Assignment 4: Optimizing `sgemm` on Cell BE

Lars Karlsson (assistant)
Bo Kågström

Due date: 2009-05-28 (17.00)

1 Introduction

Optimizing for a SIMD architecture requires both a capable compiler and an expert programmer. Data structures, alignment issues, loop overhead, and dependencies, are some of the things a programmer should be aware of when writing fast code. In this assignment, you will practice some of the most fundamental optimizations in the process of improving the performance of the single precision general matrix multiply, `sgemm`, on the Cell Broadband Engine processor. You have been given a skeleton code (download the `assignment4-skeleton.tgz` tarball from the course webpage) that is capable of computing the matrix product

$$C \leftarrow C + AB$$

where all matrices are 64×64 and in single precision. The storage format is column major and the matrices are assumed contiguous (leading dimension is 64).

This assignment consists of two parts. In part one, your task is to optimize `sgemm` so that it achieves at least 10 Gflops/s on one SPU. In part two, your task is to enhance the skeleton code so that it can compute much larger products by using several SPUs in parallel.

2 Part I: `sgemm` on a single SPU

Unpack the skeleton code, compile it, and make sure it runs as expected. The performance should be around 0.2 Gflops/s for this unoptimized scalar code. See the `README` file for details on command line options. Read the source code to familiarize yourself with the program logic. Modify the `sgemm_64x64` function in `sgemm-spu.c` so that it performs with at least 10 Gflops/s.

3 Part II: `sgemm` on multiple SPUs

The skeleton code is prepared for performing `sgemm` in parallel using the function from part I as a building block. However, it currently uses only one SPU and it only works for $M = N = K = 64$ since the submatrix load and store functions are lacking functionality for discontinuous submatrices. Begin by completing the implementations of `load_matrix` and `store_matrix` in `sgemm-spu.c`. Then design and implement a parallel algorithm that uses the available SPUs.