

Lab Assignment: Programming for Cell B.E. (1.5hp)

Remark: It is acceptable (and perhaps advisable) to do this lab assignment in groups of two. In this case, each group member must be able to explain all parts of the code.

For technical questions, remote access etc., please refer to Erik Hansson, `eriha @ ida.liu.se`.

Report your solution to Erik Hansson, `eriha @ ida.liu.se`.

Deadline: 1 April 2011.

Exercise 1

Download the Cell BE (TM) documentation and the Cell SDK from the IBM-developerworks website. Install the Cell SDK with the software simulator on your own Linux computer (make sure to have a compatible (older) Linux installation), or get ready for remote connection (ssh) to our PS3 (192.168.0.11) at PELAB, located in the IDA research domain. In the latter case, make sure to have/get an account or guest account (1) on the IDA research domain, and (2) on our PS3. To log on the PS3 from any IDA research computer, use

```
ssh user@lap-33 -p 1234
```

where `user` is your user name on PS3.

Read the Cell BE SDK 3.0 Programming Tutorial from the Cell documentation.

Consider the simple example programs in the SDK, and start modifying these to get somewhat familiar with SDK programming.

Exercise 2

We consider the problem of evaluating a polynomial of degree k ,

$$f(x) = a_k x^k + a_{k-1} x^{k-1} + \dots + a_1 x + a_0$$

over a given sequence $\langle x_1, x_2, \dots, x_N \rangle$ of N elements of a very large input sequence, where the coefficients a_i and operands x_j are single-precision floatingpoint numbers. For simplicity, assume that $k = 6$.

A technique to avoid computing all products x^q for $q > 1$ by repeated multiplications is *Horner's rule*, by which the polynomial above can be computed as

$$f(x) = (\dots(a_k x + a_{k-1})x + a_{k-2})x + \dots + a_1)x + a_0$$

thus with k multiplications and k additions per sequence element.

Write the following Cell programs:

(a) A program running only on the PPE, that computes an input sequence of $N = 2^{24}$ floatingpoint values and stores it in main memory, then computes the sequence $y_j = f(x_j)$ for all $j = 1, \dots, N$ using Horner's rule above, and stores the result in memory again. Add some test code to verify that the output is correct. Measure the time for the actual computation, and record it as baseline for future comparisons. (It is not required to use the AltiVec vector instructions on the PPE).

(b) A program that spawns $p = 6$ SPE threads and computes on each SPE N/p of the elements. Vary p between 1 and 6. What is the speedup over (a)? Explain your observations.

(c) A program like (b) that uses SIMD computation primitives on the SPEs to speed up computation. Determine the speedup over (a) and (b), and explain your observations.

(d) A program like (c) that uses double-buffering to overlap computation with DMA operations for prefetching blocks of operands from, and poststoring blocks of results to main memory. Experiment with different block sizes (from 128 bytes to 16384 bytes) for double buffering. Determine the speedup over (a), (b) and (c), and explain your observations.

You should also try to vary N over different orders of magnitude (as far as it still fits in main memory) to analyze the overhead incurred by the different code variants. At which value of N does it become profitable to switch the computation (i) from PPE to one SPE (with its fastest code variant), and (ii) from one to six SPEs?

The points will be awarded if (a)–(d) are solved properly and you can explain all your code.

Hint: If an SPE program hangs without obvious reason, it is often caused by misalignment of addresses and/or sizes at DMA operations. Try padding structs etc. to work with offsets and sizes that are multiples of 16 or 128 bytes.