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

Objective

• Determine the suitability CodeSourcery’s VSIPL++ implementation for real-time signal processing
  – In the context of a Mercury PowerStream 7000 and a targeted signal processing application
  – In comparison to vendor tuned VSIPL Core Lite and Vendor native math call (Mercury’s Scientific Algorithm Library)

Executive Summary

• The current VSIPL++ implementation is rapidly evolving into a legitimate real-time software programming tool
  – improves software development productivity
  – CodeSourcery is rapidly and aggressively tuning code
  – several operations do not yet meet the performance of VSIPL Core Lite or Mercury SAL
  – basic parallel computation obtains improved performance using sound principles
  • users need to ensure communication overhead is properly managed
Application Mode “A” Performance Model Results

- **Performance model built by timing individual vector operations**
- **Using n elements per vector,**
  - $T(f_x(n))$ represents the time to perform a vector operation “x”
  - $T_A(F(n))$ represents the time to complete Application Mode “A”
- **The models in this study use seven basic vector operations, so**
  \[ T(F(n)) = aT(f_1(n)) + bT(f_2(n)) + cT(f_3(n)) + \ldots + gT(f_7(n)) \]

Processing Estimates (Mode A), VSIPL++ vs. VSIPL vs. Mercury SAL

- Predicted SAL
- Measured VSIPL (not in cache)
- Measured VSIPL (in cache)
- Measured VSIPL++ (not in cache)
- Measured VSIPL++ (in cache)
Parallel Performance

Floating Point, Out of Place, Element-wise Vector Multiplication on the Mercury PowerStream

VSIPL++ Serial vs. Parallel

- Using VSIPL++ Parallel Maps to handle data distribution and collection
  - Developers still need to ensure there is sufficient computation to offset additional overhead of distribution data and collecting results