Intel® System Studio
Signal Processing
What you will learn from this slide deck

• How to use various components of Intel® System Studio to target signal processing workloads for System & Application code running Linux* & Tizen™

• Note: there are no specifics for each development environment in this slide deck. The concepts should apply to both.

• Please see subsequent slide decks for in-depth technical training on other components
Intel® System Studio 2014

Reliable C/C++ system software development tool suite that allows you to optimize your Intel Architecture based intelligent systems and embedded devices

✓ **Most Insight**
  - Processor and SoC wide analysis tools to optimize for low power and maximum performance
  - Powerful debug tools for efficient system and SoC wide defect analysis

✓ **Best Performance**
  - High-performance C++ Compiler
  - Key software building blocks that help performance optimize data and **signal processing**, and math operations

✓ **Broad Platform Coverage**
  - Best IA platform coverage from Atom™ to Xeon®
  - Key embedded Linux* OS and real-time extension support
# Intel System Studio

## Signal Processing

<table>
<thead>
<tr>
<th>Module</th>
<th>Features</th>
</tr>
</thead>
</table>
| **Eclipse**     | • Set up your development environment and configure Intel System Studio software components for your cross build environment.  
                 | • Windows* Host and Linux* Host Support with Eclipse* Integration                                                                                                                                     |
| **App & JTAG Debug** | • Low level system debug for system stabilization  
                         | • In-Depth view of Vector Data and SIMD registers  
                         | • Find single and multithreaded code run time programming errors such as memory leaks, stack overflows, unexpected exceptions, and other algorithmic problems. |
| **VTune™**      | • Get detailed SoC wide analysis and tune your code for optimal performance & power consumption.  
                         | • Get detailed analysis of your code running on Intel processors and identify performance hotspots, analyze concurrency, and tune threaded and non-threaded code. |
| **C++ Compiler & Perf. Libraries** | • Supercharge your code for Intel architecture with Intel® C++ Compiler  
                         | • Boost your performance through SSSE3, SSE, AVX, AVX2 microarchitecture features in the high performance libraries  
                         | • Take advantage of advanced vectorization |
| **Inspector**   | • Powerful and easy to use memory and threading error checking tool designed for serial and parallel code. Facilitates application reliability, finds intermittent and non-deterministic errors. |
Unlock the Signal Processor Inside Intel® Architecture!

Accelerate your development and quickly achieve the best performance with Intel® C++ Compiler, Intel® IPP and Intel® MKL

- Hand-tuned code with Intel® Advanced Vector Extensions (Intel® AVX) & Intel® Advanced Vector Extensions 2 (Intel® AVX2)
- Functions for common embedded algorithms
- Thread-safe functions
- No blocking calls
- Multiple OS support
- Minimal OS overhead
- User-replaceable memory allocation mechanism

Intel® Integrated Performance Primitives (Intel® IPP)
Intel® Math Kernel Library (Intel® MKL)
Intel® Advanced Vector Extensions (Intel® AVX)
Intel® Advanced Vector Extensions 2 (Intel® AVX2)
The Basics

IPP

- Single precision emphasis, focus on breadth of type support rather than depth
- IPP has a function for every data type, integer friendly
- Smaller Data Sets
- Atom, Server, and Core Processors
- Vectorized Optimizations
- 32 and 64 bit

MKL

- Heavy emphasis on double precision
- Larger Data Sets
- Vectorized and Threaded Optimizations
- Host: Ubuntu, Fedora
- Target: WindRiver 5, Yocto 1.x, Fedora 14, Custom Linux
- Motivation is to support anything that “makes sense” for Core

Target OSs are with or without virtualization and with or without RT scheduler.
Intel® Math Kernel Library (MKL)

- Highly optimized threaded math routines
- Optimizations using Intel® AVX and the new Intel® Advanced Vector Extensions 2 (Intel® AVX2) including the new FMA3 instructions
- Embedded systems in Communications, Medical, Industrial, MAG, and other segments
- Use Intel® MKL on your Linux* targets
- Use Intel® MKL with Intel® C++ Compiler and GCC

Unlock processor performance with Intel® MKL
Intel® MKL is industry’s leading math library *

<table>
<thead>
<tr>
<th>Linear Algebra</th>
<th>Fast Fourier Transforms</th>
<th>Vector Math</th>
<th>Vector Random Number Generators</th>
<th>Summary Statistics</th>
<th>Data Fitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>• BLAS</td>
<td>• Multidimensional</td>
<td>• Trigonometric</td>
<td>• Congruential</td>
<td>• Kurtosis</td>
<td>• Splines</td>
</tr>
<tr>
<td>• LAPACK</td>
<td>(up to 7D)</td>
<td>• Hyperbolic</td>
<td>• Recursive</td>
<td>• Variation</td>
<td>• Interpolation</td>
</tr>
<tr>
<td>• Sparse solvers</td>
<td>• FFTW interfaces</td>
<td>• Exponential, Logarithmic</td>
<td>• Wichmann-Hill</td>
<td>• Quantiles, order statistics</td>
<td>• Cell search</td>
</tr>
<tr>
<td>• ScaLAPACK</td>
<td>• Cluster FFT</td>
<td>• Power / Root</td>
<td>• Mersenne Twister</td>
<td>• Min/max</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rounding</td>
<td>• Sobol</td>
<td>• Variance-covariance</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Neiderreiter</td>
<td>• ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Non-deterministic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary Statistics

• Kurtosis
• Variation coefficient
• Quantiles, order statistics
• Min/max
• Variance-covariance
• ...

* 2013 Evans Data N. American developer survey
What Is Intel Math Kernel Library?

Performance, Performance, Performance!

Industry’s leading math library (* 2013 Evans Data N. American developer survey)

Addresses:

- Linear equation Solvers
- Eigenvector/Eigenvalue solvers
- PDEs, signal processing, seismic, solid-state physics (FFTs)
- General scientific, financial [vector transcendental functions (VML) and vector random number generators (VSL)]
- Sparse Solvers (PARDISO, DSS and ISS)
- Data fitting functions, Spline construction, interpolation, extrapolation, cell search

Tuned for Intel processors – current and the next generation

Vectorized, threaded, and distributed multiprocessor aware
Third-party Tools Powered by Intel MKL

IMSL* Fortran Numerical Libraries (Rogue Wave)

NAG* Libraries

MATLAB* (MathWorks)

GNU Octave*

NumPy* / SciPy*

PETSc* (Portable Extensible Toolkit for Scientific Computation)

WRF* (Weather Research & Forecasting run-time environment)

The HPCC* benchmark

And more ...
## Where Does the Parallelism Come From?

<table>
<thead>
<tr>
<th>Domain</th>
<th>SIMD</th>
<th>Open MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS 1, 2, 3</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>FFTs</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>LAPACK (dense LA solvers)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(relies on BLAS 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScALAPACK (cluster dense LA solvers)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(hybrid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARDISO (sparse solver)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(relies on BLAS 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VML/VSL</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Cluster FFT</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Why is Intel MKL faster?

Optimization done for maximum speed.

Resource limited optimization – exhaust one or more resource of system:

- **CPU**: Vectorization, Register use, FP units.
- **Cache**: Keep data in cache as long as possible; deal with cache interleaving.
- **TLBs**: Maximally use data on each page.
- **Memory bandwidth**: Minimally access memory.
- **Computer**: Use all the processor cores available using threading.
- **System**: Use all the nodes available. Optimized for Intel® MPI.
DGEMM Performance (Intel MKL vs. ATLAS*)

Performance improves using Intel® Math Kernel Library versus ATLAS*

DGEMM on Intel® Server Processor

Intel® MKL offers significant performance boost over ATLAS*
Performance scales as number of CPU cores increases
Performance up to 93% of CPU Gflop/s peak!

Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 10.3.9 ATLAS 3.8.4; Hardware: Intel® Xeon® Processor E5-2690, 2 Eight-Core CPUs (20Mb L3 Cache, 2.90GHz), 32GB of RAM; Operating System: RHEL 6.6 x86_64; Benchmark Source: Intel Corporation.
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 the performance of Intel products, refer to www.Intel.com/performance/resources/benchmark_limitations.htm.
* Other brands and names are the property of their respective owners.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804
Intel MKL Components

BLAS
- Basic vector-vector/matrix-vector/matrix-matrix computation routines.

Sparse BLAS
- BLAS for sparse vectors/matrices

LAPACK (Linear algebra package)
- Solvers and Eigenvalue solvers. Many hundreds of routines total!
- C interface to LAPACK

ScaLAPACK
- Computational, driver and auxiliary routines for distributed-memory architectures

DFTs (General FFTs)
- Mixed radix, multi-dimensional transforms, FFTW interfaces

Sparse Solvers (PARDISO, DSS and ISS)
- Direct and Iterative sparse solvers for symmetric, structurally symmetric or non-symmetric, positive definite, indefinite or Hermitian sparse linear system of equations
- Out-Of-Core (OOC) version for huge problem sizes
Intel MKL Components (cont’d)

VML (Vector Math Library)
- Set of vectorized transcendental functions, most of libm functions, but faster

VSL (Vector Statistical Library)
- Set of vectorized random number generators
- SSL (Summary Statistical Library) : Computationally intensive core/building blocks for statistical analysis

DFL (Data Fitting Library)
- Linea, quadratic, cubic, step-wise const, and user-defined Splines
- Cell search with configuration parameters for optimal performance
- User defined interpolation & extrapolation

PDEs (Partial Differential Equations)
- Trigonometric transform and Poisson solvers.

Optimization Solvers
- Solvers for nonlinear least square problems with/without constraints.

Support Functions
# Intel MKL Environment

<table>
<thead>
<tr>
<th>Domain</th>
<th>C/C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS</td>
<td>Via CBLAS</td>
</tr>
<tr>
<td>Sparse BLAS Level 1</td>
<td>Via CBLAS</td>
</tr>
<tr>
<td>Sparse BLAS level 1&amp;2</td>
<td>x</td>
</tr>
<tr>
<td>LAPACK</td>
<td>x</td>
</tr>
<tr>
<td>PARDISO</td>
<td>x</td>
</tr>
<tr>
<td>DSS &amp; ISS</td>
<td>x</td>
</tr>
<tr>
<td>VML/VSL</td>
<td>x</td>
</tr>
<tr>
<td>FFT</td>
<td>x</td>
</tr>
<tr>
<td>PDEs</td>
<td>x</td>
</tr>
<tr>
<td>Optimization (TR) Solvers</td>
<td>x</td>
</tr>
<tr>
<td>SSL</td>
<td>x</td>
</tr>
</tbody>
</table>

**Language Support**

**Linux***

- Compiler: Intel, Gnu
- Libraries: .a, .so

64 bit static and dynamic libraries
**Intel® MKL: Fast Fourier Transform (FFT)**

- Single and double precision complex and real transforms.
  - 1, 2, 3 and multidimensional transforms
- Multithreaded and thread-safe.
- Transform sizes: 2-powers, mixed radix, prime sizes
  - Transforms provide for efficient use of memory and meet the needs of many physical problems. *Any* size transform can be specified, but not all transform sizes run equally fast.
- User-specified scaling supported.
- Multiple transforms on single call.
- Strides
  - Allow FFT of a part of image, padding for better performance, transform combined with transposition, facilitates development of mixed-language applications.
- **Integrated FFTW interfaces**
  - Source code of FFTW3 and FFTW2 wrappers in C/C++ and Fortran are provided.
  - FFTW3 wrappers are also built into the library.
  - Not all FFTW features are supported.
Fast Fourier Transform Performance

Threading Optimizations

2D FFT Performance Boost by using Intel® Math Kernel Library versus FFTW®

- Intel® MKL provides higher performance than FFTW®
- Performance scales as number of CPU cores increases

Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.0, FFTW® 3.3.2; Hardware: Intel® Xeon® Processor E5-2690, 2 Eight-Core CPUs (20MB LLC, 2.9GHz), 32GB of RAM; Operating System: RHEL 5 GA x86_64; Benchmark: Single precision complex 2-dimension FFT, data may have been padded to avoid each thrashing, source: Intel Corporation.

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 the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

* Other brands and names are the property of their respective owners.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804
Intel® MKL FFT Interface (DFTI)
(see also http://portal.acm.org/citation.cfm?id=1114271)

Overview

• DFTI_DESCRIPTOR_HANDLE — pointer to an opaque structure
• The 5-stage usage model: Create, Configure_{opt}, Commit, Compute, Free
• Numerous parameters for Configure_{opt}

Example (configuring this $F_M \otimes I_N \otimes F_K$):

• DftiCreateDescriptor(&hand, DFTI_SINGLE, DFTI_COMPLEX, 2, &{M,K} );
• DftiSetValue(hand, DFTI_INPUT_STRIDES, &{0,NK,1} ); /* row-major */
• DftiSetValue(hand, DFTI_NUMBER_OF_TRANSFORMS, N );
• DftiSetValue(hand, DFTI_INPUT_DISTANCE, K );
• DftiCommitDescriptor(hand);
• loop (call this repeatedly to compute arbitrary number of FFTs)
  - DftiComputeForward(hand, X, Y);
  - DftiComputeBackward(hand, Y, X); /* caution: Y uses input strides */
• DftiFreeDescriptor(&hand)
DFTI Functions

- DftiCreateDescriptor: Create default computation plan
- DftiSetValue: Adjust configuration of the plan
- DftiCommitDescriptor: Commit the plan
- DftiComputeForward: Forward/Backward Transforms
- DftiComputeBackward
- DftiFreeDescriptor: Release plan’s memory
DFTI Example

- Complex-to-complex 1D transform, double precision, not in place.

/* Create a descriptor */
Status = DftiCreateDescriptor( &Desc_Handle, DFTI_DOUBLE, DFTI_COMPLEX, 1, n );

/* Set placement of result: DFTI_NOT_INPLACE */
Status = DftiSetValue(Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE);

/* Commit the descriptor */
Status = DftiCommitDescriptor( Desc_Handle );

/* Compute a forward transform */
Status = DftiComputeForward(Desc_Handle, x_in, x_out);
DFTI Example (continue)

/* Set Scale number for backward transform */
Scale = 1.0/(double)n;
Status = DftiSetValue( Desc_Handle, DFTI_BACKWARD_SCALE, Scale );

/* Commit the change made to the descriptor */
Status = DftiCommitDescriptor( Desc_Handle );

/* Compute a backward transform */
Status = DftiComputeBackward( Desc_Handle, x_out, x_in );

/* Free the descriptor */
Status = DftiFreeDescriptor( &Desc_Handle );
**FFTW API** (see http://www.fftw.org)

Overview

- `fftw_plan` — pointer to an opaque structure, created by planners.
- Many planners
  - problem types: dft, r2c, c2r, and r2r (limited support in MKL).
  - data layout: complex vs split-complex, embedded data.
  - simple and guru interfaces.
- Wisdom management.

Example (computing $F_M \otimes I_N \otimes F_K$):

- `plan *fwd = fftw_plan_guru_dft(2,&{{K,1,1},{M,NK,NK}},1,&{{N,K,K}},X,Y,FFTW_FORWARD,FFTW_PATIENT)`
- `plan *bwd = fftw_plan_guru_dft(...,Y,X,FFTW_BACKWARD,FFTW_PATIENT)`
- loop
  - `fftw_execute(fwd);`
  - `fftw_execute(bwd);`
- `fftw_destroy_plan(fwd);`
- `fftw_destroy_plan(bwd);`

Compute FFT as many times as you like, with data contained in arrays X and Y. Alternatively, use new-array execute functions, like

`fftw_execute_dft( fwd, another_X, another_Y )`
FFTW Usage Model

Setup

• plan p = plan_dft(rank, dims, X, Y, sign, flags)
• plan_dft_1d(n,...), ..._2d(nx, ny, ...), ..._3d(nx, ny, nz, ...)
• FFTW_ESTIMATE | _MEASURE | _PATIENT | _EXHAUSTIVE
• In-place or out-of-place
• Alignment
• Measurement (unless FFTW_ESTIMATE)

Execution

• execute_dft(p, X, Y),
  execute_split_dft(p, Xr, Xi, Yr, Yi)

Cleanup

• destroy_plan(p)
MKL FFTW Interface via Wrappers

Note: The FFTW3 wrappers are built as part of library. Users don’t need to build by themselves.

/* Create & Commit a descriptor for 1D forward transform */
plan = fftw_plan_dft_1d( n, x_in, x_out,
                        FFTW_FORWARD,FFTW_ESTIMATE );

/* Compute forward DFT*/
fftw_execute( plan );

/* Set Scale number for Backward transform */
Scale = 1.0/(double)n;
/* Create & Commit a descriptor for 1D backward transform */
Desc_Handle = fftw_plan_dft_1d( n, x_out, x_in, 
                              FFTW_BACKWARD, FTW_ESTIMATE );

/* Compute backward DFT */
fftw_execute(Desc_Handle);

/* Free Dfti descriptor */
fftw_destroy_plan(Desc_Handle);

/* Result scaling */
scaling_d(x_in, Scale, n);
Summary of FFT Support

• Intel MKL FFTs support 1, 2, 3 and multidimensional transforms.

• Mixed Radix Support.

• Multithreaded for 1, 2, 3 and multidimensional transforms.

• Scales very well on multi-core systems (single node) and across many nodes in clusters.
New optimizations

Optimizations using the new Intel® Advanced Vector Extensions 2 (AVX2) including the new FMA3 instructions—the following parts have optimizations:

• BLAS
• FFTs
• Vector math functions
• Data fitting functions
• Random number generators
• Summary statistics functions
**Intel® Integrated Performance Primitives (IPP)**

**Optimized for Performance & Power Efficiency**
- Highly optimized using SSSE3, SSE, and AVX instruction sets
- Performance beyond what an optimized compiler produces alone

**Intel Engineered & Future Proofed to Save You Time**
- Ready-to-use
- Fully optimized for current and past processors
- Save development, debug, and maintenance time
- Code once now, receive future optimizations later

**Wide range of Cross Platform & OS Functionality**
- Thousands of highly optimized functions
- Supports various Linux* targets
- Supports Intel® Atom™, Core™, and Xeon® processors

---

**Signal Processing (1D)**
- **Filters**
  - FFT
  - FIR
  - Threshold
  - Convolution
  - Median
- **Statistics**
  - Mean
  - StdDev
  - NormDiff
  - Sum
  - MinMax

**Image & Frame Processing (2D)**
- **Transforms**
  - FFT
  - Resize
  - Rotate
  - Mirror
  - Warp/Shear
- **Filters**
  - Convolution
  - Morphology
  - Threshold
  - Histogram
- **Computer Vision**
  - Canny
  - Optical Flow
  - Segmentation
  - Haar Classifiers
  - Hough Transform
- **Color Conversion**
  - RGB/BGR
  - YUV/YCbCr
  - 420, 422, 444
- **Statistics**
  - Mean
  - StdDev
  - NormDiff
  - Sum
  - MinMax

Performance building blocks to make your embedded device faster
Intel IPP - overview

Application Source Code

Intel IPP Usage Code Samples
- Sample video/audio/speech codecs
- Image processing and JPEG
- Signal processing
- Data compression
- .NET and Java integration

Intel IPP Library C/C++ API
- Cryptography
- Image processing
- Image color conversion
- JPEG / JPEG2000
- Computer Vision
- Video coding
- Audio coding
- Data Compression
- Data Integrity
- Signal processing
- Matrix mathematics
- Vector mathematics
- String processing
- Speech coding
- Data Integrity

Intel IPP Processor-Optimized Binaries
- Intel® Core™ i7 Processors
- Intel® Atom™ Processors
- Intel® Core™ 2 Duo and Core™ Extreme Processors
- Intel® Core™ Duo and Core™ Solo Processors
- Intel® Pentium® D Dual-Core Processors
- Intel® Xeon® 64-bit Dual-Core Processors
- Intel® Xeon® DP and MP Processors

Free Code Samples
Cross-platform API
Processor-Optimized Implementation

Rapid Application Development
Compatibility and Code Re-Use
Outstanding Performance

API calls
Static/Dynamic Link
Why are the functions fast?

- Intel® IPP functions exploit the instruction set architecture by
  - processing multiple data elements in parallel
    - Streaming SIMD Extensions like SSE4
  - processing data in larger chunks with each instruction

- Intel® IPP functions exploit the processor micro architecture by
  - pre-fetching data and avoiding cache blocking
  - resolving data and trace cache misses
  - avoiding branch mispredictions

- Intel® IPP functions use all execution resources available in the CPUs
  - Multi-Core Technology
Intel® IPP function naming convention and usage

Function names
- are easy to understand
- directly indicate the purpose of the function via distinct elements
- each element has a fixed number of predefined values

```
ippiCopy_8u_C1MR
```

<table>
<thead>
<tr>
<th>Name Elements</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
<td>Indicates the functional data type in 1D, 2D and Matrix</td>
<td>ipps,ippi,ippm</td>
</tr>
<tr>
<td>Base Name</td>
<td>Abbreviation for the core operation</td>
<td>Add, FFTFwd, LuDecomp</td>
</tr>
<tr>
<td>Data Type</td>
<td>Describes bit depth and sign</td>
<td>8u, 32f, 64f</td>
</tr>
<tr>
<td>Execution mode</td>
<td>Indicates data layout and scaling</td>
<td>ISfs, C1R, P</td>
</tr>
</tbody>
</table>

Each function performs a particular operation on a known type of data in a specific mode.
Intel® IPP libraries components

Header files, Dynamic and Static Libraries are sorted by Domains.

The dispatcher libraries and SSE-based optimized libraries are included in both Dynamic and Static Libraries

For example:

<table>
<thead>
<tr>
<th>Domains</th>
<th>Header File</th>
<th>Dynamic Linking</th>
<th>Static Linking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video Coding (vc)</td>
<td>ippvc.h</td>
<td>ippvc.lib, ippvc.dll</td>
<td>lppvc_l.lib, lppvc_t.lib</td>
</tr>
</tbody>
</table>
Function implementation

Intel IPP uses codes optimized for various central processing units (CPUs). Dispatching refers to detection of your CPU and selecting the corresponding Intel IPP binary. For example, ippiv8-7.0.dll in the \redist\ia32\ipp directory, reflects the imaging processing libraries optimized for the Intel(R) Core(TM) 2 Duo processors.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Identifier</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32 Intel® Architecture</td>
<td>px</td>
<td>C-optimized for all IA-32 processors</td>
</tr>
<tr>
<td></td>
<td>v8</td>
<td>Optimized for processors with Intel® Supplemental Streaming SIMD Extensions 3 (Intel SSSE3)</td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>Optimized for processors with Intel® Streaming SIMD Extensions 4.1 (Intel SSE4.1)</td>
</tr>
<tr>
<td></td>
<td>s8</td>
<td>Optimized for the Intel® Atom™ processor</td>
</tr>
<tr>
<td>Intel® 64 (Intel® EM64T) architecture</td>
<td>mx</td>
<td>C-optimized for processors with Intel® 64 instructions set architecture</td>
</tr>
<tr>
<td></td>
<td>u8</td>
<td>Optimized for 64-bit applications on processors with Intel® Supplemental Streaming SIMD Extensions 3 (Intel SSE3)</td>
</tr>
<tr>
<td></td>
<td>y8</td>
<td>Optimized for 64-bit applications on processors with Intel® Streaming SIMD Extensions 4.1 (Intel SSE4.1)</td>
</tr>
<tr>
<td></td>
<td>n8</td>
<td>Optimized for the Intel® Atom™ processor</td>
</tr>
<tr>
<td></td>
<td>e9</td>
<td>Optimized for processors that support Intel® Advanced Vector Extensions instruction set</td>
</tr>
</tbody>
</table>

Intel ® IPP gets updated with these libs to match the latest CPU features.
Threading control flexibility in Intel® IPP

Intel® IPP are thread-safe. It supports threading above it:

- Intel® IPP threading functions are self-contained, which do not necessarily require application level threading to use OpenMP*
- Intel® IPP threading can be disabled or fine-tuned by applications

In a case that application needs fine-grained threading control

- Call the function `ippSetNumThreads` with argument 1
- Use static library to avoid OpenMP dependency
- Use completely single thread ideal for kernel development
Threading in Intel® IPP functions

Many computational intensive functions are threaded

Many (~2480) of Intel IPP functions are threaded.

• Where it improves performance

Usage model:

• Intel IPP threading Control
  - ippSetNumThreads
  - ippGetNumThreads

<table>
<thead>
<tr>
<th>Domains</th>
<th>Threaded Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippi</td>
<td>1346</td>
</tr>
<tr>
<td>ippr</td>
<td>11</td>
</tr>
<tr>
<td>ipm</td>
<td>527</td>
</tr>
<tr>
<td>ipps</td>
<td>586</td>
</tr>
</tbody>
</table>

*Intel IPP functions are threaded when it maximizes performance*
Threading in application

1 of 2 cores

application thread

application thread 1

application does threading

call Intel IPP

call Intel IPP

thread-safe functions

2 of 2 cores

application thread 2

call Intel IPP

call Intel IPP

application thread continues

application thread continues
Threading inside Intel® IPP

1 of 2 cores

application thread

call Intel IPP

Intel IPP does threading

Intel IPP internal thread

Intel IPP returns to application

application thread continues

2 of 2 cores
Intel® IPP - linking options

- Dynamic linking using the run-time dynamic link libraries
- Static linking with dispatching by using emerged and merged static libraries
- Static linking without automatic dispatching using merged static libraries
- Dynamically building your own, custom, dynamic link library

What are the main differences?

- Code size (application executable or application installation package)
- Distribution method
- Processor coverage
- Application executes in kernel mode?
Dynamic linking

Dynamic linking is the simplest method and the most commonly used. It takes full advantage of the dynamic dispatching mechanism in the dynamic link libraries (DLLs)

To dynamically link with Intel ® IPP, follow these steps:

1. Include `ipp.h` in your application. This header includes the header files for all Intel IPP functional domains.
2. Use the normal Intel IPP function names when calling the functions.
3. Link corresponding domain import libraries. For example, if you use the function `ippsCopy_8u`, link to `ipps.lib`.
4. Make sure that the run-time libraries are on the executable search path at run time.

Run the ` IPP\bin ippvars.bat` from directory to ensure that the application loads the appropriate processor-specific library.
Static Linking with Dispatching

✓ To use the static linking libraries, you need to link to all required domain libraries ipp*._l.lib, adding ippcore_l.lib and libraries on which domain libraries depend (see next section below). The * denotes the appropriate function domain.

✓ If you want to use the Intel IPP functions threaded with the OpenMP*, you need to link to the threaded versions of the libraries ipp*_t.lib, ippcore_t.lib, and libiomp5md.lib.

All domain-specific and core libraries are located in the \ipp\lib<arch> directory.
## Static Linking without Dispatching

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Small executable size with support for only one processor type</td>
<td>• The executable is optimized for only one processor type</td>
</tr>
<tr>
<td>- Suitable for kernel-mode/device-driver/ring-0</td>
<td>• Updates to processor-specific optimizations require rebuild and/or relink</td>
</tr>
<tr>
<td>- Suitable for a Web applet or a plug-in requiring very small file download and support for only one processor type</td>
<td></td>
</tr>
<tr>
<td>- Self-contained application executable that does not require the Intel IPP run-time DLLs</td>
<td></td>
</tr>
<tr>
<td>- Smallest footprint for application package</td>
<td></td>
</tr>
<tr>
<td>- Smallest installation package</td>
<td></td>
</tr>
</tbody>
</table>
## Custom Dynamic Linking

<table>
<thead>
<tr>
<th>Benefits</th>
<th>•Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Run-time dispatching of processor-specific optimizations</td>
<td>▪ Application executable requires access to the Intel compiler specific run-time libraries that are delivered with Intel IPP</td>
</tr>
<tr>
<td>▪ Reduced hard-drive footprint compared with a full set of Intel IPP DLLs</td>
<td>▪ Developer resources are needed to create and maintain the custom DLLs</td>
</tr>
<tr>
<td>▪ Smallest installation package to accommodate use of some of the same Intel IPP functions by multiple applications</td>
<td>▪ Integration of new processor-specific optimizations requires rebuilding the custom DLL</td>
</tr>
<tr>
<td></td>
<td>▪ Not appropriate for kernel-mode/device-driver/ring-0 code</td>
</tr>
</tbody>
</table>
## Intel® IPP Supported Linkage Model - quick comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Dynamic Linkage</th>
<th>Static Linkage with Dispatching</th>
<th>Static Linkage without Dispatching</th>
<th>Using Custom DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Updates</td>
<td>Automatic</td>
<td>Recompile &amp; redistribute</td>
<td>Release new processor-specific application</td>
<td>Recompile &amp; redistribute</td>
</tr>
<tr>
<td>Optimization</td>
<td>All processors</td>
<td>All processors</td>
<td>One processor</td>
<td>All processors</td>
</tr>
<tr>
<td>Build</td>
<td>Link to stub static libraries</td>
<td>Link to static libraries and static dispatchers</td>
<td>Link to merged libraries</td>
<td>Build separate DLL</td>
</tr>
<tr>
<td>Calling</td>
<td>Regular names</td>
<td>Regular names</td>
<td>Processor-specific names</td>
<td>Regular names</td>
</tr>
<tr>
<td>Distribution</td>
<td>Distribute linked IPP dll</td>
<td>No extra Distribution</td>
<td>No extra distribution</td>
<td>Distribute custom dll</td>
</tr>
<tr>
<td>Total Binary Size</td>
<td>Large</td>
<td>Small</td>
<td>Smallest</td>
<td>Small</td>
</tr>
<tr>
<td>Executable Size</td>
<td>Smallest</td>
<td>Small</td>
<td>Small</td>
<td>Smallest</td>
</tr>
<tr>
<td>Kernel Mode</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multi Threading Support</td>
<td>Yes</td>
<td>Yes, when linking with threaded static merged libraries</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Intel® IPP provides a lot of flexibility**
Intel® AVX Optimization

Intel® AVX (Intel® Advanced Vector Extensions) is a 256-bit instruction set extension to SSE designed to provide even higher performance for applications that are floating-point intensive.

Benefits of Intel AVX

✓ Support for wider vector data (up to 256-bit).

✓ Efficient instruction encoding scheme that supports 3 and 4 operand instruction syntaxes.

✓ Flexible programming environment, ranging from branch handling to relaxed memory alignment requirements.

✓ New data manipulation and arithmetic compute primitives, including broadcast, permute, fused-multiply-add, etc.
List of functions optimized for Intel AVX

Mainly following domain APIs are directly optimized for Intel AVX

- Signal Processing
- Image Processing
- SPIRAL (GEN) Functions
- Audio Coding
- Speech Coding
- Color Conversion
- Realistic Rendering
- Computer Vision
- Image Compression

Intel AVX Optimization : Performance data

Data compression functions performance data on SNB -
Conclusion

Many Applications

- Take Multiple Levels of efforts to optimize for threading
- Intel® IPP Libraries help you on primitives threading if your data allow
- Intel® IPP Samples demonstrate how you can thread at the algorithm levels

Application Levels

Algorithm Levels
(Data flow level)

Computational Kernel

Common Core Algorithms

Intel® IPP Primitives Libraries

Intel® IPP Samples
Industry Signal Processing Demands...

Greater Computing Performance
- Sustained and reliable performance increases (Moore’s Law)
- New instructions, architecture continues to deliver increasing performance

Shorter Development Times
- One key toolchain for all your system and application development needs
- Software libraries available for multiple uses
- Huge ecosystem of software and hardware vendors

Product Simplification
- Application focus first, then decide on the platform(s) that’s right for you from Atom™, Core® to Xeon® processors
- Flexibility and scalability by running your application on any IA platform
Functions in Signal Processing Applications

Software to help you take advantage of Intel platforms such as SSSE3, SSE, Intel® Advanced Vector Extensions (Intel® AVX) & Intel® Advanced Vector Extensions 2 (Intel® AVX2) in our performance libraries and Intel® C++ Compiler

- Voice recognition/generation
- Echo addition/cancellation
- Pattern recognition
- Noise reduction
- Anti-aliasing
- Compression
- Smoothing
- Morphing
- Correlation
- Filtering

Supercharge Performance, Accelerate Development, Code Quality and Reliability
How does each define Signal Processing?

**IPP**

_Signal Processing Restricts to data domain of a 1-D input signal_
- Covers the following function groups:
  - Support, Initialization, Essential Vector, Filtering, Transform, Data Integrity, Speech Coding, Audio Coding, String, Vector Arithmetic, Data Compression.

**Image and Video processing: input data is 2-D image**
- Covers the following function groups:
  - Support, Support, Data Exchange and Initialization, Image Arithmetic and Logical Ops, Color Conversion, Threshold and Compare, Morphological, Filtering, Linear Transforms, Image Statistics, Geometric Transforms, Wavelet Transforms, Computer Vision, Image Compression, Video Coding

**MKL**

- Wide variety of FFT support
- Emphasis on high performance, large data sets, for multiple dimensions
- Depth, not breadth, FFT-centric
Where is the overlap?

- Preparation of Data similar
  - Vector Math, Summary Statistics, Descriptors, Specification of Structure, Convolution/Correlation
- FFT/DFTs
  - Products diverge heavily with 2D support (both in how it is defined and what is possible with a 2D data set)
- IPP has support for
  - Creation of various digital signals
  - Conversion of digital signals – heavy diverse IPP support
    - E.g. DemodulateFM
  - Thresholding
  - Conversion to different coordinate systems
  - All kinds of transforms
IPP – Diverse Array of Signal Processing Functions

- Windowing Functions – FFT-based spectral analysis
  - Bartlett, Blackman, Hamming, Hann, Kaiser windowing functions
  - Generate window samples and multiply them into existing signal
- Change sampling rates of input signals
- Filtering functions
- More transforms: Hartley, Hilbert, Walsh-Hadamard, Wavelet
- Entire set of application domains covered (50+ pages of RefMan functions for each)
  - Data Integrity
  - Speech Coding
  - Audio Coding
  - Fixed-Accuracy Arithmetic Functions
  - String Functions
  - Data Compression
  - Image Processing
# IPP Data Type Support

Data Types Supported by Intel IPP for Signal Processing

Different function for each data type

<table>
<thead>
<tr>
<th>Type</th>
<th>Usual C Type</th>
<th>Intel IPP Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u</td>
<td>unsigned char</td>
<td>Ipp8u</td>
</tr>
<tr>
<td>8s</td>
<td>signed char</td>
<td>Ipp8s</td>
</tr>
<tr>
<td>16u</td>
<td>unsigned short</td>
<td>Ipp16u</td>
</tr>
<tr>
<td>16s</td>
<td>signed short</td>
<td>Ipp16s</td>
</tr>
<tr>
<td>16sc</td>
<td>complex short</td>
<td>Ipp16sc</td>
</tr>
<tr>
<td>32u</td>
<td>unsigned int</td>
<td>Ipp32u</td>
</tr>
<tr>
<td>32s</td>
<td>signed int</td>
<td>Ipp32s</td>
</tr>
<tr>
<td>32f</td>
<td>float</td>
<td>Ipp32f</td>
</tr>
<tr>
<td>32fc</td>
<td>complex float</td>
<td>Ipp32fc</td>
</tr>
<tr>
<td>64s</td>
<td>__int64 (Windows*) or long long (Linux*)</td>
<td>Ipp64s</td>
</tr>
<tr>
<td>64f</td>
<td>double</td>
<td>Ipp64f</td>
</tr>
<tr>
<td>64fc</td>
<td>complex double</td>
<td>Ipp64fc</td>
</tr>
</tbody>
</table>
Apples to Apples Comparison for FFT

IPP
• Support for different data formats PACK, PERM, CCS
• Conversion functions for each format
• Function for each data format
• Delineates between FFT and DFT
• Appears to be 1-D only
• hint argument suggest using a special algorithm, faster, or more accurate
• The usual specification structure within contained within initialization functions
• FFT for real and complex signal (power of 2)
• DFT for real and complex signals (non power of 2, any size)
• Special set of IPP functions for OOO
• DFT of complex signal - DEPRECATED
• DFT for a given frequency (Goertzel)
• Hartley, Walsh-Hadamard, DCT, Hilbert, Wavelet

MKL
• Configure descriptor – different data formats included within there
• DftiCompute Forward/Backward, simpler function interface adapts according to much larger descriptor than IPP
• Large data set emphasis, higher precision levels
• Higher dimensions
• FFTW interface
Links for More Info
Key Links

Content URLs

**Whitepapers and support articles**


**Release notes**


**Support forum**

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk
are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that
are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and
other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on
microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended
for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for
Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information
regarding the specific instruction sets covered by this notice.

Notice revision #20110804