CS267 - Spring 2013
Homework 0
Shufps Instruction

Cuong Nguyen

1 About Me

I am a first year EECS graduate student, working on program optimization and correctness, specifically on scientific programs involving floating-point computations. I have a decent background in computer programming and I have done some parallel programming before, but the software tools used in this course such as OpenMP or MPI would be something totally new to me. I expect to improve my programming skills through this course and learn more about software tools available for parallel computing. I am also interested in knowing more about applications of high performance computing (HPC) in science research and the current research state of HPC and parallel computing.

2 Parallel Application: Shufps Instruction

Consider the problem of 4x4 matrix transpose. A sequential algorithm looks as follows, which costs $O(n^2)$ computations where n is the size of the matrix.

```
1  int[16] tranpose(int[16] M) {
2    int[16] T = 0;
3    for (int i = 0; i < 4; i++)
4      for (int j = 0; j < 4; j++)
5        T[4*i+j] = M[4*j+i];
6    return T;
7  }
```

One can utilize SIMD instruction (which stands for single instruction, multiple data) to perform computations on multiple data in one instruction, resulting in significant performance improvement, 4 computations at once. In the problem of matrix transposition, one can use the `shufps` instruction to change the positions of 4 matrix entries at once time. The `shufps` instruction requires 2 arrays of 4 elements and 1 mask. `shufps` selects 2 elements from each arrays based on the mask. 2 elements from the first array are copied to the lower 2 elements in the destination array and 2 elements from the second operand are copied to the higher 2 elements in the destination array. For a more detail explanation of `shufps`, see [1]. The code utilizing `shufps` requires only $O(n)$ computation is as follows.

```
1  int[16]trans(int[16] M) {
```

```
2       S[4:4] = shufps(M[6:4], M[2:4], 11001000b);
3       S[0:4] = shufps(M[11:4], M[6:4], 10010110b);
4       S[12:4] = shufps(M[0:4], M[2:4], 100001101b);
5       S[8:4] = shufps(M[8:4], M[12:4], 11010111b);
6       T[4:4] = shufps(S[11:4], S[1:4], 10111100b);
7       T[12:4] = shufps(S[3:4], S[8:4], 11000011b);
8       T[8:4] = shufps(S[4:4, S[9:4], 11100010b);
9       T[0:4] = shufps(S[12:4], S[0:4], 10110100b);
10    }
```

A challenge here is apparently the complexity of `shufps` instruction semantic which results in convoluted code which might be hard to debug and maintain. It is also unclear for me how to scale this approach for bigger n.

References

[1] http://www.songho.ca/misc/sse/sse.html