

Introduction:

The peak of the 3.2 GHz STI CELL processor is above 200 Gflop/s for single precision floating-point arithmetics. From the programmer standpoint there are three key techniques in achieving high performance on the CELL processor: SIMD'ization (vectorization), parallelization (between the cores - SPEs) and overlapping of communication and computation (double/tripple buffering – DMAs). The goal of this exercise is to get hands-on experience with the first aspect of optimizing CELL code - SIMD'ization, combined with the standard technique of minimizing loop overhead by unrolling.

What is Given:

You are given a tarball with two subdirectories *strsv_notrans* and *strsv_trans*. The directories contain CELL code to perform two kinds of triangular solve of fixed size in single precision on one SPE.

Each directory contains PPE source (*ppe_strsv.c*), SPE source (*spe_strsv.c*) and a makefile. The PPE code launches a single SPE thread. The thread fetches the data from main memory to Local Store, computes the triangular solve and returns the result back to main memory. The PPE checks the calculations against locally computed results and reports the speed of the SPE code.

The file *strsv_notrans/spe_strsv.c* contains the routine *spe_strsv_notrans()*, which implements one kind of triangular solve in standard C code with two nested loops. The file *strsv_trans/spe_strsv.c* contains the routine *spe_strsv_trans()*, which implements another kind of triangular solve in the same manner.

What is Required:

The triangular solves are implemented on the SPE using standard C code with two nested loops. When implemented in such a way, the first routine runs at the speed of 170 Mflop/s and the second one runs at the speed of 240 Mflop/s. The peak performance of a single SPE in single precision is 25.6 Gflop/s. Your task is to optimize the routines *spe_strsv_notrans()* and *spe_strsv_trans()* to achieve performance equal to or greater than 1.0 Gflop/s. Submit only the source code of the two routines.

How to Do It:

The SPU is an Single Instruction Multiple Data (SIMD) architecture implementing in-order dual-issue instruction scheduling. SIMD'ize (vectorize) your code by using C language extensions or inline assembly and manually unroll loops to hide the latency of loads and stores. Remember, the goal of unrolling is to increase the number of *independent* instructions within the loop body.

What is Okay: Everything. Your code will not be judged based on beauty. It needs to produce correct results and run fast.

What is NOT Okay: There will be NULL credit for code failing the correctness check. There might be partial credit for code running slightly below 1.0 Gflop/s.

Sources:

<http://www.ibm.com/developerworks/power/cell/>

C/C++ Language Extensions for Cell Broadband Engine Architecture

SPU Assembly Language Specification

Synergistic Processor Unit Instruction Set Architecture