

CS 594 - Scientific Computing for Engineers

Homework #3

January 31, 2007

Due: February 14, 2007

Introduction:

The peak of the 3.2 GHz STI CELL processor is above 200 Gflop/s for single precision floating-point arithmetic. From the programmer standpoint there are three key factors in achieving high performance on the CELL processor: SIMD'ization (vectorization), parallelization (between the cores - SPEs), and overlapping of communication and computation (double/triple 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 `dgemv_notrans` and `dgemv_trans`. The directories contain CELL code to perform fixed-size matrix vector product on a single SPE. The former performs the $C=C-AxB$ operation, the latter performs $C=C-trans(A)xB$ operation in double precision, where A is a 32×32 matrix stored in row-major format, B and C are 32-element vectors.

Each directory contains a PPE source (`ppu_dgemv.c`), an SPE source (`spu_dgemv.c`), and the Makefile which compiles the two to a CELL executable. The PPE code launches a single SPE thread. The thread fetches the data from the main memory to the Local Store, computes the matrix-vector product, and returns the result back to the main memory. The PPE checks the calculation against locally computed results, and reports the speed of the SPE code.

The file `dgemv_notrans/spu_dgemv.c` contains the routine `spu_dgemv_tile_notrans()`, which implements the operation $C=C-AxB$ using standard C code with two nested loops. The file `dgemv_trans/spu_dgemv.c` contains the routine `spu_dgemv_tile_trans()`, which implements the operation $C=C-trans(A)xB$ in the same manner.

What is Required:

The matrix-vector multiplication is implemented on the SPE using standard C code with two nested loops. Implemented in such way the code runs at the speed of less than 200 Mflop/s. The peak performance of a single SPU in double precision is 1.8 Gflop/s. Your task is to optimize the routines `spu_dgemv_tile_notrans()` and `spu_dgemv_tile_trans()` to achieve performance in excess of 1.0 Gflop/s. Submit only the source code of the two routines. Comment your source code to make your design choices clear.

How to Do It:

The SPU is a Single Instruction Multiple Data (SIMD) architecture implementing in-order instruction scheduling and static branch prediction. SIMD'ize (vectorize) your code by using C language extensions and manually unroll loops to decrease loop (branch) overhead. Remember, the goal of unrolling is to increase the number of *independent* instruction within the loop body.

What is Okay: pointer arithmetic, explicit type casting, defines and nested defines.

What is NOT Okay: assembly – use C language extensions (intrinsics) *instead*.

Tips:

There is no reason for your source code to exceed 250 lines, and for the object code to exceed 4KB for both routines altogether, but do not consider it a limitation.

Sources:

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

PPU & SPU C/C++ Language Extension Specification

Language_Extensions_for_CBEA.v2.2.1.pdf