Elon University

Course Information

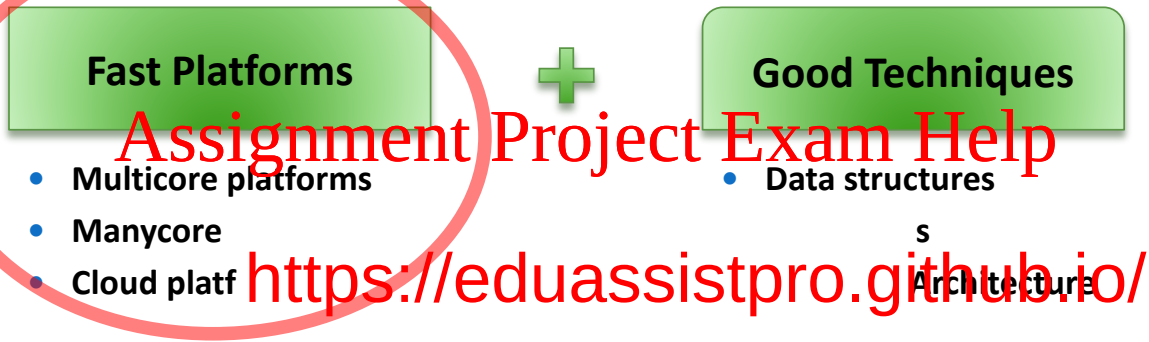
- **Lectures:**
  - Tuesday and Thursday 6:00pm-7:20pm ET
- **Office Hours:**
  - Instructor Office Hours: Wednesdays 4:30pm-5:30pm ET
  - TA Office Hours: T
- **Course Links:**
  - **Canvas:** https://canvas.cmu.edu/cou course-schedule
  - **Piazza:** https://piazza.com/class/jtkm
  - **Gradescope:** https://www.gradescop 1050

https://canvas.cmu.edu/courses/21510/pages/course-schedule

Course Goal

- **Course Goal**
  - When your research/application needs to be fast, you will be able to: **FLIRT**
    1. **F**eel comfortable hacking up a solution
    2. **L**everage e
    3. **I**ndicate w
    4. **R**eason abo
    5. **T**ake care of potential performance

How to Write Fast Code?



- We focus on the fast hardware in this l
- Combines with **good techniques** to produce fast code…

  …in order to solve a problem or improve an outcome

Outline

- **Landscape of Computing Platforms**
- Hardware Architectures
  - Multicore vs
  - Instruction le
  - SIMD
  - Simultaneous multithreading
  - Memory hierarchy
  - System hierarchy
- How to Write Fast Code?

Landscape of Computing Platforms

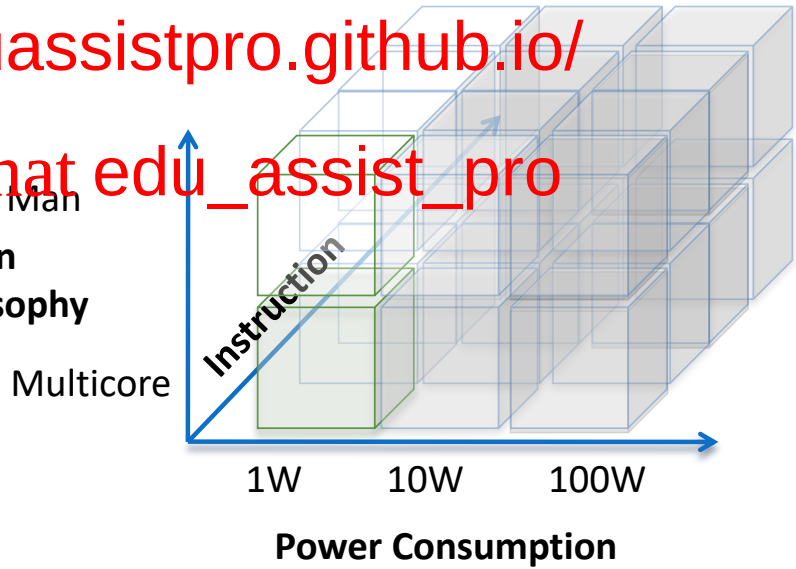
- This is what you see most often

  The **exponential increase** in transi integration, and t **flattening** of cloc speed, power consumption, and performance per clock

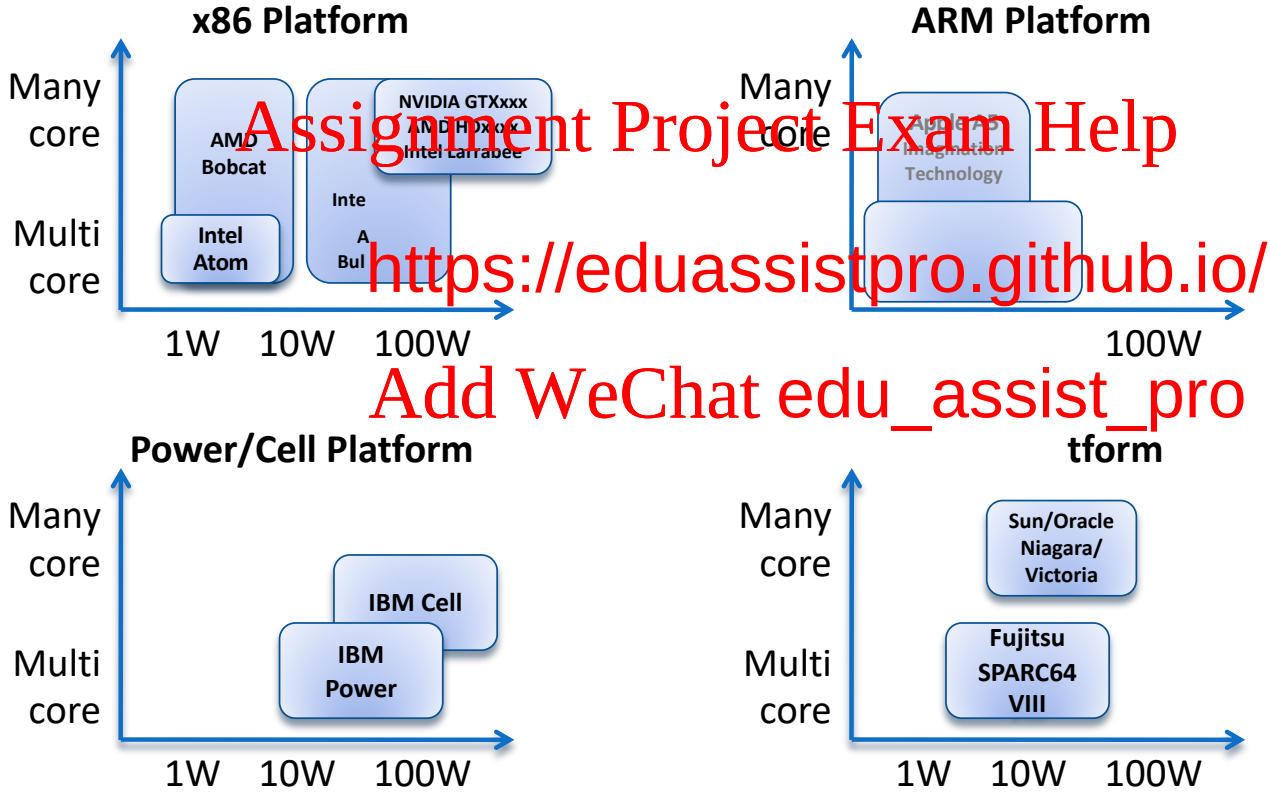
Landscape of Computing Platforms

- This is triggering a flurry of technology innovations!
- Let's look at the innovations in three axis:
  - Instruction Set
  - Design Philoso
  - Power Consump

Instruction Set
- **x86**
- **ARM**
- **Power/Cell**
- **SPARC**



Computing Technology Innovations

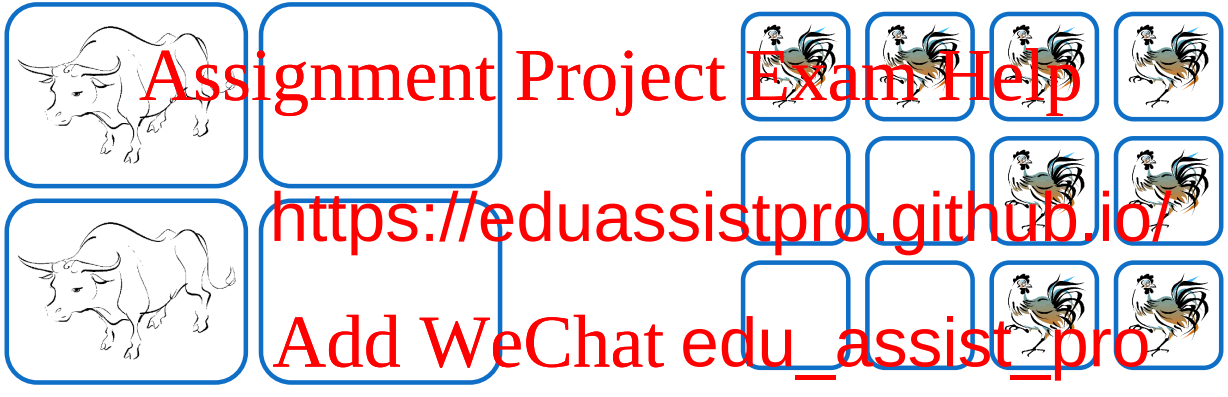


Outline

- Landscape of Computing Platforms
- **Hardware Architectures**
  - **Multicore vs**
  - Instruction le
  - SIMD
  - Simultaneous multithreading
  - Memory hierarchy
  - System hierarchy
- How to Write Fast Code?

(1) Multicore vs Manycore Processors

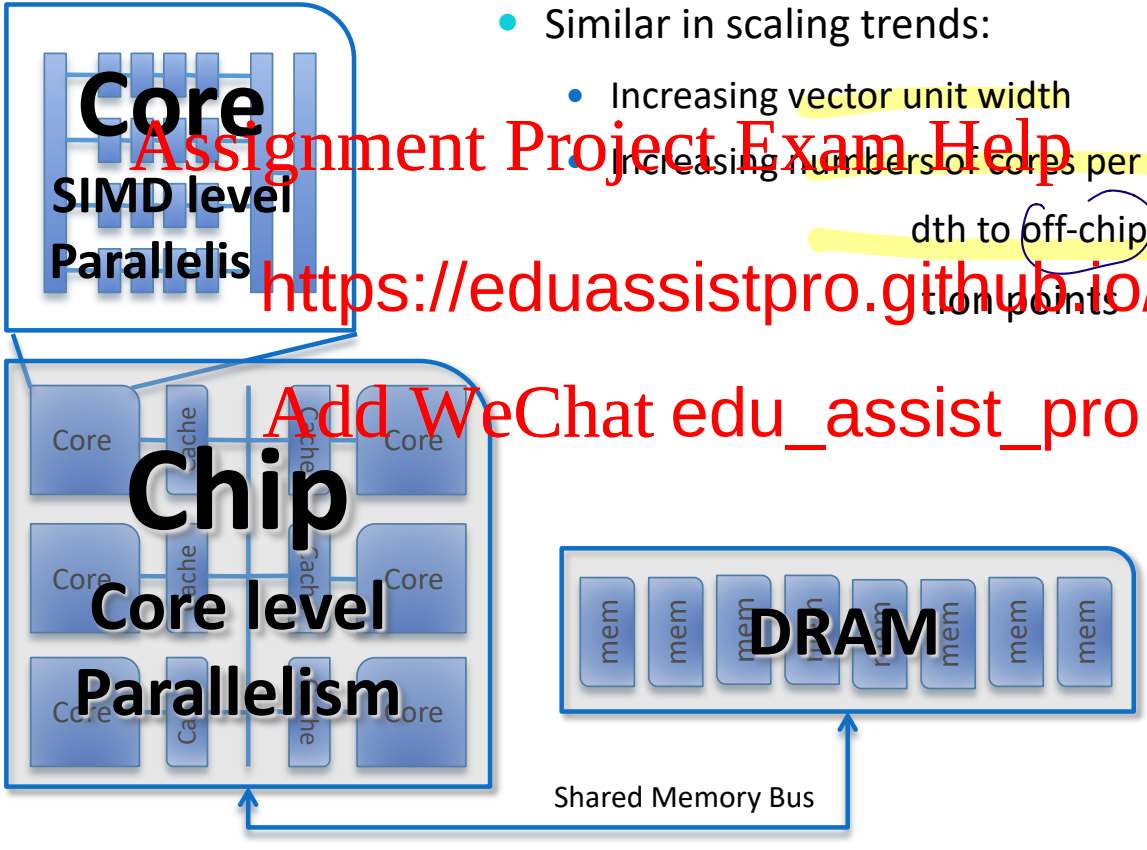
- What's the difference between Multicore vs Manycore?



Multicore Manycore

- Multicore: yoke of oxen
  - Each core optimized for executing a single thread
- Manycore: flock of chickens
  - Cores optimized for aggregate throughput, deemphasizing individual performance

Multicore and Manycore Parallelism



- Similar in scaling trends:
  - Increasing vector unit width
  - Increasing numbers of cores per die

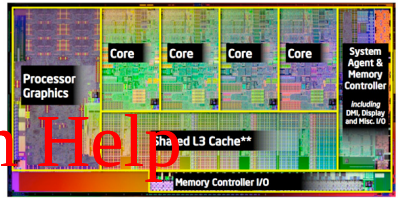
dth to off-chip memory

tion points

Significant Architectural Difference

SIMD single instruction multiple data.

Specifications	Core i7 2600K	GTX580	
Processing Elements	4 cores, 8 way SIMD @2.5-3.4 G	16 cores, 16 way SIMD, dual issue	0.46x
Resident Threads (max)	4 cores, 2 thr 8 width SI 64 strands	SIMD 24,576 strands	
SP GFLOP/s	160-218	1587	7.3x - 9.9x
Memory Bandwidth	21.4GB/s – 42.7GB/s	192.4 GB/s	4.5x - 9.0x
Die info	995M Transistors 32nm process 216mm^2	3B Transistors 40nm process 520mm^2	



Intel Core i7-2600K

NVIDIA GTX580

Outline

- Landscape of Computing Platforms
- **Hardware Architectures**
  - Multicore vs
  - **Instruction le**
  - SIMD
  - Simultaneous multithreading
  - Memory hierarchy
  - System hierarchy
- How to Write Fast Code?

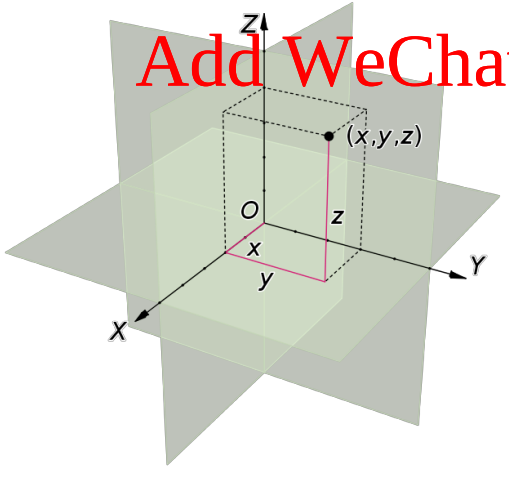
Load-store Architecture: MIPS

- Data and instructions are loaded from memory for processing
  - Follows the Von Neuman Architecture – separation of CPU and memory

(2) Instruction Level Parallelism (ILP)

- Instructions in a sequence that can be computed at the same time

- Example:
  - Euclidean distance between two points

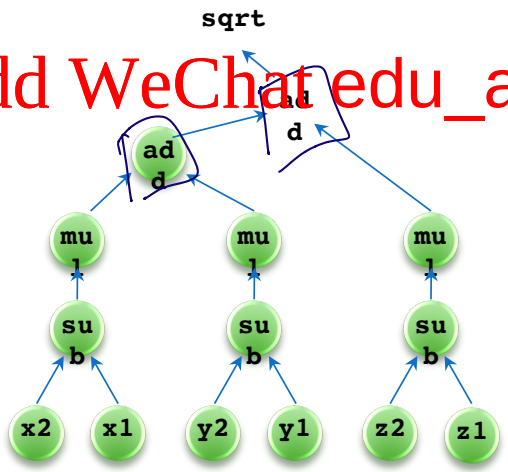



```
ld r1, x1
ld r2, y1
ld r3, z1
ld r4, x2
ld r5, y2
ld r6, z2
sub r1, r1, r4
sub r2, r2, r5
sub r3, r3, r6
mul r1, r1, r1
mul r2, r2, r2
mul r3, r3, r3
add r1, r1, r2
add r1, r1, r3
sqrt r1, r1
st d, r1
```

Multiple Valid Instruction Sequence

- Compilers may produce valid instruction sequences that have less ILP when executed in order


```
ld r1, x1
ld r4, x2
sub r1, r4, r1
mul r1, r1, r1
ld r2, y1
ld r5, y2
sub r2, r5, r2
mul r2, r2, r2
add r1, r1, r2
ld r3, z1
ld r6, z2
sub r3, r6, r3
mul r3, r3, r3
add r1, r1, r3
sqrt r1, r1
st d, r1
```




```
ld r1, x1
ld r2, y1
ld r3, z1
ld r4, x2
ld r5, y2
ld r6, z2
sub r1, r1, r4
sub r2, r2, r5
sub r3, r3, r6
mul r1, r1, r1
mul r2, r2, r2
mul r3, r3, r3
add r1, r1, r2
add r1, r1, r3
sqrt r1, r1
st d, r1
```

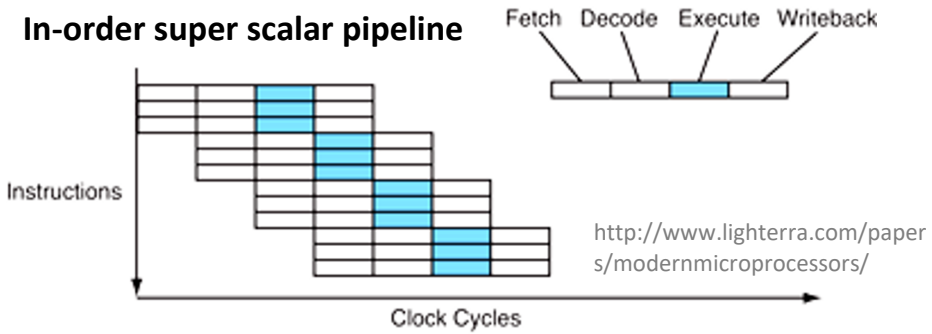
Traditional In-order Pipeline

- An **in-order processor pipeline** could run into issues
  - Reduced ILP and Read/Write operand dependency


```
ld r1, x1
ld r4, x2
sub r1, r4, r1
mul r1, r1, r1
ld r2, y1
ld r5, y2
sub r2, r5, r2
mul r2, r2, r2
add r1, r1, r2
ld r3, z1
ld r6, z2
sub r3, r6, r3
mul r3, r3, r3
add r1, r1, r3
sqrt r1, r1
st d, r1
```

In-order scalar pipeline

In-order super scalar pipeline

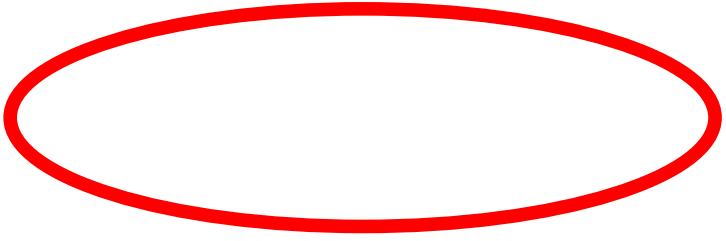


http://www.lighterra.com/papers/modernmicroprocessors/

Later Out-of-order Pipelines

- A **out-of-order processor pipeline**:
  - Allows instruction re-ordering
  - Uses register-renaming


```
ld r1, x1
ld r4, x2
sub r1, r4, r1
mul r1, r1, r1
ld r2, y1
ld r5, y2
sub r2, r5, r2
mul r2, r2, r2
add r1, r1, r2
ld r3, z1
ld r6, z2
sub r3, r6, r3
mul r3, r3, r3
add r1, r1, r3
sqrt r1, r1
st d, r1
```



Exploiting ILP

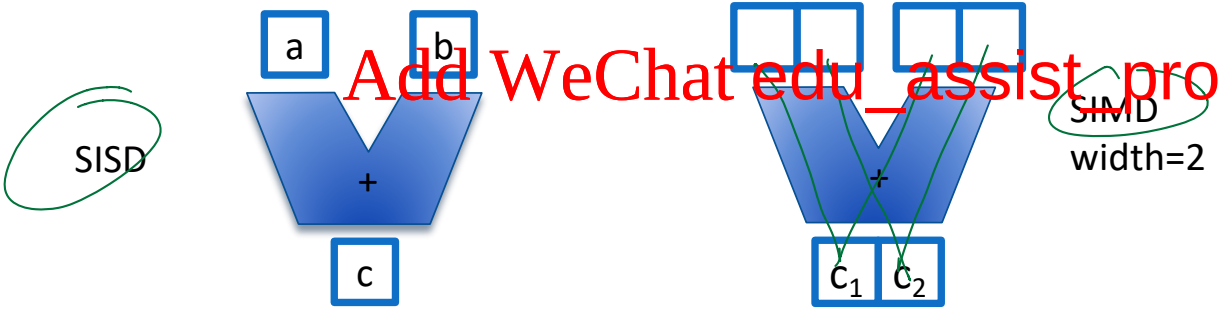
- Motivation:
  - Given a single, sequential stream of instructions, how to execute it as fast as possible?
- Advantages:
  - No changes in s
- Disadvantages:
  - Significantly more complex processor architecture
  - Longer to design the processor
  - Longer to verify the correctness of the processor design
  - Consumes more energy than simple in-order processor

Outline

- Landscape of Computing Platforms
- **Hardware Architectures**
  - Multicore vs
  - Instruction le
  - **SIMD**
  - Simultaneous multithreading
  - Memory hierarchy
  - System hierarchy
- How to Write Fast Code?

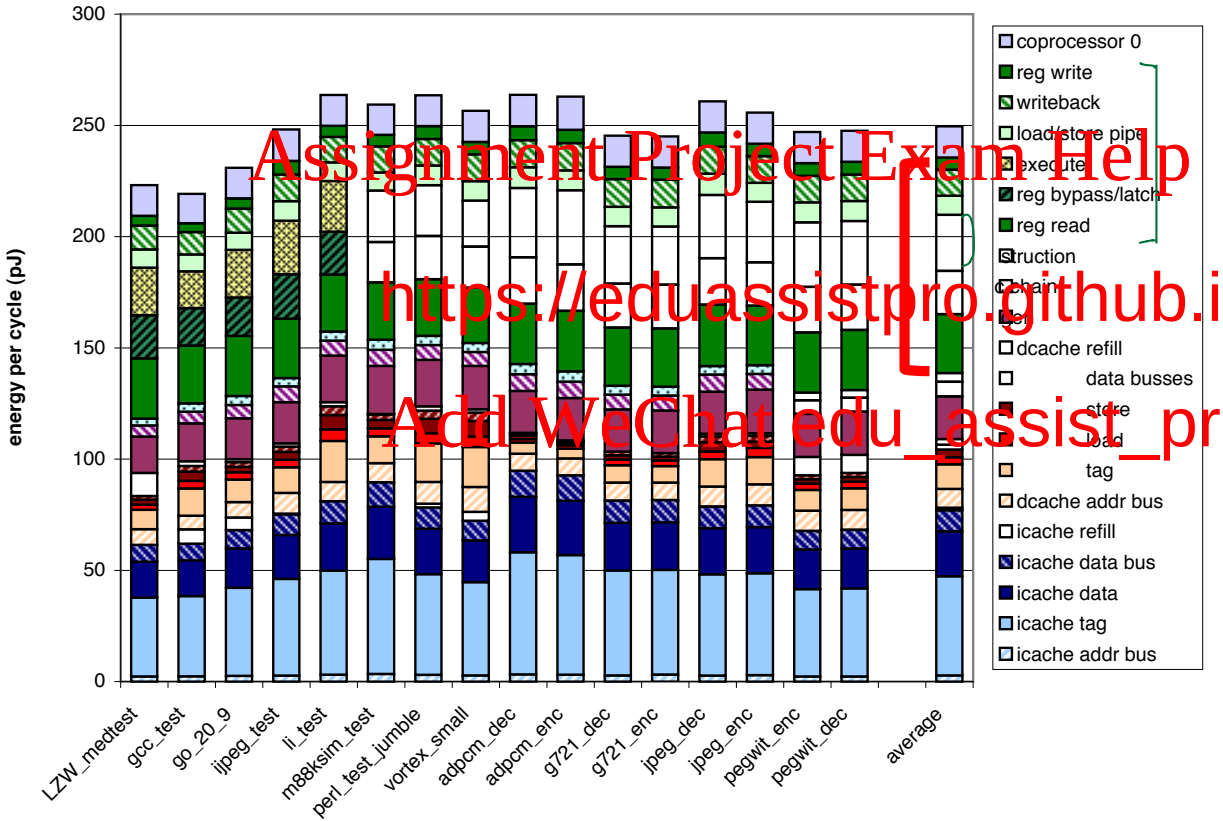
(3) SIMD – Single Inst, Multiple Data

- Taxonomy proposed by Michael J Flynn in 1966
  - **SISD**: No parallelism
  - **SIMD**: Exploits data parallelism
  - **MISD**: Redundancy (used in Space Shuttle flight control)
  - **MIMD**: Distribu

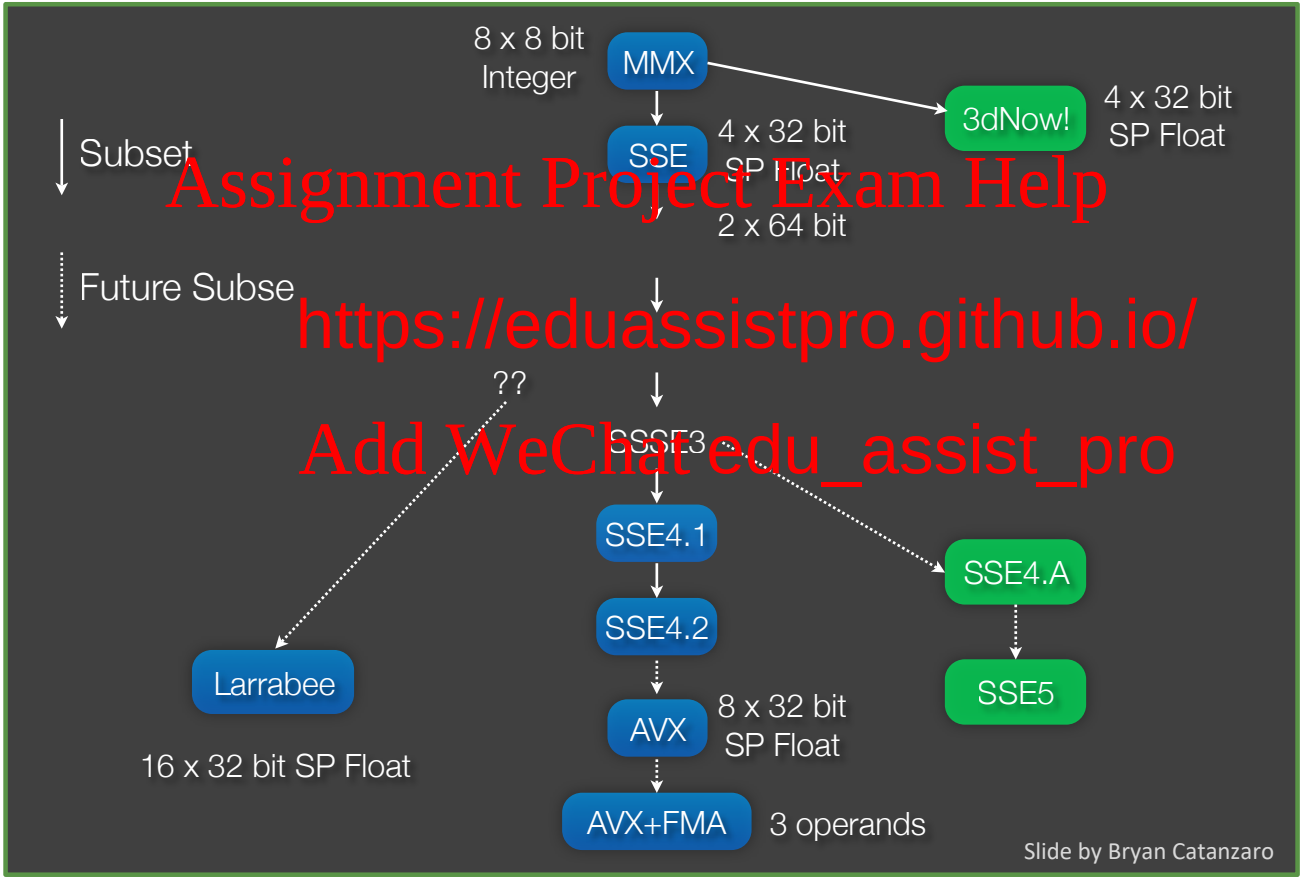


- SIMD:
  - Can be area and power efficient: Amortize control overhead over SIMD width
  - Parallelism exposed to programmer & compiler

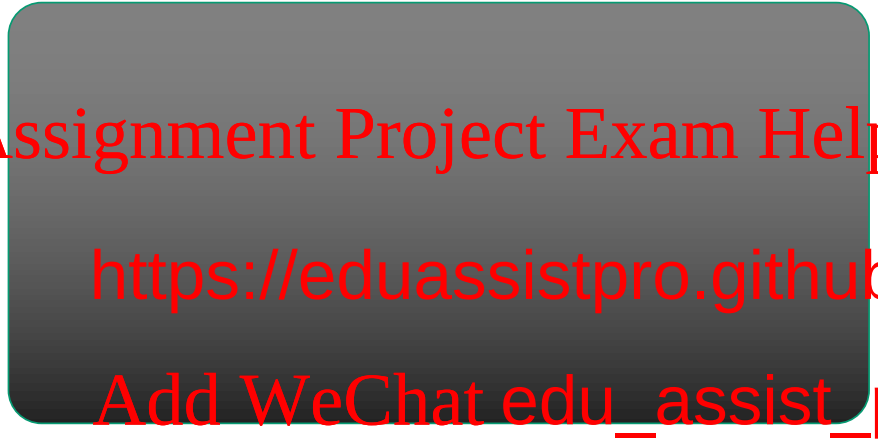
Processor Power Consumption



A Brief History of x86 SIMD



What to do with SIMD?



4 way SIMD (SSE) 16 way SIMD (LRB)

- Neglecting SIMD is expensive
  - AVX: 8 way SIMD, Larrabee: 16 way SIMD
  - NVIDIA: 32 way SIMD, ATI: 64 way SIMD

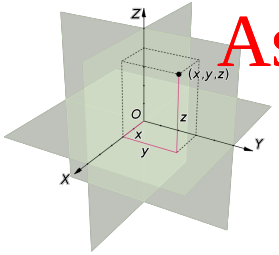
How to Utilize SIMD Capabilities?

- **Compilers:**
  - Option available in GCC, ICC, and others
  - GCC:
    - Enable by options such as:
      - -ftree-vect
      - -msse2
      - -ffast-math
      - -fassociative-math
    - Also enabled by default with "-O3"
  - Examples at:
    - `http://gcc.gnu.org/projects/tree-ssa/vectorization.html#using`

- **Hand optimization:**
  - Optimization using intrinsics

Example: Exploiting SIMD Parallelism

- Applied to finding distance between two points:




```
V1 = _MM_LOAD(&X1[0])
V2 = _MM_LOAD(&Y1[0])
V1 = _MM_SUB(V2, V1)
V1 = _MM_MUL(V1, V1)
_MM_STORE(&res[0], V1)
add r1, &res[0], &res[1]
add r1, r1, &res[2]
sqrt r1, r1
st d, r1
```

```
ld r1, x1
ld r2, x2
ld r3, x3
ld r4, y1
ld r5, y2
ld r6, y3
sub r1, r4, r1
sub r2, r5, r2
sub r3, r6, r3
mul r1, r1, r1
mul r2, r2, r2
mul r3, r3, r3
add r1, r1, r2
add r1, r1, r3
sqrt r1, r1
st d, r1
```

Example…

```
#include <stdio.h>
#include <pmmintrin.h>  //  header for SSE3
#define k 32

int main(){
  float x[k]; float y[k];          // vectors of length k
  __m128 X, Y;                     // 128-bit values
  __m128 acc = _mm_setzero_ps(); // set to (0, 0, 0, 0)
  float inner_prod, temp
  int i, j;

  for (j=0; j<k; j++){ x

  for(i = 0; i < k - 4; i += 4) {
    X = _mm_load_ps(&x[i]);    // load chunk of 4
    Y = _mm_load_ps(y + i);    // alternate way, p
    acc = _mm_add_ps(acc, _mm_mul_ps(X, Y));
  }
  _mm_store_ps(&temp[0], acc);   // store acc into an array of floats
  inner_prod = temp[0] + temp[1] + temp[2] + temp[3];

  // add the remaining values
  for(; i < k; i++)
    inner_prod += x[i] * y[i];

  printf("Inner product is: %f\n", inner_prod);

  return 0;
}
```

Compile using:

```
gcc –lm –msse3 –O2 test.c
```

Run with:

```
./a.out
```

```
product is: 32.000
```


SSE Version	gcc header
SSE	xmmintrin.h
SSE2	emmintrin.h
SSE3	Pmmintrin.h

Common SSE Intrinsics

```
__m128 _mm_setzero_ps(void)// set to (0, 0, 0, 0)
__m128 _mm_set_ps1(float f)// set to (f, f, f, f)
__m128 _mm_set_ps(float w, float x, float y , float z) // set to (z,y,x,w)
__m128 _mm_setr_ps(                                    ) // set to (w,x,y,z)
__m128 _mm_add_ps(_
__m128 _mm_sub_ps(__m128, __m128)
__m128 _mm_mul_ps(__m128, __m128)
__m128 _mm_load_ps(float *base_addr) // load 4 floats from base_addr
__m128 _mm_loadr_ps(float *base_addr) // load in reverse order void
_mm_store_ps(float *addr, __m128 val) // write val into location addr
```

Exploiting SIMD Parallelism

- Motivation:
  - Reduce processor complexity by explicitly representing instruction-level parallelism in vector form
- Advantages:
  - Power-efficient
  - Exploitable in many compute-intensive
- Disadvantages:
  - Explicit representation in vector instructions
  - Software requires re-compilation to take advantage of new SIMD capabilities
  - May require hand-tuning to exploit full benefit

Outline

- Landscape of Computing Platforms
- **Hardware Architectures**
  - Multicore vs
  - Instruction le
  - SIMD
  - **Simultaneous multithreading**
  - Memory hierarchy
  - System hierarchy
- How to Write Fast Code?

https://canvas.cmu.edu/courses/21510/quizzes/55266

Outline

- Landscape of Computing Platforms
- **Hardware Architectures**
  - Multicore vs
  - Instruction le
  - SIMD
  - **Simultaneous multithreading**
  - Memory hierarchy
  - System hierarchy
- How to Write Fast Code?

(4) Simultaneous Multithreading

- Question:
  - When multiple threads of work are available, how do we improve performance?

```
ld r1, x1
ld r4, y1
sub r1, r4, r1
mul r1, r1, r1
ld r2, x2
ld r5, y2
sub r2, r5, r2
mul r2, r2, r2
add r1, r1, r2
ld r3, x3
ld r6, y3
sub r3, r6, r3
mul r3, r3, r3
add r1, r1, r3
sqrt r1, r1
st d, r1
```

Simultaneous Multithreading (SMT)

- Question:
  - When multiple threads of work are available, how do we improve performance?

different tasks (instructions)

Exploiting SMT Parallelism

- Motivation:
  - Capturing the opportunity to run faster when more than one thread of instructions are available
- Advantages:
  - Gain power-effi ization
- Disadvantages:
  - Requires multiple threads available
  - **May trigger conflicts in shared cache during execution**
  - Does not improve latency of each thread

Outline

- Landscape of Computing Platforms
- **Hardware Architectures**
  - Multicore vs
  - Instruction le
  - SIMD
  - Simultaneous multithreading
  - **Memory hierarchy**
  - System hierarchy
- How to Write Fast Code?

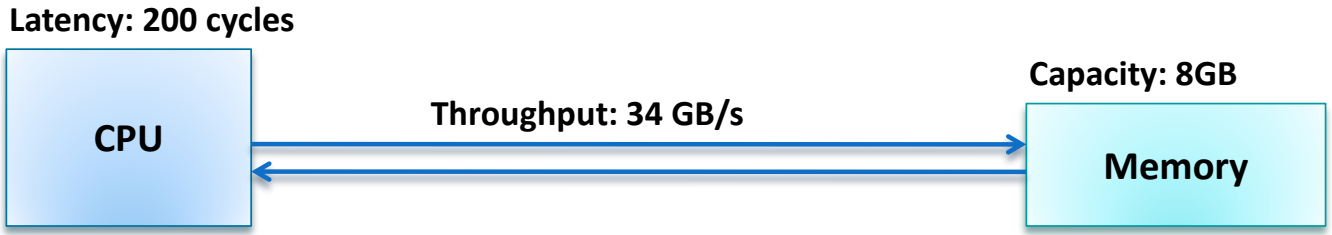
(5) Memory Hierarchy

- **Cache**:
  - A piece of fast memory close to the compute modules in a microprocessor
  - Allows faster access to a limited amount of data
  - Physical properties of wires and transistors determines trade-off between **cache capacity**

Capacity: Size, e.g. number of byt

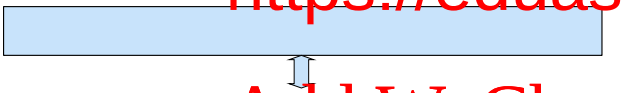
Latency: From start to finish, in u CPU clock cycles

Throughput: Tasks accomplished per unit time, e.g. GB/s



(5) Memory Hierarchy

- **Cache**:
  - A piece of fast memory close to the compute modules in a microprocessor
  - Allows faster access to a limited amount of data
  - Physical properties of wires and transistors determines trade-off between **cache capacity**



ity, (throughput), (latency)

tes, 0.1 GB/s, ~1-10B cycles

6 GB/s, ~100 cycles

16GB, 34.08 GB/s, ~200 cycles

8 MB, 108.8 GB/s, 26-31 cycles

256 KB, 108.8 GB/s, 12 cycles

32 KB Data Cache, 163.2 GB/s, 4-6 cycles

Intel Core2 – 2600k @ 3.4GHz
(viewed from one core)

Working with a Memory Hierarchy

- **Writing fast code → get data from the fastest (closest) level**
- When requesting data from a level in memory hierarchy of limited size, there are two outcomes
  - **Hit**: Data is ava
  - **Miss**: Data is m
- **For fast code → minimize misses, maxi**
- What **application behavior** can a memory hierarchy help with?

Working with a Memory Hierarchy

- **Principle of Locality**
  - The phenomenon of the same value or related storage locations being frequently accessed, usually amenable to performance optimization

- **Temporal Locality**
  - Reuse of specific and/or resources within relatively small time durations

- **Spatial Locality**
  - Use of data elements within relatively close storage locations

When would you get a miss?

- Three types of misses in a memory hierarchy
  - **Compulsory** misses: caused by the first reference
  - **Capacity** misses: due to the finite size of the memory hierarchy
  - **Conflict** misses lly avoidable

Exploiting Benefits of Mem Hierarchy

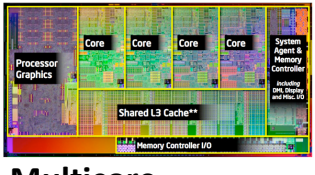
- Motivation:
  - When application exhibit data locality, cache/memory provides faster access to frequently used data
- Advantages:
  - Allows faster a
  - Caches are managed storage: transpare for functional purposes
  - Lower the energy consumption when g ”
- Disadvantages:
  - When using multiple threads share a cache, they will compete for cache space

Outline

- Landscape of Computing Platforms
- Hardware Architectures
  1. Multicore vs Manycore
  2. Instruction lev
  3. SIMD
  4. Simultaneous multithreading
  5. Memory hierarchy
  6. System hierarchy
- How to Write Fast Code?

(6) System Architecture

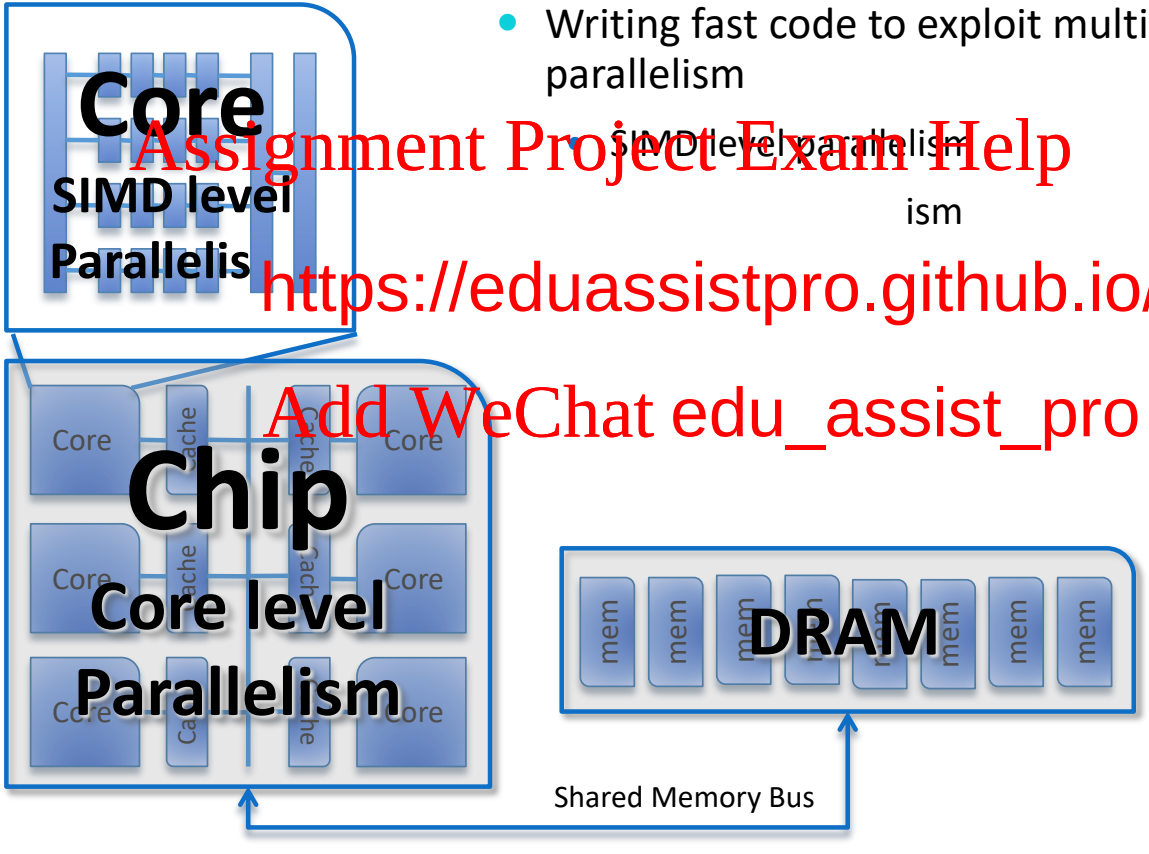
- A Journey through the opportunities to write ***fast code*** with:
  - Multicore
  - Manycore
  - The Cloud



Multicore

Manycore

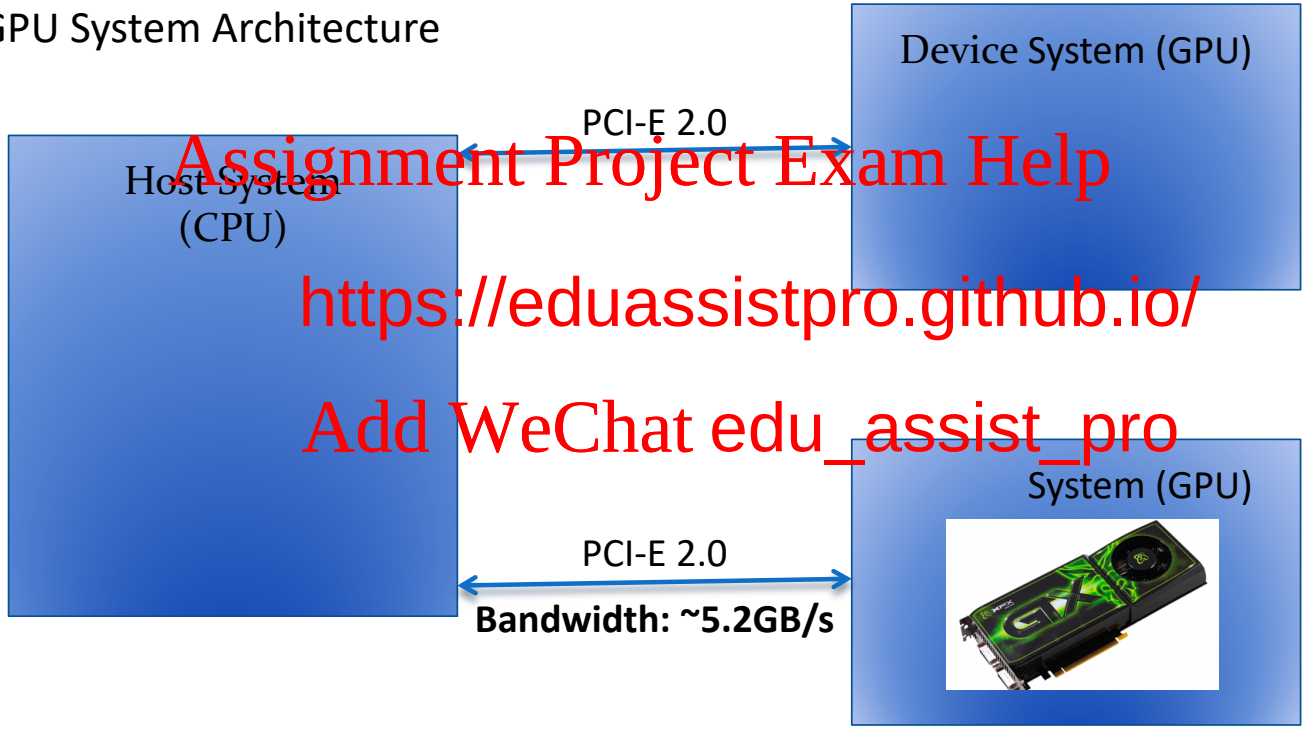
Section 1 - Multicore



- Writing fast code to exploit multicore parallelism
  - SIMD level parallelism
    ism

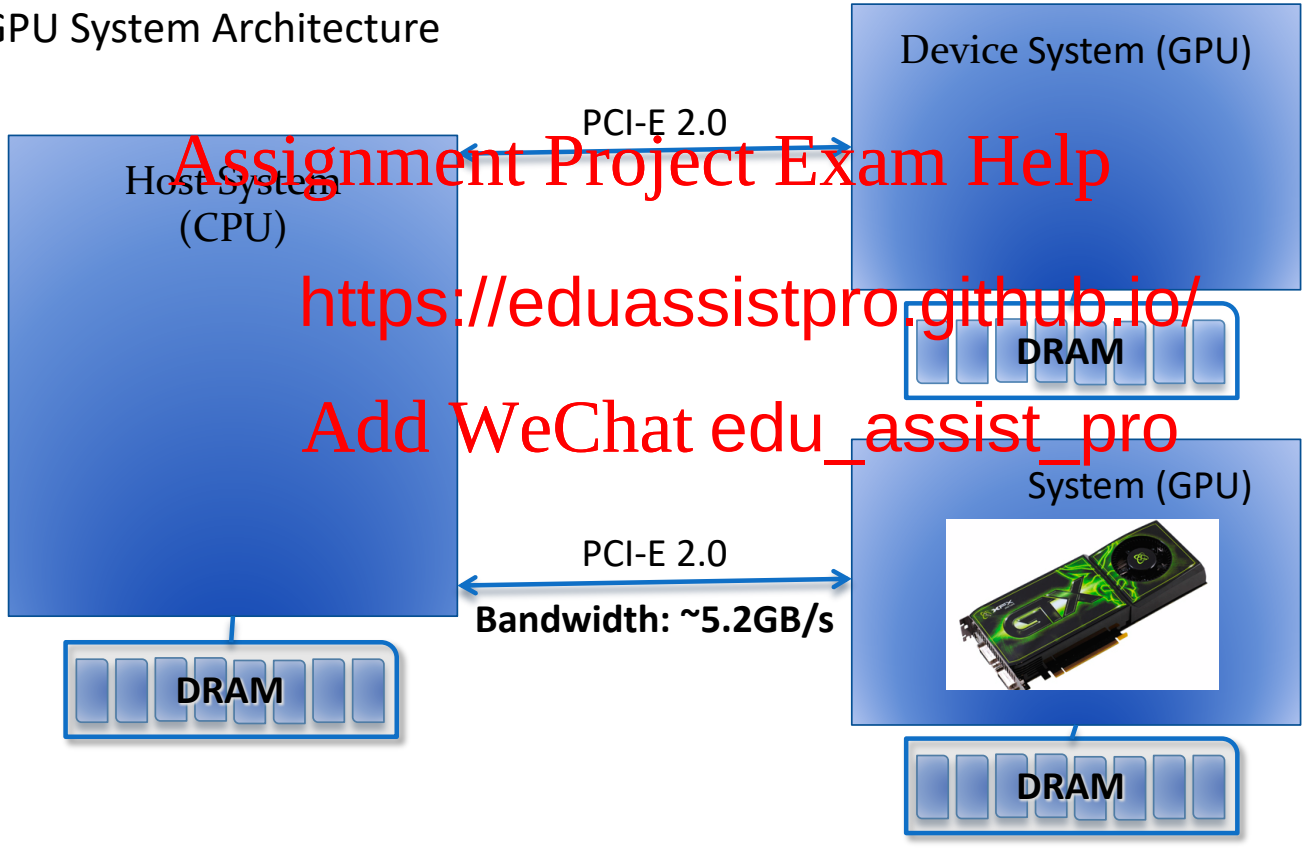
Section 2 - Manycore

- GPU System Architecture

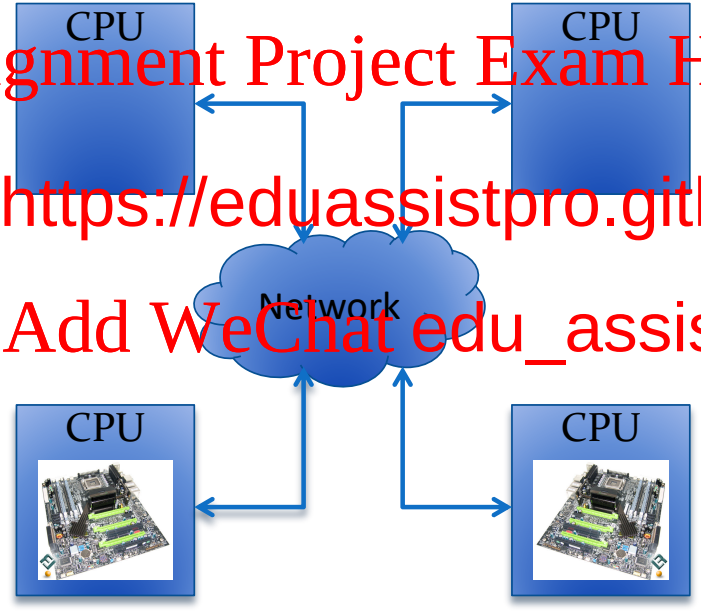


Section 2 - Manycore

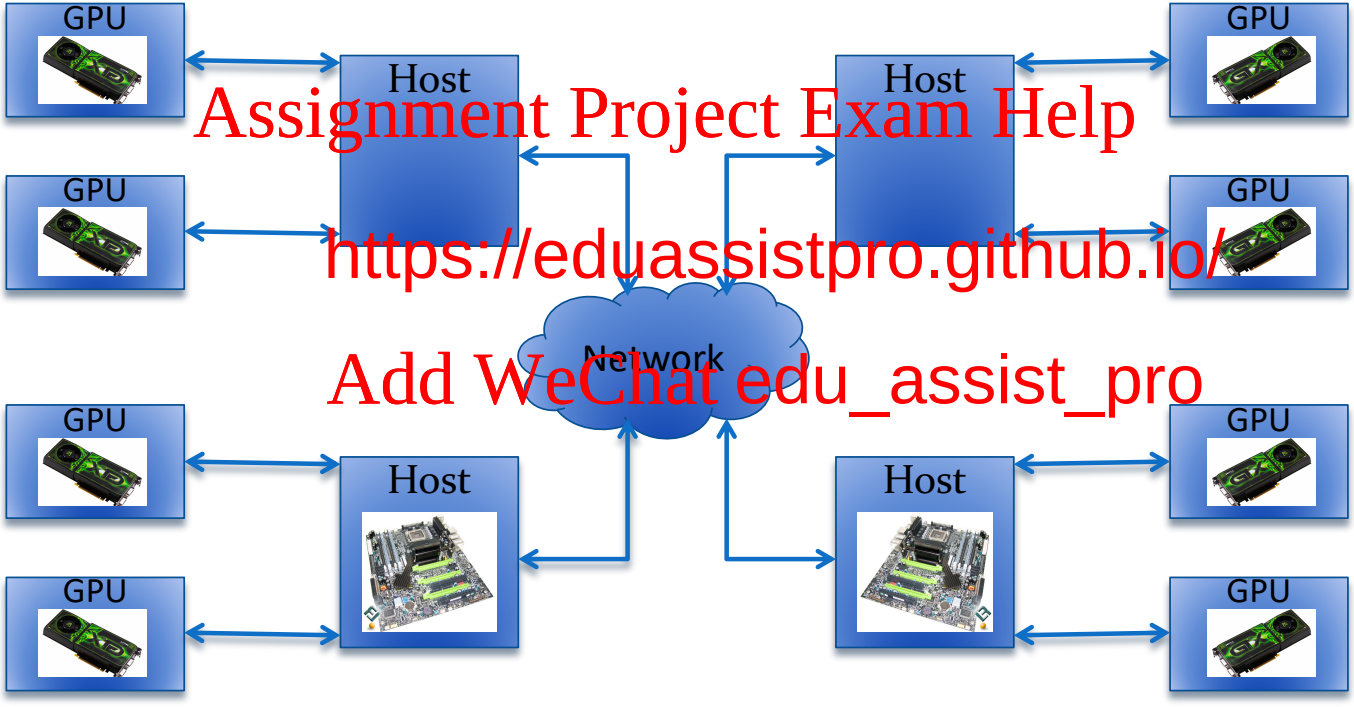
- GPU System Architecture



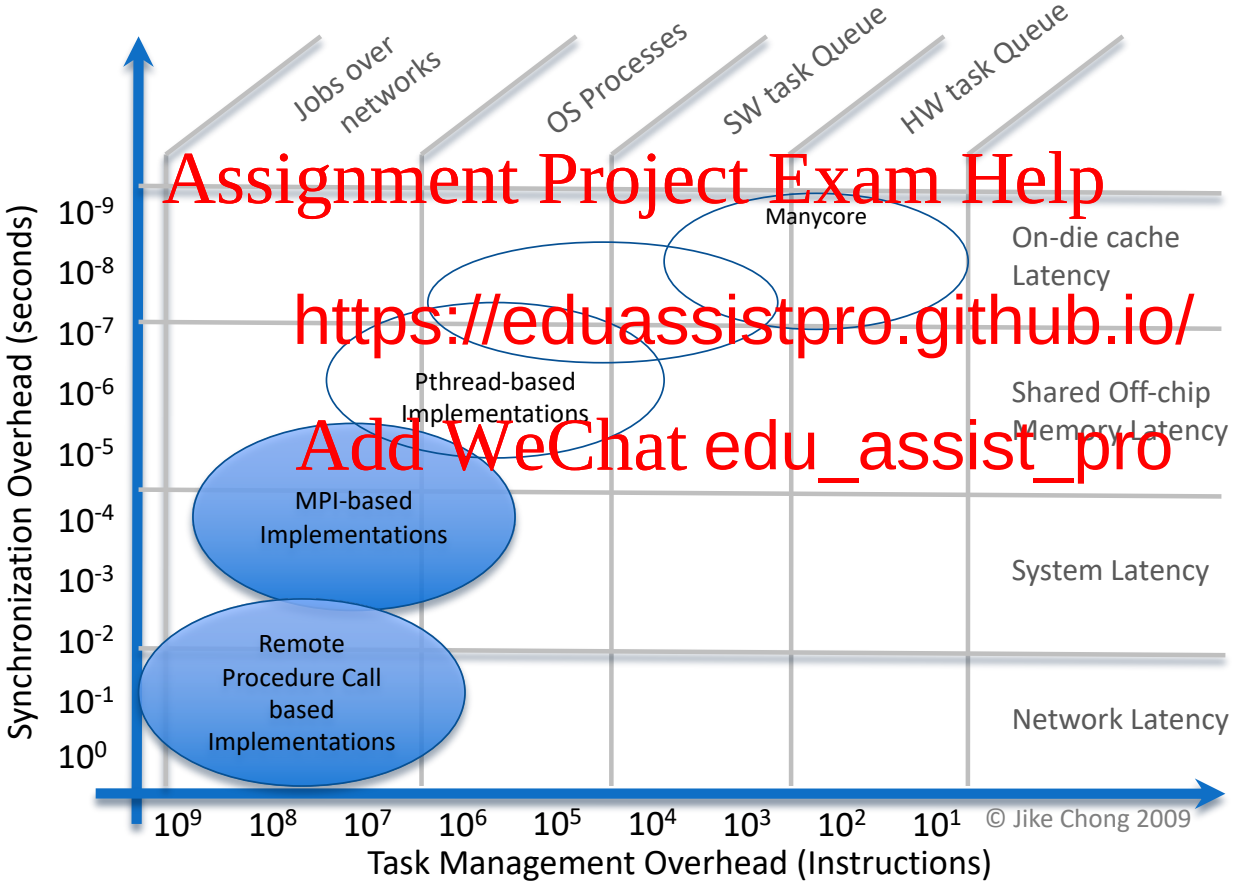
Section 3 – The Cloud



Section 3 – The Cloud



A Game of Granularity



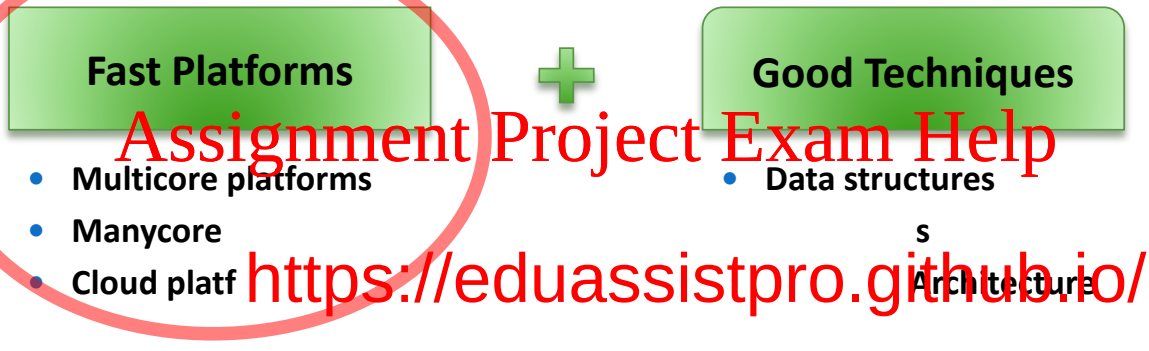
Outline

- Landscape of Computing Platforms
- Hardware Architectures
  1. Multicore vs Manycore
  2. Instruction lev
  3. SIMD
  4. Simultaneous multithreading
  5. Memory hierarchy
  6. System hierarchy
- How to Write Fast Code?

Outline

- Landscape of Computing Platforms
- Hardware Architectures
  - Multicore vs
  - Instruction le
  - SIMD
  - Simultaneous multithreading
  - Memory hierarchy
  - System hierarchy
- **How to Write Fast Code?**

How to Write Fast Code?



- We focus on the fast hardware in this l
- Combines with **good techniques** to produce fast code…

  …in order to solve a problem or improve an outcome