A Comparison of VSIPL++ Performance to VSIPL and Mercury SAL

Dr. Thomas Steck
Lockheed Martin MS2, thomas.f.steck@lmco.com

Introduction

Commercial software vendors are starting to sell VSIPL++ implementations for several computer platforms. The promise of improved productivity and inherent support for parallelism is attractive, but performance is critical. This paper presents some results on raw VSIPL++ performance and extends to locally developed performance models being used to verify system sizing for a radar signal processing application. CodeSourcery VSIPL++ 1.1 is compared to Mercury's VSIPL Core Lite and SAL libraries. A Mercury PowerStream 7000 with Freescale 7447A processors is used for testing raw computational performance. Additionally, the Rapid IO fabric of the PowerStream combined with Verari's MPI/Pro is used to take a first look at VSIPL++ parallel performance.

VSIPL++

VSIPL++ is an extension to the Vector Signal Image Processing Library API with support for C++ bindings. The library provides an open-standards API with primitives frequently used in embedded signal and image processing. CodeSourcery's VSIPL++ 1.1 is one of the first commercial versions of VSIPL++ available. In addition to being compliant with the VSIPL++ 1.0 standard, parallel constructs using the Message Passing Interface (MPI) API are incorporated. The claims are VSIPL++ increases productivity though object-oriented syntax, improves performance through layering on other standard libraries and vendor tuned libraries, and provides potential performance gains from parallel processing.

Performance Models

It is sometimes the case that the engineers who specify a computer system are not the same people that develop the final application software that will run on the system. Primitive performance models can assist these engineers with sizing the target computer system. In a particular radar signal processing application of interest, a number of vector processing applications are needed. These operations are

- Vector-Vector Addition
- Scalar-Vector Multiplication
- Vector-Vector Multiplication (dot product)
- Vector-Vector Greater-than (element-wise)
- Vector-Vector Max (element-wise)
- Vector-Vector Logical AND (element-wise)
- Vector-Vector Logical OR (element-wise)

For each of the operations, the time as a function of the vector length is measured. The vector length can be as short as 8 or as long as 128k for the application of interest. Letting $T(x(n))$ represent the time it takes to perform operation x on a vector (or vectors) of length n, the performance model looks like

$$T(F(n)) = aT(f_1(n)) + bT(f_2(n)) + cT(f_3(n)) + \ldots + gT(f_7(n)),$$

with F representing a higher level radar processing function, such as detection thresholding.

Quantitative Comparison

The first set of figures in this section compare the performance of equivalent Mercury SAL, VSIPL, and CodeSourcery VSIPL++ operations. Figure 1 shows how the performance of VSIPL compares to Mercury SAL on the PowerStream 7000. This figure will be updated to reflect the measured performance of the equivalent VSIPL++ call.

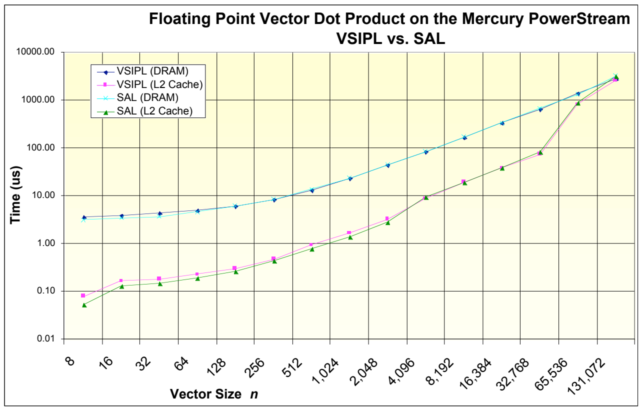

Figure 1. Vector-Vector Multiplication Comparison

Using the performance model for "Mode A", VSIPL and VSIPL++ are compared to SAL based on Mercury's predictions. Estimated times are available for every Mercury SAL call based on whether the data is L1 or L2 cache or DRAM. The times are given for 1,024 point vectors, with all other vector lengths estimated by dividing by 1,024 and multiplying by the new length. Since the L1 and L2 cache are limited in size (32KB and 512KB, respectively), the estimates for the L1 cache stop at 1K words, and for the L2 cache at 32K words. This plot will be updated to include VSIPL++ measured performance.

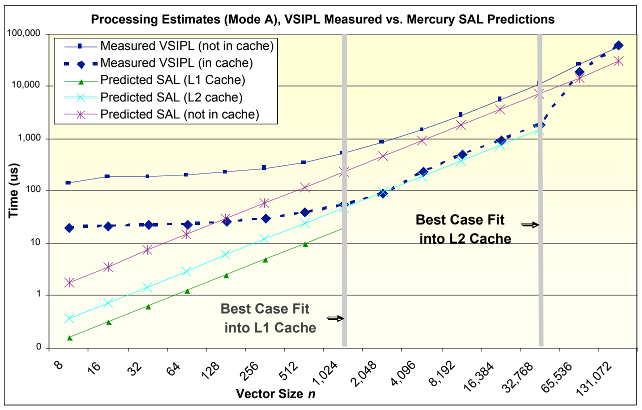

Figure 2. "Mode A" Floating point performance comparison

VSIPL++ Trade Studies

Standard VSIPL has a different function name for each numerical type. For example, vsip_vadd_f performs vector-vector addition on two single-precision floating point vectors, while vsip_vadd_d operates on two double-precision floating point vectors. This means that a global search and replace is required to change numerical precision. With VSIPL++, a single line change is all that is needed to switch between numerical type representations. In the previous section, every floating point number was represented as a single-precision number. The time to perform the same operations is modeled now using integers and double precision floating point. For this study, any additional checks for integer overflow is ignored.

Latency constraints may demand operations be performed in parallel. CodeSourcery VSIPL++ assists the developer by supporting the single program multiple datastream (SPMD) model. Each processor executes the same code, but operates on its local portion of the data. Operands (vectors in this case) are mapped to a set of processors. The user must specify the map. The VSIPL++ operators are then called the same way as they were for the single processor program. If data is needed from another processor, VSIPL++ uses MPI calls to transfer the data on demand.

The performance of two different functions is measured using up to 4 processors. The first function is embarrassingly parallel so no additional communication is required once the data is distributed to the processors. The second function requires a small amount of data to be shared at the edge (first several or last several elements) of the vectors. Figure 3 shows the performance in the embarrassingly parallel case. The data shown is predicted parallel performance for VSIPL and will be updated to include measured parallel performance using VSIPL++.

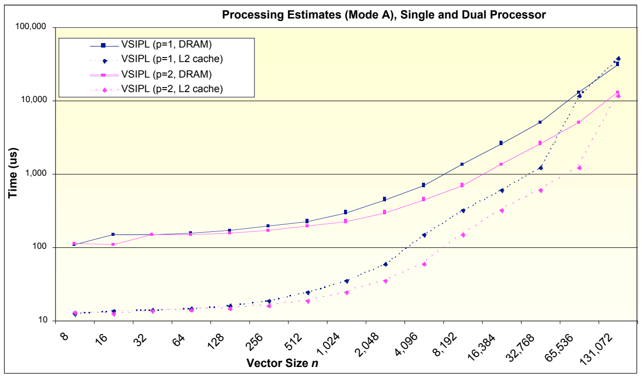

Figure 3. "Mode A" parallel performance.

Conclusions and Recommendations

The use of VSIPL++ is expected to improve productivity by reducing the number of lines of code required for signal processing applications, while preserving or improving serial performance. The support for parallel processing is convenient for some embarrassingly parallel cases, but the lack of non-blocking communications currently limits potential performance gains when using multiple processors. Future versions of VSIPL++ should continue to add support for parallel processing to include overlapping data distributions and non-blocking communicators.

References

[1] www.codesourcery.com

[2] www.mc.com

[3] www.mpi-softtech.com

[4] www.vsipl.org