Intel® Integrated Performance Primitives

intel®
Software

Intel® Integrated Performance Primitives
Spend More Time Differentiating, Less Time Optimizing

Optimized Function Library

Many Functions

- 2D Image processing
- 1D Signal Processing
- Enterprise Data

Goals

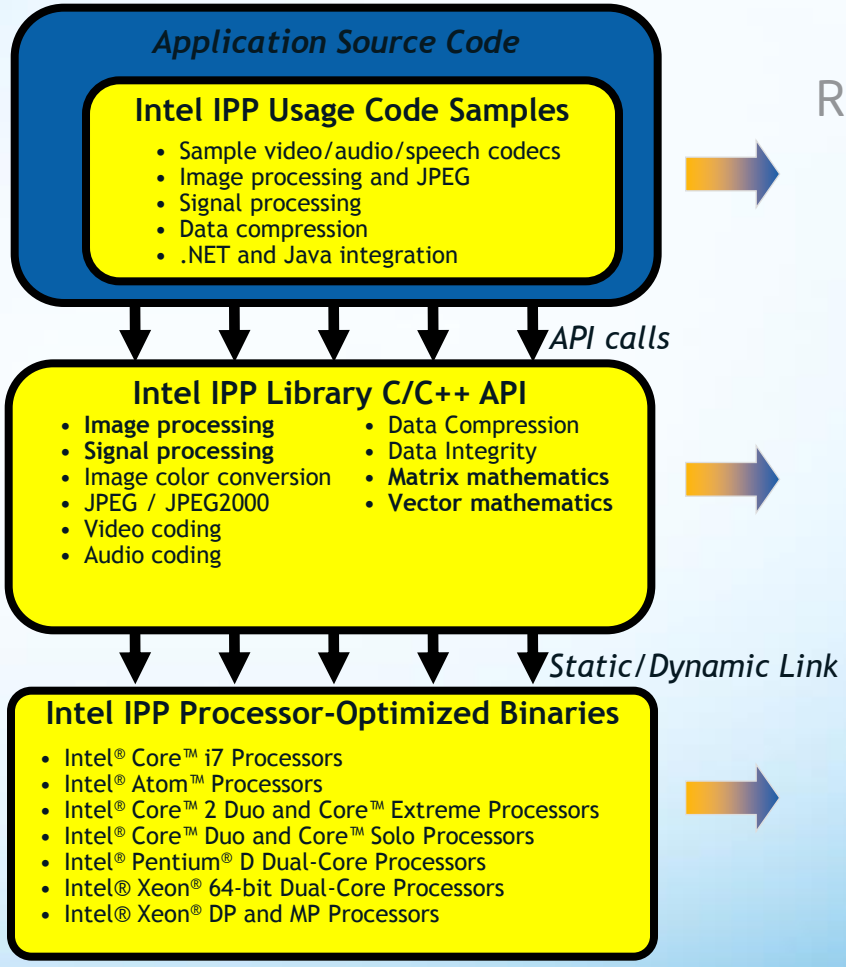
- Raw Performance
- Platform Throughput
- Power Efficiency

Intel IPP - overview

Free Code Samples

Cross-platform API

Processor-Optimized Implementation



Rapid Application Development

Compatibility and Code Re-Use

Outstanding Performance

Intel® Integrated Performance Primitives Image/Signal Domain Details

Image/Frame Processing

2D Operations

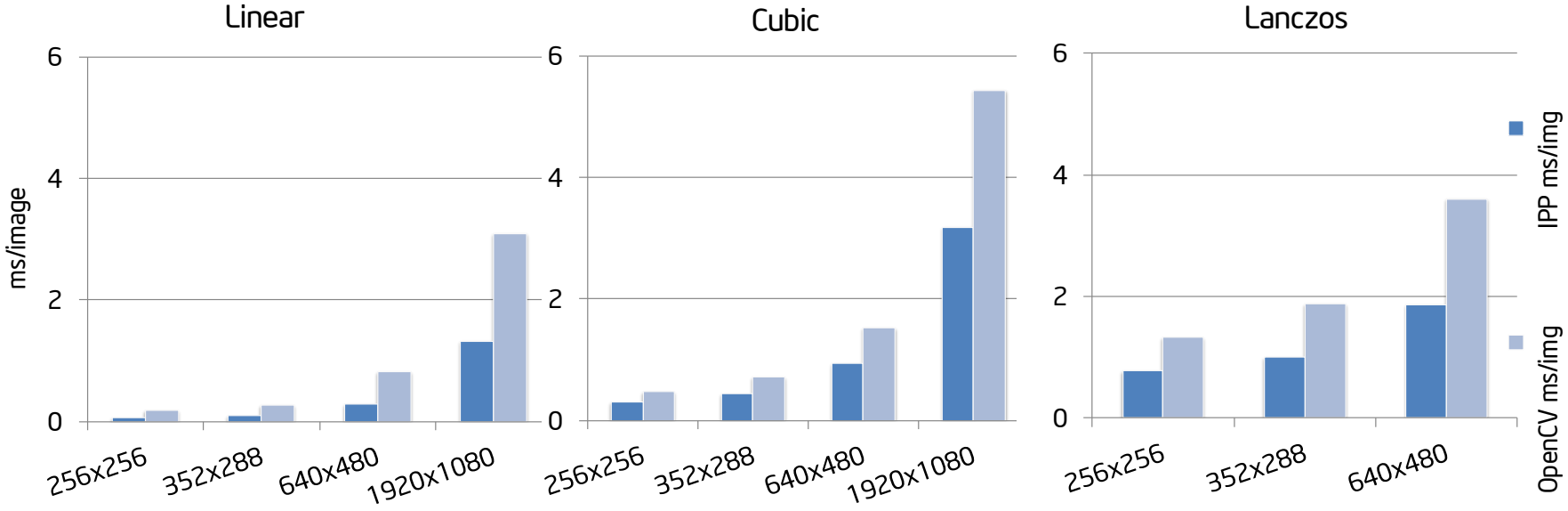
- **Filters/Transforms**
  - Convolution,Morphology,Threshold)
  - Fourier Transforms (FFT, etc.), Resize, Rotate, Mirror,
- **Computer Vision**
  - Canny, Optical Flow, Segmentation, Haar Classifiers
- **Color Conversion**
  - RGB/BGR <-> YUV/YCbCr

Signal Processing

1D Operations

- **Filters/Transforms**
  - Fourier Transforms (FFT, etc.), Finite Impulse Response (FIR), Infinite Impulse Response (IIR), Threshold, Convolution, Median
- **Statistics**
  - Mean, StdDev, NormDiff, Sum, MinMax

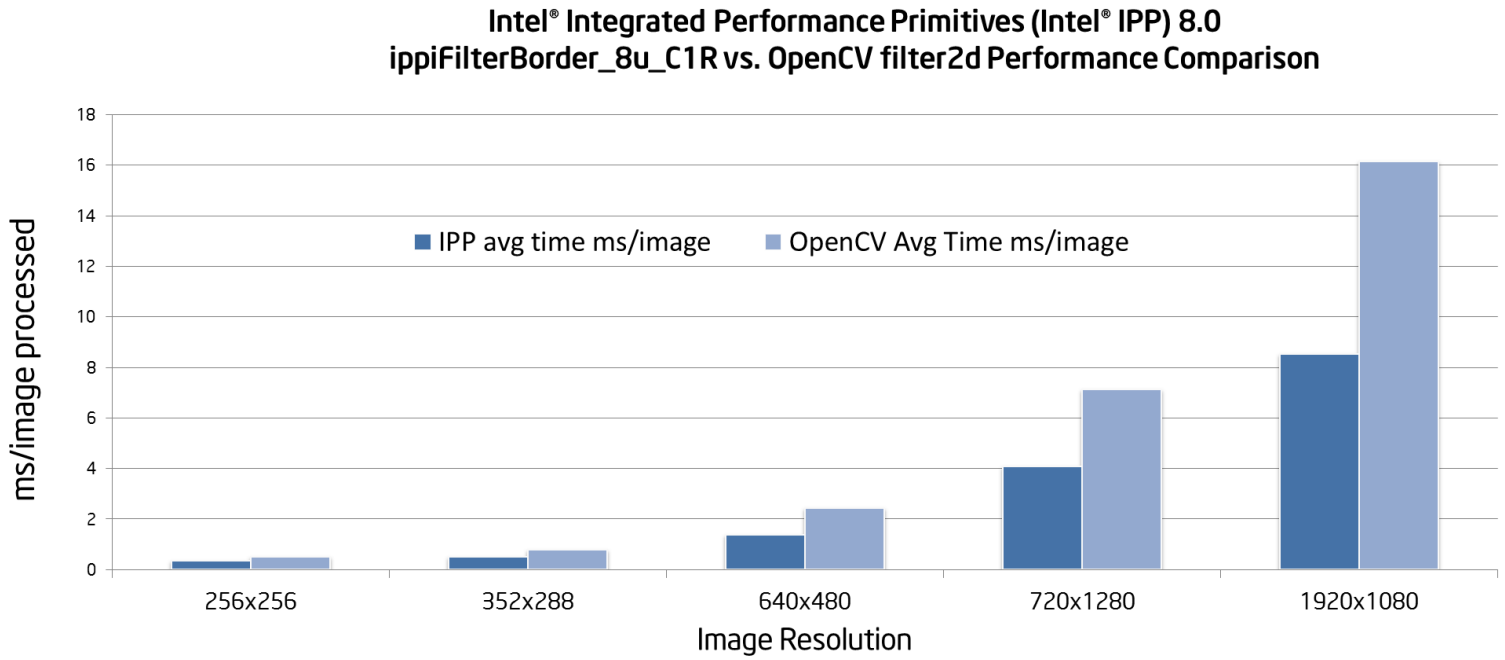
Intel® Integrated Performance Primitives Image Resize Performance vs. OpenCV



System configuration: Intel® IPP 8.0 (build 83); Precompiled OpenCV 2.4.6. Hardware: Intel® Core™ i7 4770s Processor, 3.10 GHz, 256 KB L2 Cache, 8 GB RAM; Operating System: Windows* 8 64 bit; Visual Studio 2012; Single Threaded; Benchmark source: Intel® Corporation; location: Folsom, CA, 02/07/2013; Notes: Linear, Cubic (Catmul-Rom), and Lanczos(3 node) interpolation, 1 channel (greyscale) 1280x720 source image.

Intel® IPP provides faster resize

Intel® Integrated Performance Primitives Pixel Convolution Performance vs. OpenCV



System configuration: Intel® IPP 8.0 (build 83); Precompiled OpenCV 2.4.6. Hardware: Intel® Core™ i7 4770s Processor, 3.10 GHz, 256 KB L2 Cache, 8 GB RAM; Operating System: Windows* 8 64 bit; Visual Studio 2012; Single Threaded; Benchmark source: Intel® Corporation; location: Folsom, CA, 02/07/2013; Notes: , 1 channel (greyscale) image, 7x7 nonseparable kernel.

Intel® IPP provides faster filter operations

Code Example

```
int main(int argc, char **argv)
{
    ippInit();

    Mat kernel(parms.k, parms.k, CV_32F);
    kernel=1.0/(parms.k*parms.k);

    Mat imgSrc(img_h, img_w, CV_8UC1);
    Mat imgDst(img_h, img_w, CV_8UC1);

    int src1step=(int)imgSrc.step1();
    int dst1step=(int)imgSrc.step1();

    //set up IPP Filter
    IppiPoint IPPanchor={parms.k/2,parms.k/2};
    IppiSize kernelsize={parms.k,parms.k};
    IppiSize roiSize={img_w-kernelsize.width,
                      img_h-kernelsize.height};
    int SpecSize, BufferSize;
    ippiFilterBorderGetSize(kernelsize,roiSize,
             ipp8u,ipp32f,1,&SpecSize,&BufferSize);
    IppiFilterBorderSpec *Spec    =
      (IppiFilterBorderSpec*)ippsMalloc_8u(SpecSize);
    Ipp8u *Buffer = (Ipp8u*)ippsMalloc_8u(BufferSize);
    int ROI_offset_elements = (IPPanchor.y * src1step)
      + IPPanchor.x;

    status = ippiFilterBorderInit_32f(
          (Ipp32f*)kernel.data,
          kernelsize, ipp8u, 1, ippRndNear, Spec );

    Ipp8u borderValue = 0;

    ippiFilterBorder_8u_C1R(
      imgSrc.data + ROI_offset_elements, src1step,
      imgDst.data + ROI_offset_elements, dst1step,
      roiSize, ippBorderRepl, &borderValue,
      Spec, Buffer );

    ippFree(Spec);
    ippFree(Buffer);

    return 0;
}
```

Note common pattern: GetSize, Init, Operation

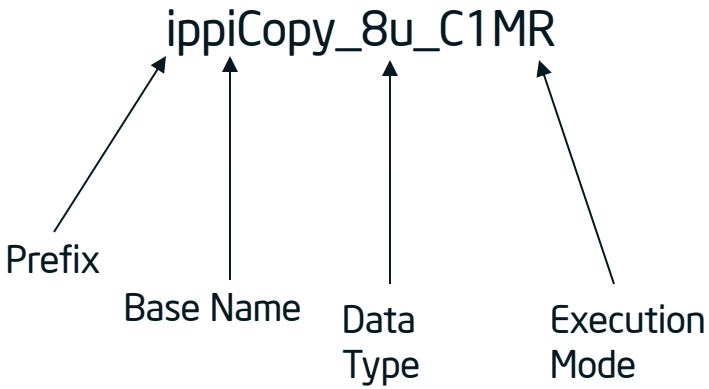
Why are the functions fast?

- 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 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 pre-defined values



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

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:

Domains	Header File	Dynamic Linking	Static Linking
Video Coding (vc)	*ippvc.h*	*ippvc.lib,* *ippvc.dll*	*lppvc_l.lib,* *lppvc_t.lib*

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.

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

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

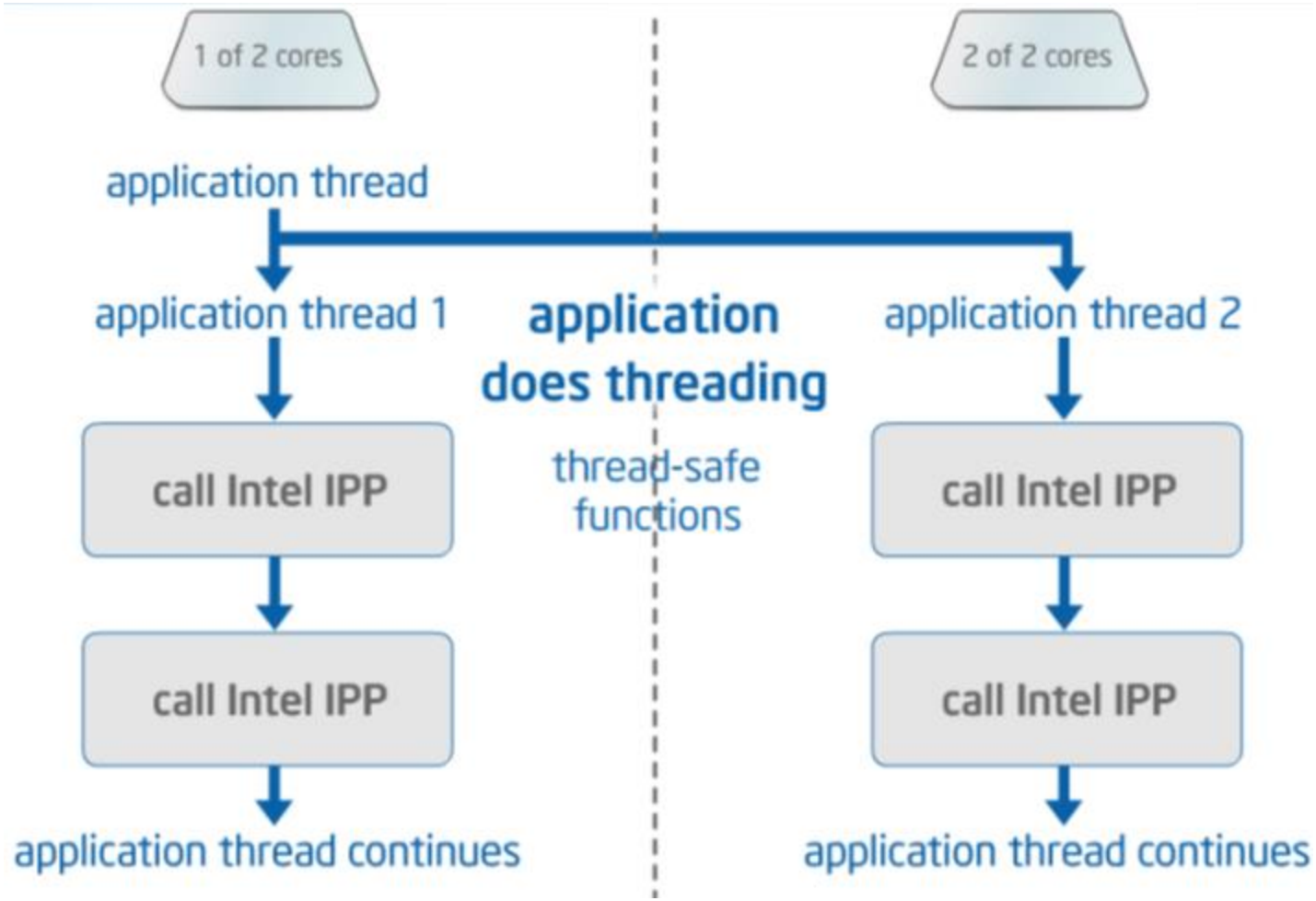
Usage model:

- Intel IPP threading Control
  - *ippSetNumThreads*
  - *ippGetNumThreads*

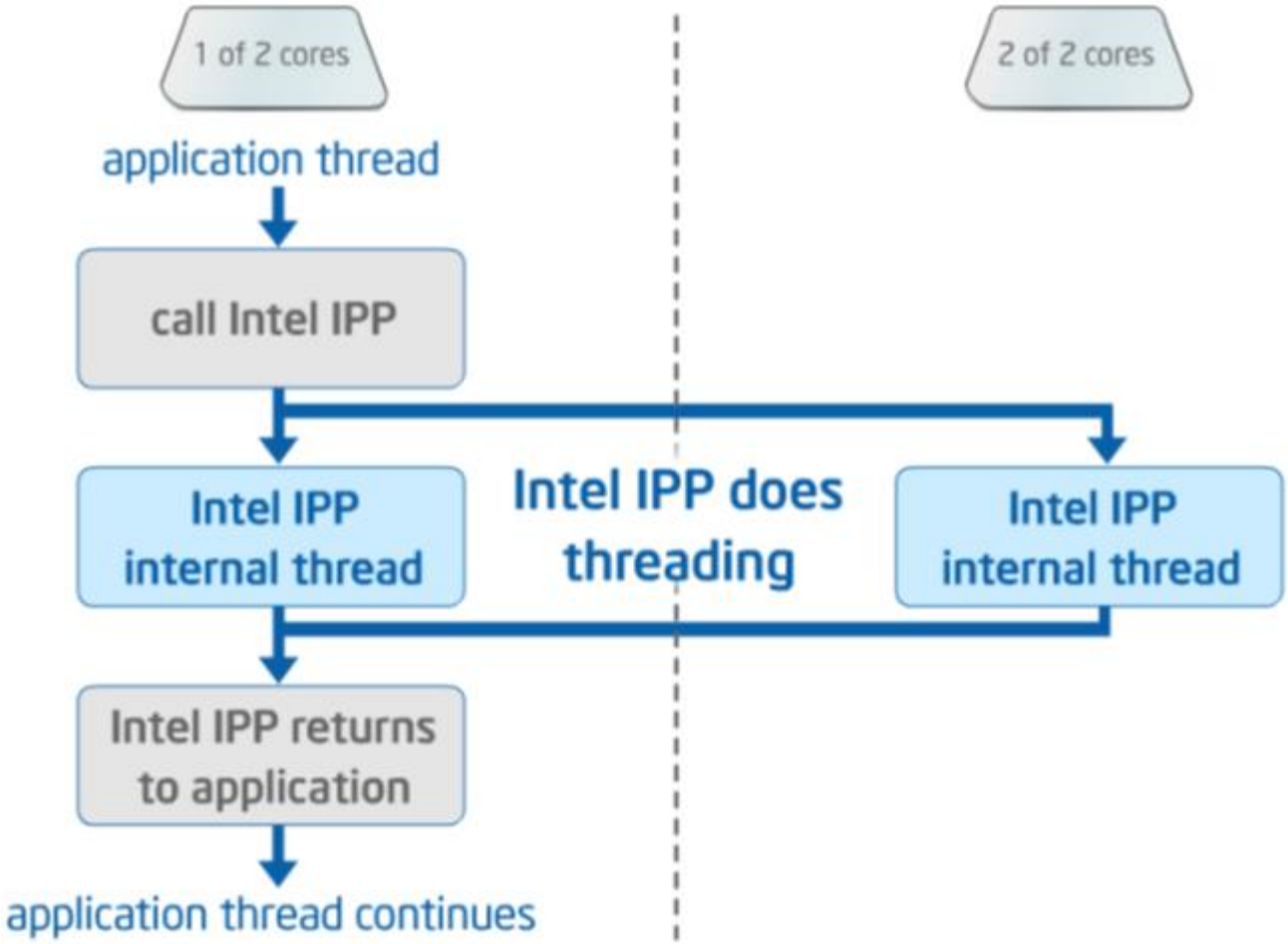
Domains	Threaded Functions
ippi	1346
ippr	11
ipm	527
ipps	586

Intel IPP functions are threaded when it maximizes performance

Threading in application



Threading inside Intel® IPP



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 ippvars.bat from directory **\ipp\bin** 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

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

Custom Dynamic Linking

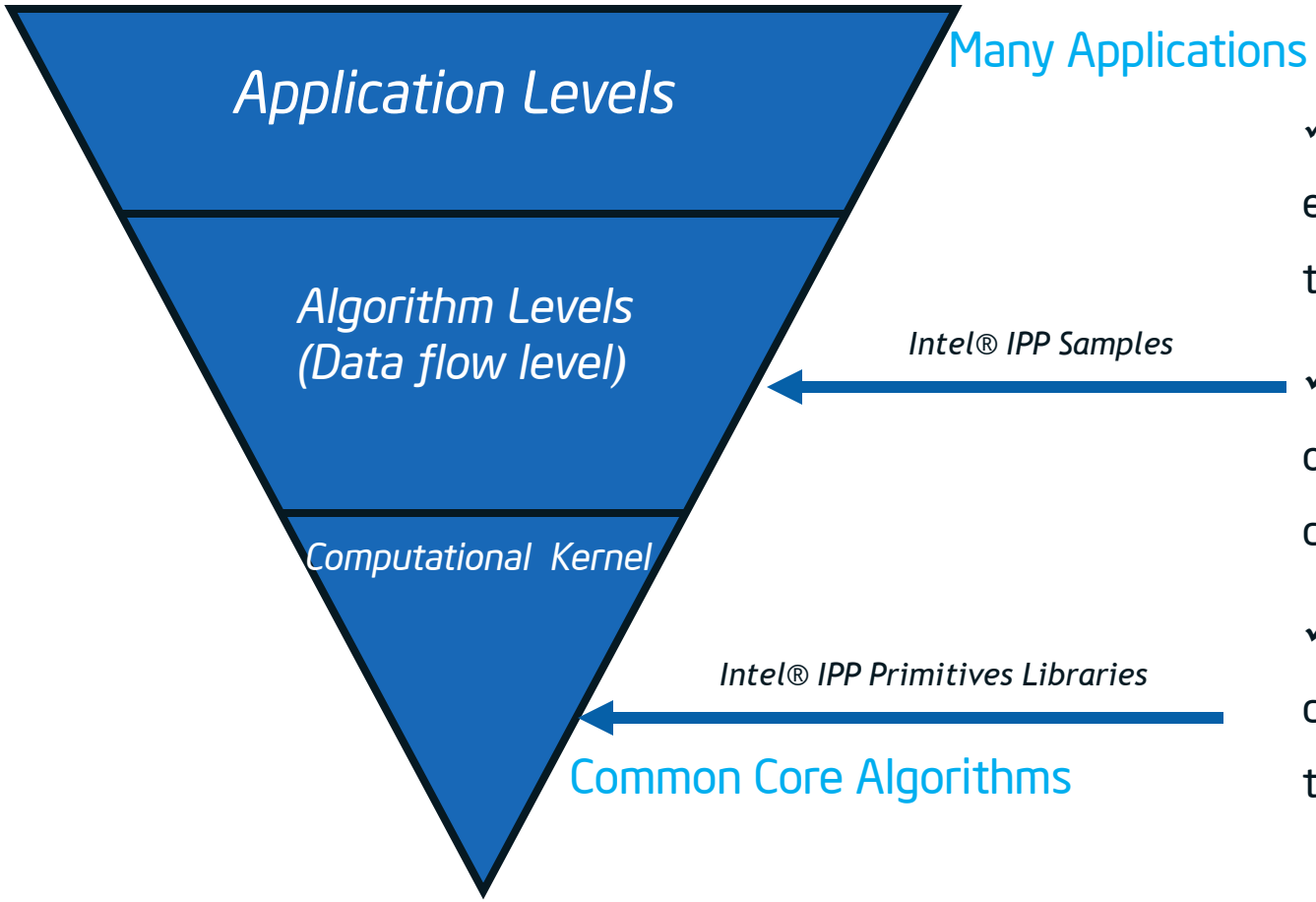
Benefits	•Drawbacks
▪ Run-time dispatching of processor-specific optimizations ▪ Reduced hard-drive footprint compared with a full set of Intel IPP DLLs ▪ Smallest installation package to accommodate use of some of the same Intel IPP functions by multiple applications	▪Application executable requires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

Intel® IPP Supported Linkage Model - quick comparison

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

Intel® IPP provides a lot of flexibility

Conclusion



✓ Take Multiple Levels of efforts to optimized 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

References

Intel® IPP Product Information

- http://www.intel.com/software/products/ipp

Technical Issues

- http://premier.intel.com/

- **Self-help**
  - http://www.intel.com/software/products/ipp
    (Click “Support Resource”)

User Discussion Forum

- http://softwareforums.intel.com/ids/board?board.id=IPP

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.

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