EEE4084F Lecture 3 take-home / thinking assignment:

Thinking about parallelizing vector scalar product and matrix multiply

Matrix operations are commonly used to demonstrate and teach parallel coding; the scalar product (or dot product) and matrix multiplication are the ‘usual suspects’. The formulas are provided below:

$$C_{i,j} = \sum_{k} A_{i,k} B_{k,j}$$

Figure 1: Matrix multiply

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Figure 2: Scalar product

Attempt a pseudo code solution for parallelizing both the scalar vector product algorithm and the matrix multiplication algorithm. Assume you would want to implement your solution in C (i.e. your pseudo code should follow C-type operations). We’ll pick this up in the Monday lecture. If time is too limited, just try the scalar product. If you have more time, and are real keen, the by all means experiment with writing and testing real code to see that your suggested solution is valid.

Suggested function prototypes	

float scalarprod (float* a, float* b, int n)
{
// a,b = input vectors of length n
// Function returns the scalar product
}
	void matrix_multiply (float** A, float** B, float** C, int n)
	{
	// A,B = input matrices of size n x n floats
	// C = output matrix of size n x n floats
	}

Some code you could experiment with:

```
// Baseline for scalar product:
t0 = CPU_ticks();  // get initial tick value
    // Do processing ...

    // first initialize the vectors
    for (i=0; i<VECTOR_LEN; i++) {
          a[i] = random_f();
          b[i] = random_f();
      }
    sum = 0;
    for (i=0; i<VECTOR_LEN; i++) {
         sum = sum + (a[i] * b[i]);
       }
// get the time elapsed
t1 = CPU_ticks();  // get final tick value
printf("product = %f\n",sum);
```