How do we improve performance?

- Imagine you want to build a house. How long would it take you?

- What could you do to build that house faster?
Exploiting Parallelism

- Of the computing problems for which performance is important, many have inherent parallelism.

- E.g., computer games:
  - graphics, physics, sound, A.I. etc. can be done separately
  - Furthermore, there is often parallelism within each of these:
    - Each pixel on the screen’s color can be computed independently
    - Non-contacting objects can be updated/simulated independently
    - Artificial intelligence of non-human entities done independently

- E.g., Google queries:
  - Every query is independent
    - Google searches are read-only!!
Exploiting Parallelism at the Instruction level (SIMD)

- Consider adding together two arrays:

```c
void array_add(int A[], int B[], int C[], int length) {
    int i;
    for (i = 0; i < length; ++i) {
        C[i] = A[i] + B[i];
    }
}
```

- You could write assembly for this, something like:

```assembly
lw $t0, 0($a0)
lw $t1, 0($a1)
add $t0, $t1, $t2
sw $t2, 0($a2)
```

(plus all of the address arithmetic, plus the loop control)
Exploiting Parallelism at the Instruction level (SIMD)

- Consider adding together two arrays:

```c
void array_add(int A[], int B[], int C[], int length) {
    int i;
    for (i = 0 ; i < length ; ++ i) {
        C[i] = A[i] + B[i];
    }
}
```

Operating on one element at a time
Exploiting Parallelism at the Instruction level (SIMD)

- Consider adding together two arrays:

```c
void array_add(int A[], int B[], int C[], int length) {
    int i;
    for (i = 0 ; i < length ; ++ i) {
        C[i] = A[i] + B[i];
    }
}
```

Operating on one element at a time
Exploiting Parallelism at the Instruction level (SIMD)

- Consider adding together two arrays:

```c
void array_add(int A[], int B[], int C[], int length) {
    int i;
    for (i = 0 ; i < length ; ++ i) {
        C[i] = A[i] + B[i];
    }
}
```

Operate on MULTIPLE elements
Consider adding together two arrays:

```c
void array_add(int A[], int B[], int C[], int length) {
    int i;
    for (i = 0 ; i < length ; ++ i) {
        C[i] = A[i] + B[i];
    }
}
```

Operate on MULTIPLE elements
Intel **SSE/SSE2** as an example of SIMD

- Added new **128 bit registers** (XMM0 - XMM7), each can store:
  - 4 single precision FP values (SSE) \(4 \times 32\) bits
  - 2 double precision FP values (SSE2) \(2 \times 64\) bits
  - 16 byte values (SSE2) \(16 \times 8\) bits
  - 8 word values (SSE2) \(8 \times 16\) bits
  - 4 double word values (SSE2) \(4 \times 32\) bits
  - 1 128-bit integer value (SSE2) \(1 \times 128\) bits

<table>
<thead>
<tr>
<th>(A(3))</th>
<th>(A(2))</th>
<th>(A(1))</th>
<th>(A(0))</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0 (32 bits)</td>
<td>4.0 (32 bits)</td>
<td>3.5 (32 bits)</td>
<td>-2.0 (32 bits)</td>
</tr>
<tr>
<td>-1.5 (32 bits)</td>
<td>2.0 (32 bits)</td>
<td>1.7 (32 bits)</td>
<td>2.3 (32 bits)</td>
</tr>
<tr>
<td>2.5 (32 bits)</td>
<td>6.0 (32 bits)</td>
<td>5.2 (32 bits)</td>
<td>0.3 (32 bits)</td>
</tr>
</tbody>
</table>

\(C(3)\) \(C(2)\) \(C(1)\) \(C(0)\)
SIMD Extensions

More than 70 instructions. Arithmetic Operations supported: Addition, Subtraction, Mult, Division, Square Root, Maximum, Minimum. Can operate on Floating point or Integer data.
Clicker Question

To exploit parallelism in my code, could I put 8 characters and 2 integers into an XMM register?

a) Yes
b) No
Annotated SSE code for summing an array

\[
\begin{align*}
\text{movdqa} & \quad (%eax, %edx, 4), %xmm0 \\
& \quad \# \text{load } A[i] \text{ to } A[i+3] \\
\text{movdqa} & \quad (%ebx, %edx, 4), %xmm1 \\
& \quad \# \text{load } B[i] \text{ to } B[i+3] \\
padd & \quad %xmm0, %xmm1 \\
& \quad \# CCCC = AAAA + BBBB \\
\text{movdqa} & \quad %xmm1, (%ecx, %edx, 4) \\
& \quad \# \text{store } C[i] \text{ to } C[i+3] \\
\text{addl} & \quad $4, %edx \\
& \quad \# i += 4
\end{align*}
\]

(loop control code)
Is it always that easy?

- No. Not always. Let’s look at a little more challenging one.

```c
unsigned
sum_array(unsigned *array, int length) {
    int total = 0;
    for (int i = 0 ; i < length ; ++ i) {
        total += array[i];
    }
    return total;
}
```

- Is there parallelism here?

array(0) (4) (8) 12 (6)
Exposing the parallelism

```c
unsigned
sum_array(unsigned *array, int length) {
    int total = 0;

    for (int i = 0 ; i < length ; ++ i) {
        total += array[i];
    }

    return total;
}
```
We first need to restructure the code

```c
unsigned
sum_array2(unsigned *array, int length) {
    unsigned total, i;
    unsigned temp[4] = {0, 0, 0, 0};
    for (i = 0 ; i < length & ~0x3 ; i += 4) {
        // SIMD
        temp[0] += array[i];
        temp[1] += array[i+1];
        temp[2] += array[i+2];
        temp[3] += array[i+3];
    }
    for ( ; i < length ; ++ i) {
        total += array[i];
    }
    return total;
}
```
Then we can write SIMD code for the hot part

```c
unsigned
sum_array2(unsigned *array, int length) {
    unsigned total, i;
    unsigned temp[4] = {0, 0, 0, 0};
    for (i = 0 ; i < length & ~0x3 ; i += 4) {
        temp[0] += array[i];
        temp[1] += array[i+1];
        temp[2] += array[i+2];
        temp[3] += array[i+3];
    }
    for ( ; i < length ; ++i) {
        total += array[i];
    }
    return total;
}
```
Summary

- Performance is of primary concern in some applications
  - Games, servers, mobile devices, super computers

- Many important applications have parallelism
  - Exploiting it is a good way to speed up programs.

- Single Instruction Multiple Data (SIMD) does this at ISA level
  - Registers hold multiple data items, instruction operate on them
  - Can achieve factor or 2, 4, 8 speedups on kernels
  - May require some restructuring of code to expose parallelism
    - Create temporary vectors, which are then reduced
    - Deal with remainder of array (if not evenly divisible)