



Boosting Application Performance using Intel® Performance Libraries

Executive Summary

The Intel® Performance Libraries consist of the Intel® Math Kernel Library (Intel® MKL) and the Intel® Integrated Performance Primitives (Intel® IPP). These software libraries provide a large collection of pre-built and tested, performance-optimized functions to developers targeting Intel® architecture-based platforms. By using these libraries, developers can reduce the costs and time associated with software development and maintenance, and focus their efforts on their own application code.

The functions contained in the libraries have been carefully optimized to harvest specific performance features built into current Intel processors and will be optimized for future Intel processors. An important advantage of using the Intel Performance Libraries is that they provide transparent portability of application programs across the full range of Intel processors. The libraries are continuously maintained by Intel engineers and are backed by Intel® Premier Support. This paper introduces the benefits of using Intel Performance Libraries to boost application performance and describes opportunities for boosting performance.

Introduction

Completing key processing tasks within designated time constraints can be a critical issue, whether a development team's target application handles real-time audio processing for a battery-operated handheld PDA, or whether it tracks financial market behavior using large-scale computer clusters. Intel offers the following toolsets to support new code development for best performance when targeting Intel processors, as well as to support performance improvements for existing code bases:

- Intel® C/C++ Compiler — produces code that takes advantage of specific performance features built into each of the processors supported
- Intel® VTune™ Performance Analyzer — provides an execution profiling and run-time performance analysis tool
- Intel Performance Libraries — provide a collection of pre-fabricated and tested key functions commonly used within performance-sensitive areas of application programs

This paper focuses on the benefits of using Intel Performance Libraries for boosting application performance.

Intel Performance Libraries provide a large collection of computationally intensive functions, covering application code development areas that extend beyond general-purpose programming. These functions typically fall into the following categories:

- **Special function processing for desktop-oriented, PC applications and handheld computing devices**

These include a broad range of software functions that perform key operations within applications for desktop PCs and handheld computing devices.

- **Large dataset problem processing and high-performance computing**

These functions provide support for upwardly scalable hardware including uniprocessors, hyper-threaded CPUs, computer systems utilizing shared-memory multiprocessing (SMP) architecture, as well as distributed multiprocessing (MP) platforms and computer clusters.

Advantages of Using Intel Performance Libraries

By using Intel Performance Libraries, developers can realize the following advantages:

Performance

Intel offers a full suite of library functions that can help developers easily create the fastest software possible on Intel architecture. The libraries provide high-quality code that is optimized to take advantage of the specific performance features built into each of the Intel processor models.

Compatibility

Intel is committed to smoothing application program migration through the evolving generations of CPUs and computer architecture. By adhering to the abstract model defined by the Intel Performance Libraries' single API, developers are freed to focus on their own code development without worrying about library compatibility issues.

Support

Intel provides timely, effective technical support to developers using Intel Performance Libraries. Intel's expert support staff is dedicated to helping you receive the maximum possible performance and productivity benefits their tools can provide.

Intel Premier Support is a customer support service that provides a forum for discussing any questions or issues developers may have regarding Intel products. This support service allows developers to submit questions and issues privately to Intel's expert software engineering support staff. Developers can download software updates, find solutions to known problems, or review the status of previously submitted issues.

Functional Overview

Intel Performance Libraries are available for both the Windows* and Linux* operating environments. The code within the libraries includes specific performance optimizations that take advantage of hardware performance enhancements included within all Intel processors including the latest Intel® Pentium® 4, Intel® Xeon™, Intel® Itanium® 2, and Intel® PCA processors.

Intel® Math Kernel Library (Intel® MKL)

Intel MKL contains highly optimized functions for math, engineering, scientific, and financial applications that demand high performance on Intel platforms. These include software functions at the center of applications that manipulate large datasets including optimized, low-level vector, vector-matrix, and matrix-matrix primitives (BLAS Levels 1, 2, and 3); LAPACK solvers and optimizers; Discrete Fourier Transform (DFT); vector transcendental (vector math library/VML), and vector statistical functions (VSL).

The library code has been optimized to harvest the maximum performance from both processor and system capabilities. Code that could benefit the most from cache management techniques has been carefully laid out to obtain the best performance. The code has also been organized to facilitate the best utilization of the multiple integer and floating-point pipelines built into Intel processors. To achieve the best performance scaling across multiprocessor-equipped systems, Intel MKL employs threaded code, using industry-standard OpenMP* constructs.

In addition to the industry-standard LAPACK and BLAS libraries that support the most-common linear algebra-oriented applications, Intel MKL offers extra capabilities. One of these capabilities is the Discrete Fourier Transform (DFT) library that operates on multidimensional data. The DFT functions are able to

process mixed-radix transforms and can operate on datasets that are not limited to powers of two sizes. Intel MKL also offers the Vector Math Library (VML), which supports common mathematical functions including trigonometric, exponential, logarithmic, root, and power functions. VML also supports most of the functions contained within the standard C (libm) math library. Unlike the standard libm functions that only support scalar variables, VML functions are optimized for efficient manipulation of vector data using processor-specific performance features. The Vector Statistical Library (VSL) offers hand-optimized code implementing a collection of random number generators that directly operate on vectors. These random number generators produce output conforming to many commonly used probability distribution models. This capability is particularly important for improving the performance of applications that rely on Monte Carlo simulation methods. Such applications are common in both scientific and financial modeling applications.

Table 1 provides a breakdown of the function categories and application areas for each of the functional groups found within Intel MKL.

For each of the Intel MKL functions, Intel provides a sample program that illustrates how to call the function from an application program.

Intel® Integrated Performance Primitives (Intel® IPP)

Intel IPP libraries encompass a broad range of software functions that perform key operations within desktop-oriented, PC applications and hand-held devices. The libraries are organized according to application-specific domains including:

- Audio, JPEG image, speech, and video encoding/decoding
- Computer vision
- Cryptography
- Image, signal and string processing
- Speech recognition and small-scale vector math

For each application domain, the Intel IPP library provides key functions that implement the most-significant algorithms and performance-sensitive key operations. The code provided can be thought of as *building blocks* that can be arranged and glued together to produce the desired application-specific functionality. To provide the developer with flexibility and control over the layout of their code, Intel broke down key segments of commonly needed algorithms to produce a set of specialized, low-level functions referred to as *primitives*. Primitives are functions that perform a single, specialized operation and require minimum overhead code to manage their operation. The advantage of using primitives is that developers can still maintain maximum flexibility and control over such aspects of their code as data structure management, error control, precision control, and interface requirements. Wherever applicable, Intel IPP libraries provide variants of common functions that are also optimized for handling specific data operand size and precision.

Table 2 provides a breakdown of the function categories and application areas for each of the functional groups found within the Intel IPP software library.

Table 3 lists the available code samples provided with Intel IPP¹. These samples demonstrate how Intel IPP functions are pieced together with general program code to produce a working application.

Table 1. Function Categories and Application Areas Associated with Intel® Math Kernel Library (Intel® MKL) Functional Groups

Function Group	Function Categories	Typical Application Areas
BLAS	<p>Vector operations: dot product, scalar-vector multiply and vector add, rotations, swap, copy, and minimum and maximum element functions. Called the Level 1 BLAS.</p> <hr/> <p>Vector-matrix operations: matrix vector multiply and rank updates for various types of matrices (Hermitian and symmetric). Called the Level 2 BLAS.</p> <hr/> <p>Matrix-matrix operations: multiplication and rank updates for various data types and matrix types. Called Level 3 BLAS.</p>	<p>Science, math, engineering, manufacturing, finance</p>
LAPACK	<p>Matrix factorization, dense solver for systems of linear equations, matrix inversion</p> <hr/> <p>Least squares and eigenvalue problems</p>	<p>Solving systems of linear equations (science, math, engineering, manufacturing, finance)</p>
DFT	<p>Multidimensional, mixed-radix, Discrete Fourier Transform</p>	<p>Signal processing (oil and gas exploration, medical imaging)</p>
VML	<p>High- and low-precision versions of component-wise vector transcendental functions (for example, compute sine or exponential of each element in a vector thereby creating a new vector)</p>	<p>Financial software (for example, mortgage pricing), integral evaluation, and many scientific codes with extensive use of transcendental functions</p>
VSL	<p>Various algorithm implementations of basic random number generators (BRNGs) for both discrete and continuous cases. Functions for distributing random numbers generated by BRNGs according to various statistical distributions.</p>	<p>Monte Carlo simulations (finance, medical, chemistry, physics)</p>

Table 2. Function Categories and Application Areas Associated with Intel® Integrated Performance Primitives (Intel® IPP) Functional Groups

Function Group	Function Categories	Typical Application Areas
Audio Codecs	Low-level encode and decode functions for digital audio formats including MP3, AAC, AC3, TwinVQ	Audio recording or playback, audio support in digital video applications, network audio multicasting, games
Computer Vision	<p>Image pyramid (resample & blur)</p> <hr/> <p>Filters - Laplace, Sobel, Scharr, Erode, Dilate</p> <hr/> <p>Motion gradient, flood fill, Canny edge detection</p> <hr/> <p>Snakes, optical flow</p>	Image recognition, bio-metric identification, and security; automated quality control for manufacturing; remote operation of equipment and gesture interpretation; automated sorting of materials or other objects; tracking or counting of objects
Cryptography	More than 200 functions supporting hashing and cryptography functions for algorithms including DES, TDES, Rijndael, SHA1, and DSA	Secure communication, electronic signature, secure data storage, transaction security, ID verification, smart card interfaces, copy protection
Image Processing	<p>Data manipulation, arithmetic, and logical functions; copy and conversions</p> <hr/> <p>Geometric operations (scale, rotate, warp)</p> <hr/> <p>Color conversion, alpha composite, gamma correction</p> <hr/> <p>General and specific filters</p> <hr/> <p>FFT, DCT, wavelet transform</p>	2D image analysis; noise reduction; optical correction; image scaling; image combination; format and color-space conversion; special effects for photo or video processing; general object compression/decompression; capture, storage, and rendering of photographic or video image data; analysis and manipulation of non-photographic images such as ultrasound, CT, MRI, radar, satellite data, etc.; games involving sophisticated visual content and effects
JPEG, JPEG-2000	Functions specific to JPEG and JPEG-2000 standards	Capture and playback of photo, video, or other images; games
Matrix Math	Matrix-matrix, vector-matrix, and matrix-array-matrix operations, optimized for 6x6 and smaller matrices	Modeling and simulation, animated object rendering, general 3D transforms, determination of object behavior in 3D space, games

Function Group	Function Categories	Typical Application Areas
Signal Processing	Data manipulation, windowing, conversion, arithmetic, and logical operations <hr/> Filters and Transforms <hr/> Statistics, Audio pre- and post-processing <hr/> Signal generation	Analysis of audio and other 1D signals; recording, enhancement and playback of audio or non-audio signals; noise reduction and echo cancellation; filtering, equalization and emphasis; simulation of environment or acoustics; reverb, echo and special audio effects; games involving sophisticated audio content and effects
Speech Codecs	Low-level encode and decode functions for digital voice formats including G.722.1, G.723, G.726, G.728, G.729, GSM/FR, GSM/AMR	Voice-over-IP (VoIP), voice annotation on text or graphics, voice recording and playback, phone/PDA interoperability, multi-player or massively-multi-player games
Speech Recognition	Feature processing, model evaluation, model estimation, model adaptation and vector quantization functions	Voice-operated command and control, hands-free operation of equipment, applications for assisting disabled persons, bio-metric identification and security, automated dictation or language translation assistance
String Processing	Low-level string and character processing functions	XML parsing; text database management, search and retrieval; document indexing; text processing
Vector Math	Add, subtract, multiply, divide, exponential, logarithm, plus vectorized math.h functions	Applications that require vector or array data processing
Video Codecs	Low-level encode and/or decode functions for digital video formats including MPEG-1, MPEG-2, MPEG-4, H.263, H.264 (MPEG-4 part 10)	Video recording, playback, editing, transcoding; network video multicasting; security/remote premises monitoring; games; teleconferencing

Table 3. Intel® Integrated Performance Primitives (Intel® IPP) Code Samples

Function Group	Sample
Audio Coding	ACC decoder Aurora encoder - decoder MP3 floating-point-based encoder MP3 floating-point-based encoder - threaded MP3 floating-point-based decoder MP3 integer-based decoder TwinVQ encode TwinVQ decode AC3 decode
Video Coding	H.263 decoder MPEG-1 decoder MPEG-2 encoder MPEG-2 encoder - threaded MPEG-2 decoder MPEG-2 decoder - threaded MPEG-4 encoder MPEG-4 decoder MPEG-4 decoder - threaded
Development Samples	C#*-based Intel IPP usage C++ - based Intel IPP usage Delphi*-based Intel IPP usage Visual Basic*-based usage Kylix*-based Intel IPP usage
Image Processing Coding	IJG encoder - decoder Image processing Image tiling JPEG viewer JPEG encoder - decoder JPEG2000 encoder - decoder
Speech Coding	G.723.1 speech encoder - decoder G.729 speech encoder - decoder G.722.1 speech encoder - decoder G.726 speech encoder - decoder G.728 speech encoder - decoder GSM/FR speech encoder - decoder
Speech Recognition	Gaussian mixture Advanced Aurora
Cryptography	Crypto usage and C# samples

Performance Highlights of Intel MKL and Intel IPP

The following provides a summary of the performance achieved by using Intel MKL and Intel IPP libraries.

Intel MKL Performance

The hallmark of the Intel MKL is that library functions are implemented in variant forms—each form specifically coded to take advantage of the performance features built into the architecture of a specific Intel processor. In addition, the code is designed to yield the best system performance, taking into account memory bandwidth and caching behavior. The code is written to be thread safe and is threaded to provide upwardly scalable performance in environments that support multiple processors.

Developers of application software that utilizes linear algebra typically evaluate the performance of the Double-precision General Matrix-Matrix Multiply (DGEMM) function to assess the overall performance level of the floating-point abilities of processors. DGEMM is central to many linear algebra operations, and can be used as a benchmark for assessing the performance level that a library, such as Intel MKL, can deliver. DGEMM is a BLAS Level 3 function that is called by most linear algebra solver and optimizer functions, such as those included in the standard LAPACK library. When performing a typical core calculation task, such as matrix inversion using the common, *LU Decomposition* algorithm, DGEMM executes at its core a very large number of basic, two-operand, multiply/add floating-point operations. The number of rows and columns that the matrix consists of governs the total number of iterations that must be performed to transform the matrix. For example, a square matrix consisting of **N** rows and **N** columns requires **N**³ iterations. Accordingly, a matrix consisting of 1,000 rows and 1,000 columns requires 1,000³ (one billion) iterations.

DGEMM was designed to be a general-purpose, “Victorinox* Swiss army knife*” version of square matrix multiplication utility functions. Consequently, DGEMM takes 13 call arguments that govern the various aspects of its operations. DGEMM assumes that the matrices to be manipulated are specified in a general layout form as follows:

$$\mathbf{C} := \alpha * \mathbf{A} * \mathbf{B} + \beta * \mathbf{C}$$

Where alpha and beta are scalars, and **A**, **B**, and **C** are matrices with dimensions specified as follows:

A is an **M** by **K** matrix

B is a **K** by **N** matrix

C, the output matrix, is an **M** by **N** matrix

Three of the DGEMM arguments control the dimensions of the matrices to be manipulated by the function as follows:

M Number of rows in matrix **A**

K Number of columns in matrix **A**

N Number of columns in matrices **B** and **C**

Because the size of the manipulated arrays dictates the number of iterations that DGEMM executes internally, it is common to express dataset size for benchmark purposes in terms of **M**, **K**, and **N**.

Figure 1 shows typical DGEMM performance for single and dual Intel® Xeon™ processor configurations. The GFLOPS² values shown in the graph represent the number of floating-point multiply/add instructions that DGEMM performs internally for a given range of **M** and **K**, and for a fixed **N** value set to 2000. The average performance values, measured in MFLOPS, for any given pair of **M** and **K** values are color-coded according to the legend on the right.

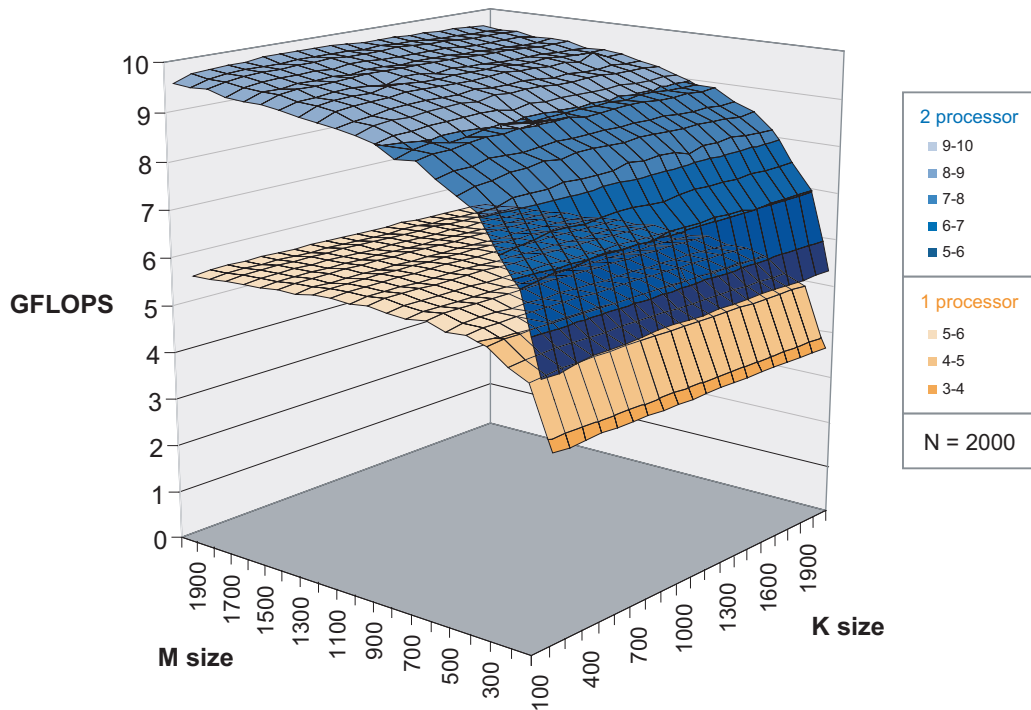


Figure 1. DGEMM Performance Increase When Comparing Single and Dual Intel® Xeon™ Processor Configurations

The performance graph in Figure 1 highlights the performance scalability of the Intel MKL DGEMM implementation. By taking full advantage of processor-specific performance features and full threading capabilities, performance can be scaled upward based upon the number of available processors. On multiprocessor systems, Intel MKL takes advantage of all available processors to accelerate the performance of computational tasks through concurrent code execution. The data used to produce the graph was measured using an Intel Xeon processor-based platform equipped with 1 MB of L3 cache and 4 GB RAM and operating at a clock speed of 3.2 GHz. The operating system used was Windows* 2003 Enterprise Edition.

Version 6.1 of Intel MKL included performance improvements for the latest Itanium 2 microprocessors. Figure 2 illustrates typical performance levels for DGEMM over a wide range of dataset sizes. The graph shows the significant performance gain, due

to threading, that can be achieved with Intel MKL functions for applications deployed on multiprocessor platforms. The results were measured on a four-way 900 MHz Itanium 2-based server, with 3 MB of L3 cache and 16 GB of memory running Red Hat Linux* Advanced Workstation 2.1 Kernel 2.4.18-e.12.

The DFT, VML, and VSL libraries are similarly optimized to take advantage of CPU-specific performance features and, where applicable, employ threading and thread-safe code. Refer to performance benchmark information related to these libraries at www.intel.com/software/products/mkl

Intel IPP Performance

The Intel IPP library provides cross-platform support through a single API. The large set of utility functions that Intel IPP provides supports a broad range of Intel processor families. For each of the supported target processors, Intel IPP provides API-conformant, variant code that is specifically written to yield the best

system performance for the target processor, taking into account memory bandwidth and caching behavior of the target environment. Wherever applicable, the code was written with thread safety in mind. These libraries also offer a variant form that takes advantage of execution thread concurrency for those functions that can realize performance increases due to threading.

Intel IPP functions yield significantly better performance than equivalent compiler-generated C code. By replacing sections of C code with equivalent Intel IPP functions, application programs are able to complete key tasks in significantly shorter timeframes. This ability to complete critical tasks in a timely manner is particularly important for event-driven and real-time-constrained operations. Such operations typically take place within device drivers, audio and video processing and motion picture rendering.

Figure 3 provides a graphic representation of the performance improvements that can be achieved by replacing C compiler-generated code with equivalent Intel IPP functions for a broad range of application domains. The graph represents performance gains achievable from Intel IPP 3.0.

For each of the specified application domains, the vertical bars indicate the average performance improvement factor that can be achieved by utilizing Intel IPP functions. The gray bars indicate performance gains obtained from the Intel IPP code optimized for the Intel Pentium III processor as compared to compiled C versions. The blue bars indicate performance gain obtained from the Intel IPP code optimized for the Intel Pentium 4 processor as compared to C compiler-generated code. These results represent averaged values derived from a test suite that exercised all the functions included within the Intel IPP libraries. This test suite consists of over 61,000 test vectors, and is available to customers who purchase the Intel IPP library product. These test vectors utilize a variety of datasets, and they manipulate all the data types that are appropriate for each of the functions within the Intel IPP library.

For more specific performance benchmark information related to Intel IPP libraries, refer to www.intel.com/software/products/ipp

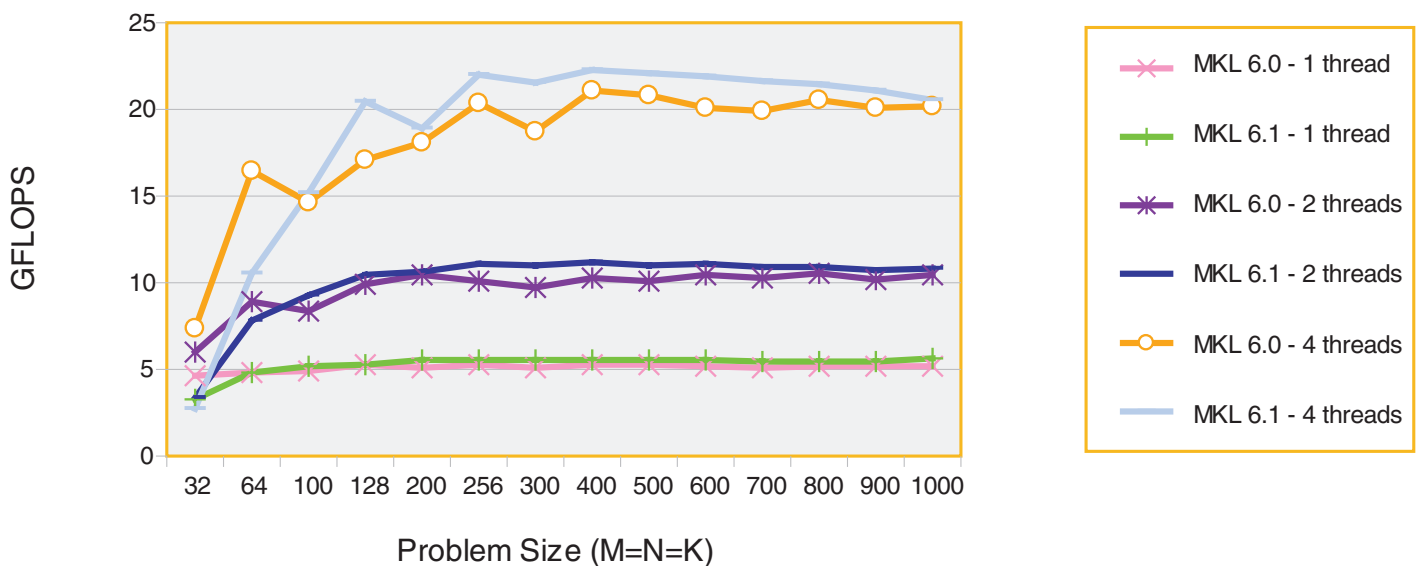


Figure 2. Comparing Intel® Math Kernel Library 6.1 to Intel Math Kernel Library 6.0 on DGEMM Performance on the Intel® Itanium® 2 Processor³

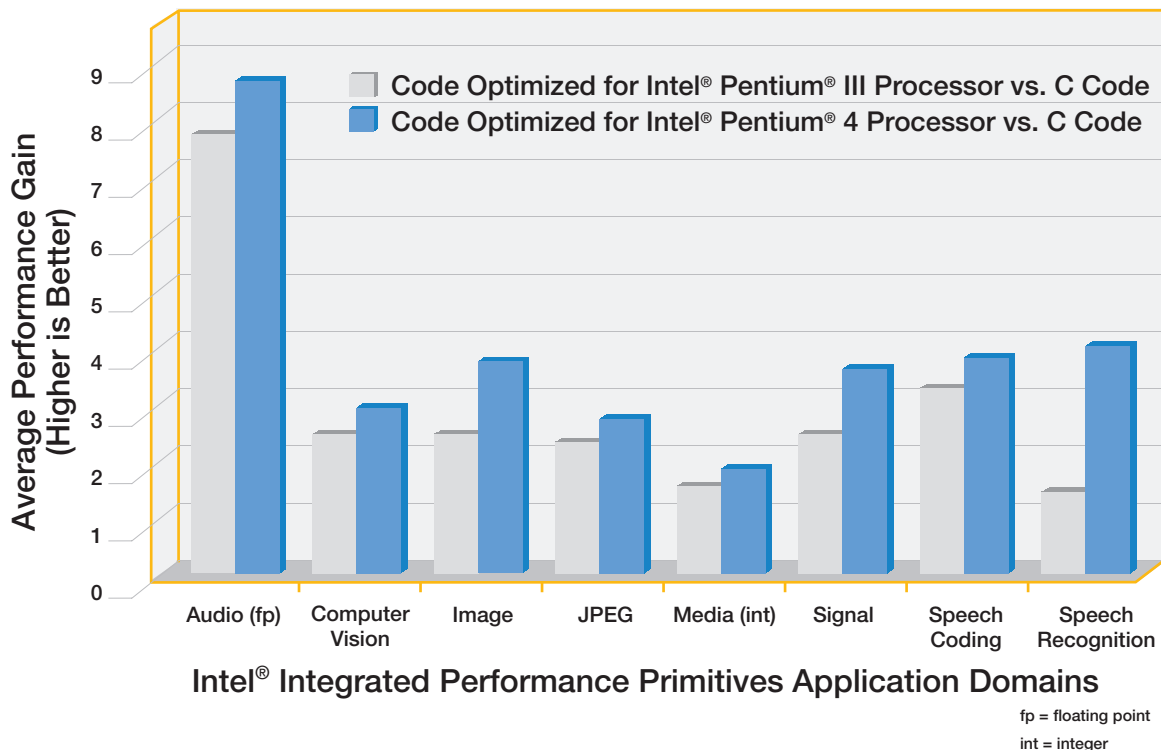


Figure 3. Average Intel® Integrated Performance Primitives (Intel® IPP) Performance Gains over Compiled C Code

Code Examples

The following code examples illustrate how replacing standard C library functions with Intel MKL and Intel IPP functions can bring performance to significantly higher levels.

Intel MKL Code Example

Let's take a look at an example of Intel MKL library use in an application. Following is C code that generates a large number of random numbers as you might expect to see in a Monte Carlo simulation.

```
double dbX, dbY;
int i;
srand( 7777777 );

for ( i = 0; i < 200048640; i++ )
{
    dbX = ((double)rand()) / ((double)(RAND_MAX)) ;
    dbY = ((double)rand()) / ((double)(RAND_MAX)) ;
    /* test functions */
}
```

This code can be replaced by calls to the Vector Statistical Library (VSL) functionality in Intel MKL. In the following code, the 31-bit multiplicative, congruential generator (MCG31) is initialized using the `vslNewStream` function. The stream data structure, `VSLStreamStatePtr`, is defined in `mkl.h` and holds the state of the basic random number generator making multiple calls possible. Multiple streams can be set up to enable multi-threaded applications. Once the stream is initialized, it is used to obtain uniformly distributed random numbers via the `vdRngUniform` function. These numbers are generated in vectors of length 16,384 for each of the coordinates reducing the outer loop count to 12,210. The stream should be deleted when the program no longer makes use of it using the `vslDeleteStream` function.

```

#include "mkl.h"

VSLStreamStatePtr stream;
static double dbBuf[2*16384];
double dbX, dbY;
int i, j;

/* Initialize MCG31 random number stream with seed
7777777*/
vslNewStream( &stream, VSL_BRNG_MCG31, 7777777 );

for ( i = 0; i < 12210; i++ )
{
    /* Fill dbBuf array with 2*16384 random numbers*/
    vdRngUniform( 0, stream, 2*16384, dbBuf, 0.0, 1.0 );

    for ( j = 0; j < 16384; j++ )
    {
        dbX = dbBuf[2*j];
        dbY = dbBuf[2*j+1];
        /* test functions */
    }
}

/* Delete the stream */
vslDeleteStream( &stream );

```

This use of VSL improved performance for one test function that was run on a two-processor Itanium 2-based system by more than 14 times on a single processor and by more than 24 times using both processors.³

Intel IPP Code Example

Following is a C code excerpt from a modeling application.

```

float *flArgSin; /* Array of aligned sine vectors. */
float *pflTmp; /* Temporary pointer to the sine array.*/
int iWvMeshSize; /* Angle harmonic and wave harmonic. */
int thrd; /* Number of threads. */

/* Calculate the sine of each value in the aligned vector
array. */
pflTmp = flArgSin[thrd].algPtr;
for ( t = 0; t < 4*iWvMeshSize; t++)
{
    pflTmp[t] = (float) sin(pflTmp[t]);
}

```

The for loop in the above code is replaced by an Intel IPP function call as follows:

```

/* IPP vector sine function with 11 correctly rounded
bits of significand. */
IppsSin_32f_All( flArgSin[thrd].algPtr,
flArgSin[thrd].algPtr, 4*iWvMeshSize );

```

This replacement of a for loop by one Intel IPP function call improved performance by 400 percent⁴, illustrating how minor efforts can provide major gains in application performance.

Identifying Opportunities for Performance Boosting

Considering the performance gains that Intel Performance Libraries offer, it is beneficial for development teams to consider using the libraries from the outset, starting with the planning stages of a new project. However, performance improvements can be realized by utilizing Intel Performance Libraries even in conjunction with code that is already in place. Typically, performance-related issues are discovered towards the end of a product development cycle. The majority of performance-related issues are exposed during product integration and testing. At such a critical juncture, the development team is likely to face the greatest pressure to find a quick fix, risk significant delays, or worse yet, face premature termination of the project.

Often the root cause of performance-related issues is unclear to the development team. Frequently, under the pressure of finding a quick fix, developers may resort to a “cut and try” approach that can be both time consuming and yield less than satisfactory results. A more disciplined alternative, however, generally yields better results. Such an approach includes one or more iterations of a process consisting of the steps in Table 4.

In this process, Steps 1 through 5 are repeated until a satisfactory solution is reached. Alternatively, if no satisfactory solution is reached, the solution may be abandoned and a new alternative solution may be pursued.

Table 4. The Optimization Process

Steps in Optimization	
1. Gather Information	Measure, collect, and characterize code behavior
2. Analyze your findings	Identify patterns and assess the meaning of collected performance data
3. Define a solution	Devise alternatives and select the most promising candidate solution
4. Implement the solution	Develop code that implements the selected, alternative solution
5. Test the effectiveness of the solution	Deploy the new solution and track its behavior

The key to determining an effective solution to a performance problem primarily depends on the developer's ability to measure, collect, and characterize the behavior of the code. Proper performance measurement tools are necessary to locate quality data, thereby facilitating a successful investigation of problem causes. The Intel VTune Performance Analyzer takes advantage of the special execution profiling facilities that are built into each Intel processor. By using such a tool, there is no need for any guesswork. Performance bottlenecks can be located with ease, and performance hotspots quickly become apparent.

Performance hotspots are usually located within code segments that are frequently executed in the course of program operations. Whenever a hotspot is observed while the program is executing a performance-sensitive operation, analysis of that hotspot may provide a useful lead to understanding the causes of unsatisfactory performance. Because run-time libraries contain frequently used code, there is a high likelihood that a given hotspot will be located within one or more library functions. Consequently, after analyzing findings in Step 2 of the process described above, and looking for alternative solutions in Step 3, the developer may consider using specific functions provided within the Intel Performance Libraries. In addition to the

performance advantages that the Intel library functions provide, developers can save a considerable amount of time in testing the new solution during Step 5, since code provided by Intel has been thoroughly tested prior to deployment.

Run-time Dispatching

Intel accommodates the need for transparent portability across CPU lines without sacrificing performance by incorporating a mechanism known as *run-time dispatching* within the Intel Performance Libraries. Run-time dispatching is a mechanism for selecting the library function variants that yield the best performance. By identifying the CPU during application program initialization, this library dispatch mechanism sets up pointers to the appropriate dynamic link libraries (DLLs) on Windows platforms or shared object archives on Linux platforms that are able to take advantage of the performance features of the particular CPU. Once these pointers are set, they remain unchanged for the duration of the application program lifetime. Calls made to DLLs or shared object archives through these pointers reference functions within the run-time libraries. Beyond the initial setup phase, this dispatching mechanism carries no additional overhead.

Conclusion

Whether designing applications using Intel MKL or Intel IPP, developers can significantly reduce their risk, expenditures and time to market by using the pre-built and tested Intel libraries. Development teams can rely on cross-platform support by adopting the Intel libraries API, freeing them to focus on their own code development without worrying about performance issues or long-term maintenance burden. They can count on Intel's commitment to maintaining and upgrading all library functions to take advantage of specific performance features built into all current and future Intel processors.

In DGEMM performance benchmarks that used Intel MKL libraries, the performance increase for the dual-processor configuration highlighted the effectiveness of the threaded code used throughout Intel MKL function implementations. The data showed that, on multiprocessor systems, Intel MKL takes advantage of all available processors to accelerate the performance of computational tasks through concurrent code execution.

Significant performance gains were realized for Intel IPP library code optimized for Pentium 4 processors when compared to C compiler generated code, with averages ranging from more than 150 percent to more than 800 percent improvement, depending on the function domains tested.

Because Intel Performance Libraries provide high-quality code that is optimized to take advantage of the specific performance features built into each of the Intel processor models, developers can expect similar performance gains for all functions included within the libraries.

¹ Code samples listed reflect those provided as of Intel IPP 4.0.

² One GFLOP is a unit of performance measurement representing one billion floating-point operations per second.

³ Two-way, 900 MHz Itanium 2-based platform with 1.5 MB L3 cache, 2 GB of memory running Microsoft Windows Server 2003 - Enterprise Edition.

⁴ 2.0 GHz Intel® Pentium® 4-based platform with 256 MB of memory running Windows® XP.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the appropriate performance of Intel products as measured by those tests. Any difference in system 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, go to www.intel.com/software/products

References

For additional information, refer to the following:

General Information

Intel Performance Libraries Web site:
www.intel.com/software/products/perflib

Intel® Software College:
www.intel.com/software/college

Intel Premier Support:
www.intel.com/software/products/support

Performance Benchmark White Papers

Intel IPP Performance Information:
www.intel.com/software/products/ipp

Intel MKL Performance Information:
www.intel.com/software/products/mkl

Choosing the Best Linkage Model for Your Intel® IPP-based Application for Intel® Pentium® and Intel® Itanium® Architectures White Paper:
www.intel.com/software/products/ipp

Technical User Notes

Intel Math Kernel Library:
www.intel.com/software/products/mkl/docs/manuals.htm

User Manuals

Intel MKL:
www.intel.com/software/products/mkl/docs/manuals.htm

Intel IPP:
www.intel.com/software/products/ipp/docs/manuals.htm



Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95052-8119
USA

For product and purchase information visit:
www.intel.com/software/products

Intel, the Intel logo, Itanium, Pentium, Intel Xeon, Intel Centrino, Intel XScale and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.