Title: Pulse Compression Made Easy[1] With VSIPL++

Authors

First Author: **Mr. Brian Chase**
(US citizen)
VSI/Pro Product Manager
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 209
Fax: +1 (205) 314-3475
E-mail: brian@mpi-softtech.com

Second Author: **Mr. WenhaoWu**
(Citizen of China)
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 209
Fax: +1 (205) 314-3475
E-mail: wenhao@mpi-softtech.com

Third Author: **Mr. Dave Leimbach**
(US Citizen)
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 209
Fax: +1 (205) 314-3475
E-mail: dleimbac@mpi-softtech.com

Fourth Author: **Mr. Rick Pancoast**
(US Citizen)
Lockheed Martin Naval Electronics and Surveillance Systems - Surface Systems
199 Borton Landing Road
Moorestown, NJ 08057
E-mail: rick.pancoast@lmco.com

Corresponding and Presenting Author: **Dr. Anthony Skjellum**
(US citizen)
Chief Software Architect,
Verari Systems Software, Inc.
Suite D103, 110 12th Street North
Birmingham, AL 35203
Phone: +1 (205) 314-3471 ext 205
Fax: +1 (205) 314-3475
E-mail: tony@mpi-softtech.com

Submission session: Open sessions.
Presentation type: Presentation

Work area: Case Study Examples of High Performance Embedded Computing

[1] High Productivity – See the abstract for more details.

In December, 2003, Verari Systems Software, Inc. (formerly MPI Software Technology, Inc.) undertook a phase I SBIR effort produce a high level design for a high performance next generation embedded VSIPL product that incorporates advanced language constructs such as those found in the VSIPL++ specification that is now under consideration. The work was divided as follows: 1) Researching strategies to mitigate performance degradation from C++ overhead; 2) High level design work for such a library; 3) Prototype implementation of the new library; 4) Implementing a benchmark application to gauge performance benefit; 5) Reporting results and making recommendations that would apply to the ongoing VSIPL++ effort and any follow-on Phase II work that might be awarded.

The recent introduction of several template based strategies (*e.g.* PETE, POOMA, FACT!, Blitz++, and others) suggests that C++ may soon become a suitable choice for technical and scientific computing application. For certain cases, (*e.g.*, matrix multiplications) inline function calls and template code outperforms straight C code. These similar technologies all share a common set of effective strategies that may be summarized as follows: 1) Avoid excessive temporary copies of objects (both implicit and explicit, where the implicit ones are generated as a side effect of algebraic type expressions); 2) Make shallow copies instead of deep cloning; 3) Pass data by constant reference instead of by value; 4) Use compile time or static polymorphism, such as templates; 5) Deferred evaluation; 6) Template metaprogramming strategies; 7) Inline function calls; 8) Loop fusion and loop unrolling. The authors acquired PETE, the Portable Expression Template Engine, and compiled several examples for the Mercury MCOE 6.0 platform using a 171MHz SPARC machine. The authors also studied the tradeoffs between runtime performance and significant compile time penalties.

Verari's advanced VSIPL package design and prototype implementation strategy is not unlike the architecture of the VSIPL++ reference implementation, which is built as a C++ layer on top of a C VSIPL library. That particular configuration readily appeals to all current vendors of VSIPL compliant middleware who would like to quickly enter the market with a VSIPL++ offering. Figure 1 shown below depicts the layered hierarchical software design used in this study.

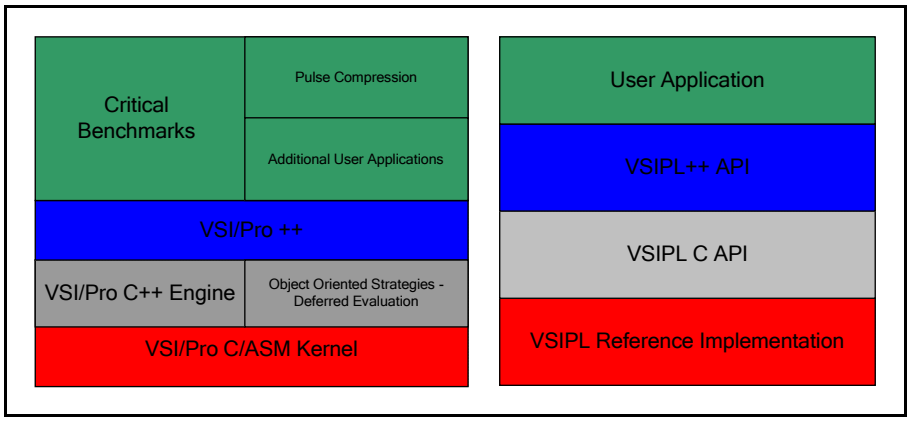

Figure 1 Architecture or the prototype versus VSIPL++

Since the API of the prototype package mirrors that of the VSIPL++ reference implementation, the first experiments were simply tests that are distributed with the VSIPL++ reference code. These tests mainly check for numerical accuracy. The Phase I study then progressed to a more complex test, a commonplace benchmarking application used in radar processing. The pulse compression benchmark typically uses a complex FFT, a complex reference multiply, followed by an inverse complex FFT. The performance of the prototype library on this benchmark was inline with performance figures than can be obtained from the VSI/Pro package. Since the VSI/Pro++ API is similar to VSIPL++, we should expect high performance from the VSIPL++ API when the time comes.

Other than improved performance, another noticeable observation that occurred while porting the pulse compression application from VSIPL to VSIPL++ was the dramatic reduction in both code size and complexity: The original VSIPL benchmark code (which was provided by Lockheed Martin as part of this SBIR effort) consisted of 1600 lines of C code. Yet, the ported VSIPL++ code consisted only 100 lines of C++ code and the whole porting effort only required 2 weeks for one engineer. In fact, Pulse Compression can be fully implemented in a single line of VSIPL++ code:

```
OutputVector = fft_ccrv (fft_ccfv (InputVector) * fft_ccfv (WeightVector));
```

In conclusion, this layered software architectural approach enables high performance, portability, high productivity, and low time to market for commercial vendors of VSIPL standard libraries. Future directions include incorporating the following strategies that will facilitate commercialization of the VSIPL++ standard: 1) Generic programming for higher productivity. 2) Expression manipulation, as well as 3) Deferred evaluation for higher performance.