Intel® Integrated Performance Primitives 2008

Boris Sabanin
Intel Corporation
boris.sabanin@intel.com

Abstract

The software product Intel® Integrated Performance Primitives (IPP) and new features are described. The primitives are the building blocks that application developers can easily integrate into their products - applications, components like media plug-ins or high level libraries, in order to significantly increase their performance on Intel and compatible architectures with Windows, Linux or MacOSX operating system installed. The IPP library covers many functional domains: image and signal processing; image coding and data compression; data integrity and cryptography; speech, audio and video coding, and others. Besides of the libraries IPP product contains 50 IPP based Samples released in the source codes. Some of the samples, for example, JPEG2000 image codec and H264 video codec are competitive to the commercial products.

Several important new features were added to the product in 2008 in version IPP 6.0. We will consider two of them in more details - a Deferred Mode Image Processing (DMIP) framework exploiting the CPU cache and multi-core capability, and the IPP functions generated and optimized automatically by a special tool Spiral developed at Carnegie Mellon University.

Keywords: *Software library, high performance, image processing, signal processing, video coding, image coding, data compressing, automatic code generation, multi-core.*

1. Introduction

In spite of the fact that the first signal processing library NSP developed by the IPP team in 1994 failed because Intel and Microsoft could not agree on library's place at the hardware/software territory [1] the team continued design, development and optimization of Intel Performance libraries - Signal Processing SPL, Image Processing IPL, and Speech Recognition RPL. Later, in 2000 the Integrated Performance Primitives were introduced to improve API of the previous generation libraries. Today, IPP library [2,3] covers more functional domains, works in many operating systems; it is optimized for all Intel CPUs and platforms. The IPP Samples released with the library also have been transformed. At the beginning the samples were just the source codes demonstrating how to use and call IPP. Currently, the samples include quite sophisticated applications such as the Face Detection and Ray Tracing demo applications, and image and video codecs which are strongly competitive to commercial products, for example JPEG2000 and H264 codecs. Designing and developing IPP the team introduces new technologies and applies new tools to improve the development and optimization process as, for example, it started to do with the Spiral tool [4]. We will present the first results of the automatically generating library codes in the paper.

2. IPP functionality

IPP is a library containing 10K functions. Many of them are optimized for IA32, Intel®64, IA64, and Atom™. The library works in operating systems Windows, Linux, MacOSX and QNX. IPP consists of 16 functional domains that cover Signal and Image processing, Speech, Audio and Video coding, String processing, Computer Vision, Speech Recognition, Jpeg & Jpeg2000, Lossless Data Compression, Cryptography, Realistic Rendering, Data Integrity, Vector Math and Small Matrix operations. Additionally to the library IPP customers get more than 50 IPP Samples – applications and components released in source codes. They are video codecs MPEG2, MPEG4, H264, VC1 and AVS; audio codecs MP3, AAC, AC3; Jpeg and Jpeg2000 codecs, speech codecs G722, G723, G726, G728; Face detection demo application, Deferred Mode Image Processing framework, Ray Tracing viewer, Data Compression libraries and utilities GZIP, ZLIB, LZO, BZIP2; interfaces for Java, C#, VB, F90, and C++.

One of the sources where the ideas and new functionality requests come from is IPP forum at Intel Software Network web site [5]. The IPP team is actively involved into discussions with IPP users and is very proud of the activity level the IPP customers ask questions, complain of issues and propose solutions.

3. Performance provided in IPP

One of the performance indicators characterizing any signal processing library is performance of the FFT/DFT transform a library provides. According to FFTW [6] in most of the benchmarks FFTW evaluates and presents the IPP FFT is faster than all known FFT implementations. Performance of several single precision read data FFT implementations in MFlops, higher is better, is presented on Fig.1; IPP FFT is on the top.

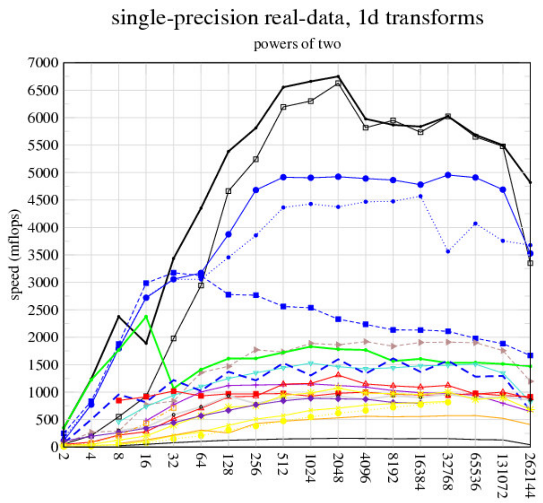

Figure 1. Performance in MFlops of different FFT implementations published on FFTW [3] site.

IPP is fastest in image processing (compared with Matrox), data compression (compared with gzip, zlib, LZO), image coding (compared with IJG), cryptography (compared with OpenSSL). And, finally IPP is a faster library compared to AMD APL library. For example, relative performance of the AMD APL 1.1 and Intel IPP 5.3 functions measured on AMD Opteron system, higher is better (for IPP), is presented on Fig. 2.

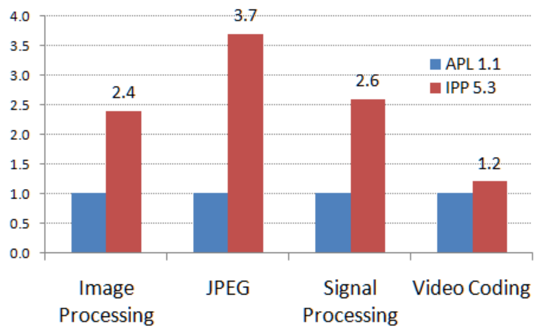

Figure 2. Relative performance of AMD APL 1.1 and Intel IPP 5.3 functions measured on Opteron.

4. New features in IPP 2008

Several new features have been introduced in IPP 6.0. Shorty, they are: new domain Data Integrity with Reed Solomon coding functions, optimization for i7 CPU (codename Nehalem) and Atom™; the high level data compression libraries; the deferred mode image processing framework, and the signal processing functions generated automatically. Let us consider several of the features.

4.1. Deferred Mode Image Processing

DMIP [7] was introduced in response to a requirement from a strategic IPP customer involved in large-scale image processing. The DMIP framework effectively handles processing of large image data that don't fit entirely within the processor L2 cache. Three main features of DMIP improve image processing capability: a) processing of images by the fragments that fit L2 cache; b) using the highly optimized IPP in such fragments processing; c) parallel processing which could be processing of different fragments or execution of different independent branches of a graph. To start such processing you create a task description as a DAG (directed acyclic graph), and translate the graph into a sequence of IPP calls. To complete (deferred) processing you run the generated sequence of IPP calls. For example, you need to filter an image by a harmonization filter. The operation can be expressed as

$$D = \min(Tmx,\max(Tmn,(A-(A-Fb(A))*c))),$$

where Fb is a filter box, Tmn and Tmx are the min and max threshold levels, and C is a constant.
Then at the symbolic level you write C++ code

```
    Image A, D;
    Kernel K;
    Ipp32f C;
    Ipp8u Tmn, Tmx;
    Graph O=To32f(A);
    D=Max(Min(To8u(O-(O-O*K)*C),Tmx),Tmn);
```

The DAG corresponding to the code is presented on Fig. 3.

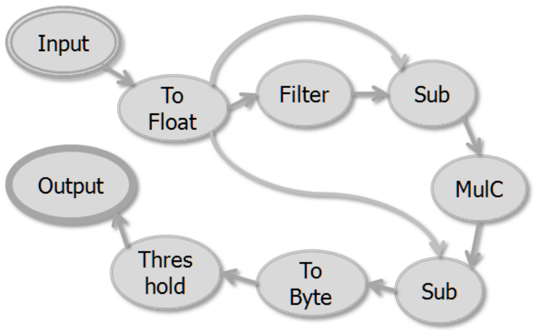

Figure 3. DMIP graph for a harmonization filter

The graph can be compiled once and executed many times. Every operation (a node of the graph) operates upon an image slice (DMIP feature a); in most of the cases it is an IPP call (feature b); the slices are processed in parallel (feature c). Processing with DMIP compared with traditional IPP processing could be up to 3 times faster. On Fig. 4, we can compare performance results for the filtering operation on image 2048x2048 with Harmonization filter 7x7 obtained with DMIP and traditional IPP calls.

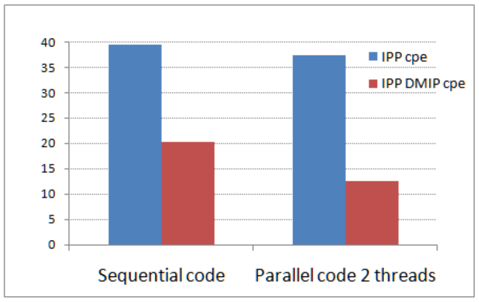

Figure 4. Performance of the Harmonization filter operation with IPP and DMIP. CPU cycles per pixel, the less the better.

4.2. Automatically generated IPP functions

IPP 6.0 includes a new functional domain called IPP Gen. In contrast to other parts of IPP, this library is not implemented by human developers but is entirely computer generated - probably "a first" in high performance library development anywhere. The tool is called Spiral and is developed at Carnegie Mellon University [8].

For given transform to be implemented Spiral generates and evaluates many different possible algorithms represented in an internal math language. Further, Spiral performs optimization such as memory hierarchy optimization, vectorization (for example with SSE3), and parallelization for multiple cores by rewriting mathematical expression. In the end, Spiral outputs the fastest code found which is often faster than existing human written code. On Fig. 5 we compare performance of the Spiral generated code for DFT transform of different size to the IPP existing code in the signal processing library. Performance is given in CPU cycles, the less the better.

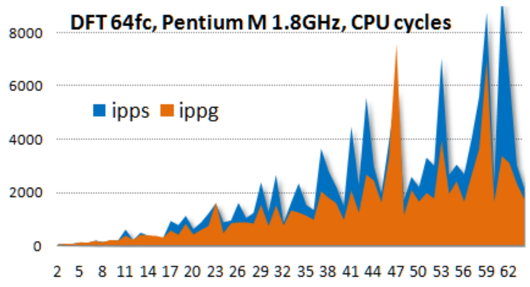

Figure 5. DFT for complex data of double type. Performance of Spiral generated code compared to IPP existing code. CPU cycles, the less the better.

5. Conclusion

Several new important features have been added in IPP 6.0: the deferred mode image processing framework and the automatically generated functions. Both define a direction of the library development: introducing a high abstraction level to better utilize new Intel architecture capabilities and development and optimization automation. IPP is a library which is unique in performance provided on Intel and compatible platforms, in functionality covered, in the technologies the team uses in its development, and in supporting software community.

6. References

[1] Robert A. Burgelman, Strategy Is Destiny, The Free Press, pp. 236, 2002.

[2] Intel® Integrated Performance Primitives (IPP) http://www3.intel.com/cd/software/products/asmo-na/eng/perflib/219780.htm

[3] Stewart Taylor, Optimizing Applications for Multi-Core Processors, Using the Intel® Integrated Performance Primitives, Intel Press, Second Edition, 2007

[4] http://www.spiral.net

[5] Intel® Software Network web site http://softwarecommunity.intel.com/isn/Community/en-US/forums/1274/ShowForum.aspx

[6] http://www.fftw.org/speed/

[7] Alexander Kibkalo, Michael Lotkov, Ignat Rogozhkin, Alexander Turovets, Deferred Image Processing in Intel® IPP Library, Proceedings of the Computer Graphics and Imaging conference, Innsbruck Austria, 2008.

[8] Markus Puschel, Jose M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson and Nicholas Rizzolo, SPIRAL: Code Generation for DSP Transforms, Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation," Vol. 93, No. 2, pp. 232- 275, 2005