers will evolve to support new processor architectures, eliminating the need for developers to recode applications. Reinders advises, "Development managers should encourage the use of OpenMP because the compiler does an enormous amount of work. And that makes developers much more productive."

#### **Intel® VTune™ Performance Analyzer:**

The Intel VTune Performance Analyzer is a powerful and an easy-to-use tool. It collects, analyzes, and displays performance data for a wide variety of applications. The VTune Performance Analyzer can help you identify and locate the sections of code that show the highest amount of activity during a specific period. The VTune Performance Analyzer also displays how an application interacts with an operating system or other software applications, such as drivers. Features such as call graph and sampling make the VTune Performance Analyzer an efficient analyzing tool. The tool helps you tune an application at different levels, such as system, application, and computer architecture.

The Intel® VTune™ Performance Analyzer includes activities and data collectors for collecting the performance-related data. Activities control collection of data. Within an activity, you can specify the types of performance data you wish to collect. For each type of performance data, you need to configure the appropriate data collector

## ***Multithreading Image Analyzer Engine for faster Image detection***

to collect the requested performance data. Different data collectors collect different types of

### **Intel® Thread Checker:**

Intel Thread Checker detects potential threading related bugs even if they do not occur. Therefore, it helps you create correct and safe multithreaded applications. Intel Thread Checker is a debugging tool used on threaded applications. It can detect threading bugs in Windows threads, POSIX threads, and OpenMP threaded applications. It is a plug-in to VTune Performance Analyzer with the same look, feel, and interface as the VTune Analyzer environment. With traditional methods, locating threading bugs may take a very long time. In fact, debugging tools and techniques may even hide the effects of threading bugs, making it even more difficult to establish the cause of errors. However, with Intel Thread Checker, you can reduce the turnaround time for bug detection and isolation. Intel Thread Checker can be used as the following:

- **Design aid:** You can create a prototype application with OpenMP by adding appropriate pragmas. Intel Thread Checker can identify conflicts in the application and generate a report. This report helps you analyze the issues and find solutions during the design phase of your application.
- **Debug aid:** You can identify actual and potential bugs in threaded applications.

### **Intel® Thread Profiler:**

Intel Thread Profiler is a threading tool that you can use to analyze the performance of applications threaded with Windows threads API, OpenMP, and POSIX threads. In Microsoft Windows, Intel Thread Profiler is a plug-in for the Intel® VTune™ Performance Analyzer. Intel Thread Profiler is implemented as a view within the VTune Performance Analyzer. You can use Intel Thread Profiler to perform the following functions:

- Group performance data by categories such as thread, synchronization object, concurrency level, or source location.
- Sort data according to various criteria such as type of activity time, concurrency level, or object type.
- Identify source locations in your code that cause performance problems.

### **Intel® Performance Libraries:**

The Intel multimedia, data processing, and floating-point math libraries are designed to squeeze the greatest possible performance from Intel multi-core platforms. As the number of cores increases over time, future processor innovations are automatically incorporated into applications through the use of Intel® Performance Libraries. Software developers can use Intel Performance Libraries now to exploit dual-Core platforms. Using Intel Performance Libraries, development teams will have little need to recode in the future to support four and eight cores. By letting Intel performance

## **Multithreading Image Analyzer Engine for faster Image detection**

Libraries do that work, software developers are free to focus on areas where they add the most value.

### **Image Analyzer: Application overview**

Image Analyzer is a robust, proprietary, real time image analysis system, which is used for identifying pornographic or inappropriate image content within all popular digital data transmission protocols. The engine has been developed to be simply integrated into an existing security application or used as a base for a new product offering. The design of this application has been modified to a great extent to run best on the latest Intel dual core and Core2 Duo processors. This white paper would cover the detailed methodology to identify potential bottlenecks and hot spots in the application and recommend solutions to overcome these problems. Some of the fixes applied to this application have also been discussed in this white paper. It also discusses as to how multithreading has been accomplished in the application.

### **Problem Identification phase:**

There are various tools that are primarily used for identifying bottlenecks/doing a profile of the application and Vtune has been the tool of our choice for this purpose. We had used sampling and call graph features available in Vtune to identify the different bottlenecks in the application.

The initial performance numbers on a sample of 1000 images on the un-optimized version of the application is reported as follows:

<b>Name of the system</b>	<b># of Cores</b>	<b>Time taken for scanning (in seconds) Engine Ver: 2.1.0</b>
Core 2 Quad	4	8
Core 2 Duo	2	8
Pentium D	2	15
Pentium 4 with HT ON	2	13

After collecting the profile information, it was identified that engine was not multithreaded and some of the functions like the face detection and body part detection were consuming a lot of time. After a deep dive, it was figured out that these functions used a lot of memory allocations unnecessarily. After further investigation it was figured out that there were a lot of memory allocation in the engine which was not optimized and since memory allocation is always single threaded, this hampered the performance of the engine. The CArray (MFC) function that had been self-developed did a lot of memory allocation and reallocation at each addition and deletion of elements in the array. This caused the process to slow when the image contained a lot of body parts.

The next step was to check if the engine by itself was "thread safe". The tool that served handy was the Thread Checker and Thread Profiler. The application again was run through these tools and it was figured out that the engine was not thread safe. Invoking multiple instances of the engine made it to crash and so work had to be

## **Multithreading Image Analyzer Engine for faster Image detection**

done to make the engine thread safe. Thread Profiler was specifically used for finding out the interaction between different threads and figure out if there were any dependencies between them.

### Fixing Memory related issues:

Having identified potential issues in the application, the next step would be to fix those issues. Most of the issues that were identified were related to memory. A lot of code and algorithms had to be rewritten to solve these issues. All the memory allocations were carefully analyzed and it was ensured that the memory allocation was done only when required and had taken in to account some of future requirements. For example, in the memory allocation for arrays, memory allocation was done in chunks of 4K, which reduced the memory re-allocation to almost 5% of the original allocation and reallocation process. This improved the speed of the engine to a great extent. The face detection and the body part detection algorithms were re-written to use minimum memory. Required care was taken in the entire code to ensure that the memory was reused as and when required, instead of deallocating the memory at that instance and reallocating the same at a later instance.

These fixes seemed to have taken care of all the memory related issues.

### Making the application Multi-core ready:

Having fixed all the memory related issues in the single-threaded version of the application, the next step was to make the application/engine multi-core ready. From the profiling and call graphs, it was evident that the application was single threaded and there was lot of scope for threading.

The first step toward threading the application was to build/port the engine using the Intel compilers, which gives us the flexibility to use threading options in the compilers like OpenMP, auto parallelize, etc.

The next question was "Where to thread the application?" Vtune did help us to answer this question and it was finally decided to thread two portions of the code:

1. **The file scanning function** `void fnScanAllFiles()`: This function scans all the images in the given folder and based on the characteristic of the image, it classifies it as a clean image or a porn image. Data decomposition would be rightly applicable here, where the number of files is split and distributed to each of the cores for scanning. After analyzing all the possible threading models, it was decided to use Open MP to parallelize this portion of the code. The sample code for the same is listed as follows:

```
Old Code:  
// Scan files from the list of files passed  
for(int i=0;i<lActualFilesToScan;i++)  
{  
    pParent->fnScanFile(arrFileNames.GetAt(i));  
}
```

## Multithreading Image Analyzer Engine for faster Image detection

### New Code:

```
// Scan files in parallel to use the OpenMP Optimally
#pragma omp parallel for
for(int i=0;i<lActualFilesToScan;i++)
{
    pParent->fnScanFileInParallel(arrFileNames.GetAt(i));
}
```

- 2. Face detection function:** This function was responsible for detecting the face of any given image. Once the face was detected, this portion of the image would not be scanned for coloring and texture. Only the rest of the image would be scanned for detecting if the image is porn or not. This function was identified by Vtune as one of the key hotspots and so it was decided to thread this portion of the code as well. The face detection portion of the code used the Open CV (open source library) libraries for face detection. The application was using a non-threaded version of the Open CV library and so it was decided to replace this library with the threaded version. The threaded version of the Open CV again had implemented the OpenMP threading model in its algorithm.

### Old Code:

```
// Following loop is used to perform face detection
for( pass = 0; pass < npass; pass++ )
{
    for( int _iy = 0; _iy < stop_height; _iy++ )
    {
```

### New Code:

```
// Following loop is used to perform face detection
for( pass = 0; pass < npass; pass++ )
{
    // Using OpenMP we shall make the scanning of the cascade on the image parallel,
    thereby
    // utilizing the advantage of multicore
    #pragma omp parallel for shared(cascade, stop_height, seq, ystep, temp, win_size,
    pass, npass, sum, p0, p1, p2, p3, pq0, pq1, pq2, pq3, stage_offset)
    for( int _iy = 0; _iy < stop_height; _iy++ )
    {
    }
}
```

Threading both these functions did considerably increase the performance of the engine and made the application multi-core ready.

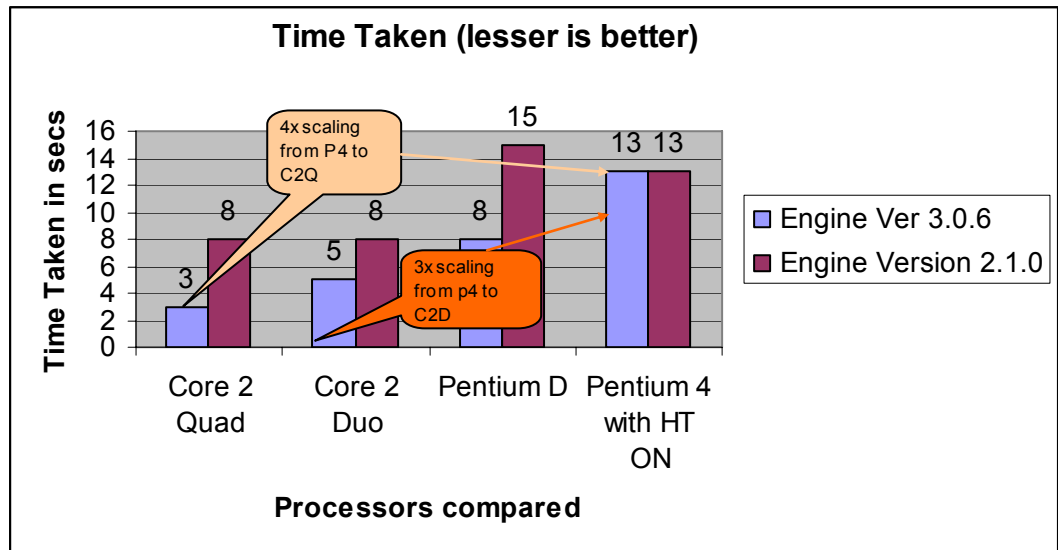
## Final Performance Summary:

The final performance numbers after threading and fixing all the memory issues read as follows:

Name of system	# of Cores	Time in Seconds (Lower Better)		Scaling Gain
		Engine Ver: 3.0.4(Optimized)	Engine Ver: 2.1.0(Non-optimized)	
Core 2 Quad(2.66 Ghz)	4	3	8	2.67
Core 2 Duo (2.66 Ghz)	2	5	8	1.6
Pentium D (3.4 Ghz)	2	8	15	1.87
Pentium 4 with HT ON (3.6 Ghz)	2 (logical)	13	13	1

A graphical representation of the same can be listed as follows:

## Multithreading Image Analyzer Engine for faster Image detection



P4HT- Intel Pentium 4 with Hyper Threading (2 logical cores)

PDP- Intel Pentium D (2 physical cores based on Intel netburst microarchitecture)

C2D- Intel Core2Duo (2 Physical cores based on Intel Core Microarchitecture)

C2Q- Intel Core2Quad (4 Physical cores based on Intel Core Microarchitecture)

As evident from the results, we can see an Out-of-box performance of 2x scaling from Pentium 4 to Core2Duo and Core2Quad using the older version of the engine. Our optimization efforts have further increased the scaling factor to a maximum of 4x from a Pentium 4 to a Core2Quad. As the number of cores increases the application would scale accordingly, thus making it multi-core ready.

### Conclusion:

As we move over to the next generation of computing, multicore is going to play a very vital role and it is very evident from the results that we have discussed in the previous section. Unless and until the software makes use of the hardware features, it would be hard to extract performance. Unlike the previous computing arena, where Out-of-box performance was the mantra, now efforts have to be made optimize the software to extract the best performance from the hardware. Having discussed the future of multicore, there seems to be a lot more scope for parallelizing the Image Analyzer engine/application. Exploring/inventing those opportunities and making the application/engine run faster would make this a great product and put this product well ahead of all of its competitors in the market.

### Contributors:

1. Randhir Shinde- CTO, Image Analyzer
2. Hemant hirve - Lead Software Architect, Image Analyzer
3. Lakshmi Narasimhan- Application Engineer- Intel Corporation
4. Rama Kishan Malladi - Application Engineer- Intel Corporation