CS 61C Fall 2016 Discussion 10

Data Level Parallelism

__m128i _mm_load1_si128( )	returns 128-bit one vector
__m128i _mm_loadu_si128(__m128i *p)	returns 128-bit vector stored at pointer p
__m128i _mm_mul_ps(__m128 a, __m128 b)	returns vector (a0*b0, a1*b1, a2*b2, a3*b3)
void _mm_storeu_si128(__m128i *p, __m128i a)	stores 128-bit vector a at pointer p

1. Implement the following function, which returns the sum of two arrays:

```
static int product_naive(int n, int *a)
{
        int product = 1;
        for (int i = 0; i < n; i++) {
                product *= a[i];
        }
        return product;
}

static int product_vectorized(int n, int *a)
{
         int result[4];
        __m128i prod_v = _________________;

        for (int i = 0; i < _____; i += ___) {   // Vectorised loop
                prod_v =_____________________________________________________;
        }

        _mm_storeu_si128(_________________, _________________);

        for (int i = _______; i < ____; i++) {  // Handle tail case
                result[0] *= ____;
        }
        return ____________________________________________;
}
```

Concurrency

1. Consider the following function:

```
void transferFunds(struct account *from,
                   struct account *to,
                   long cents)
  {
    from->cents -= cents;
    to->cents += cents;
  }
```

a. What are some data races that could occur if this function is called simultaneously from two (or more) threads on the same accounts? (Hint: if the problem isn't obvious, translate the function into MIPS first)

b. How could you fix or avoid these races? Can you do this without hardware support?

Thread Level Parallelism

```
#pragma omp parallelism
{
        /* code here */
}

#pragma omp parallel for
for (int i = 0; i < n; i++) {
        /* code here */
}
```

*Each thread runs a copy of code within the block
*Thread scheduling is non-deterministic

Same as:

```
#pragma omp parallel
{
      #pragma  omp for
      for (int i =0; i < n; i++) {...}
}
```

1. For the following snippets of code
below, circle one of the following to indicate what issue, if any, the code will experience. Then provide a short justification. Assume the default number of threads is greater than 1. Assume no thread will complete before another thread starts executing. Assume arr is an int array with length n.

a)

```
// Set element i of arr to i
#pragma omp parallel
(int i = 0; i < n; i++)
        arr[i] = i;
```

Sometimes incorrect Always incorrect Slower than serial Faster than serial

b)

```
// Set arr to be an array of Fibonacci numbers.
arr[0] = 0;
arr[1] = 1;
#pragma omp parallel for
for (int i = 2; i < n; i++)
        arr[i] = arr[i-1] + arr[i - 2];
```

Sometimes incorrect Always incorrect Slower than serial Faster than serial

c)

```
// Set all elements in arr to 0;
int i;
#pragma omp parallel for
for (i = 0; i < n; i++)
        arr[i] = 0;
```

Sometimes incorrect Always incorrect Slower than serial Faster than serial