Lecture 19: Parallel architectures

Tuesday, March 13, 2018 10:46 AM

Outline

- Types of parallel architectures
- Taxonomy for parallel architectures
- Emerging parallel architectures

Announcements

- course evaluations
- lab 5 out
  - ↳ yes partner
  - ↳ **no** late submissions
- Final Wellman 126

Review friday during discussion

→ office hours Mon 4:30-6

Why parallelism?

Incr throughput w/ multiple tasks

Incr performance/ reduce latency

2 choices → double freq → h/w limits

or double cores → lower power

$\rightarrow P = \frac{1}{2} C V^2 f$

- Scalar → not parallel single inst stream + single data stream

Types of parallel architectures →

how to program?

how to communicate?

how are instructions executed?

☆- Accelerators (GPU) + CPU

→ send a task to accelerator

→ Send data to accel explicitly

→ offload computation

→ explicitly copy results back

☆Multi core

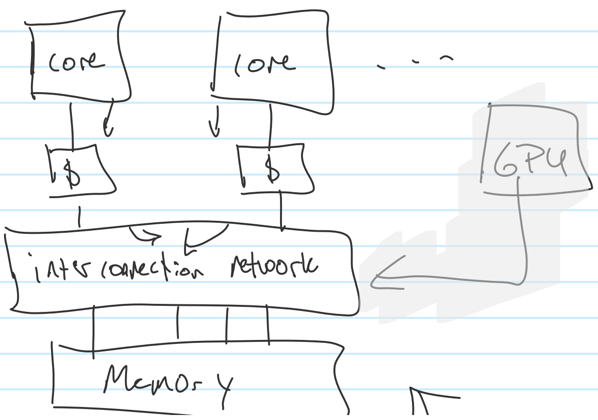
→ Shared memory

↳ because all processors see same data at same address

→ reading/ writing memory for comm.

→ programming?

threads → has a program counter



→ programming?

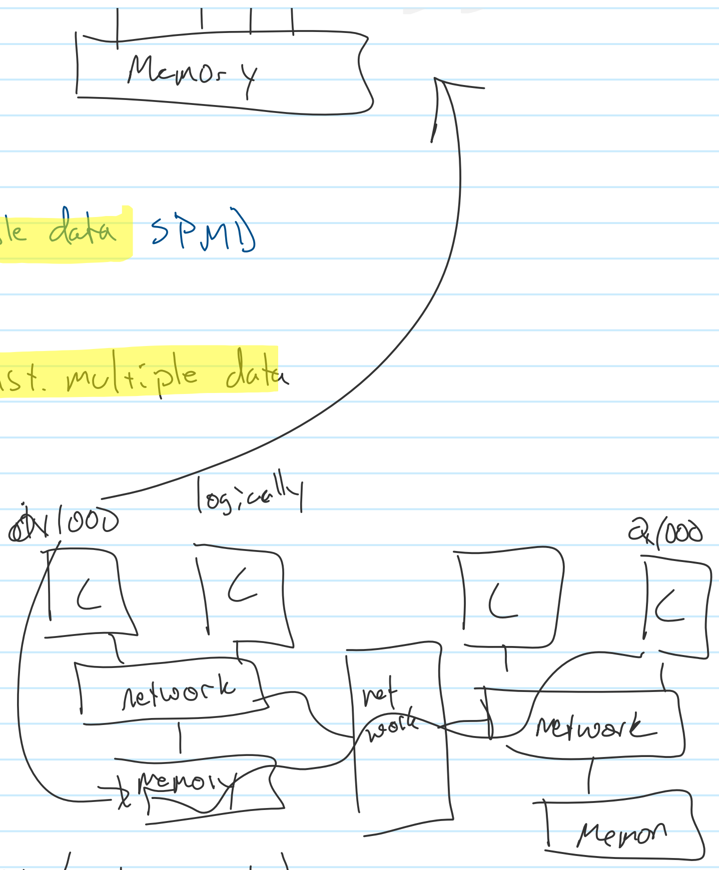
threads → has a program counter but in the same Virtual address space

usually executing same code

Single program multiple data SPMD

SPMD is a type of MIMD system

↳ multiple inst. multiple data



☆ Multi processor / multi-chip

has multiple processor "sockets"

Shared memory

- memory access time is non-uniform

- programmers must think about data locality

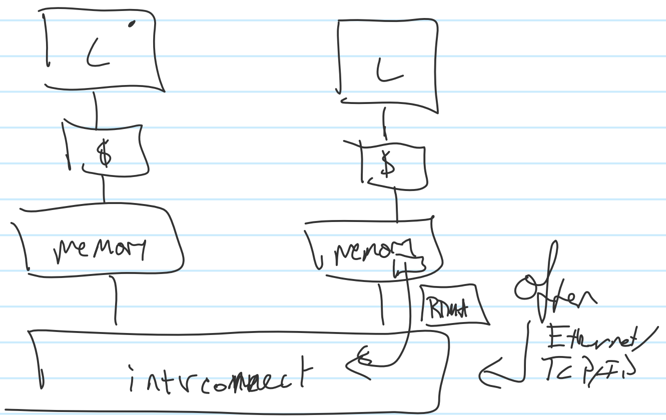
☆ Warehouse-scale computers (cloud / datacenter / super computer)

- communicate with message passing

request for data

respond w/ data

- remote DMA (Direct memory access)



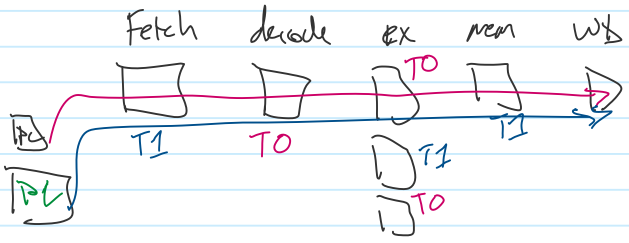
☆ Multi-threaded process (Intel hyperthreading)

4 cores / 8 threads

→ increases throughput at cost of latency

SPMD

shared memory



☆ - Vector architectures

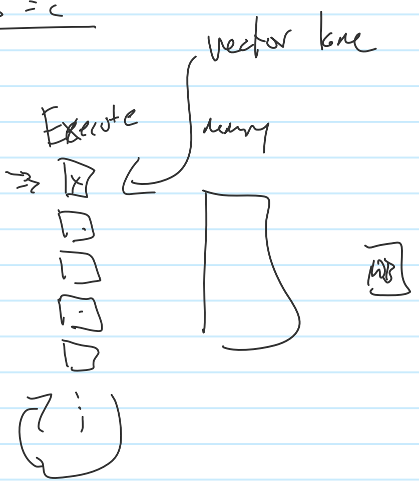
→ take advantage of data parallelism

$\vec{a} + \vec{b} = \vec{c}$

→ take advantage of data parallelism $\vec{a} + \vec{b} = \vec{c}$

for (i = 0 → n)
→ c[i] = a[i] + b[i] independent for each i

execute all vector elements in parallel



- incr throughput + perf
- easy to reason about

- fetch/decode shared
  decode once execute for all vector elements

- Single instruction stream multiple data streams
  SIMD

		Inst. Stream	
		Single	multiple
data streams	Single	SISD scalar	MISD ? not really...
	multiple	SIMD vectors/GPUs	MIMD SPMD threading shared memory message passing

Examples of parallel programs

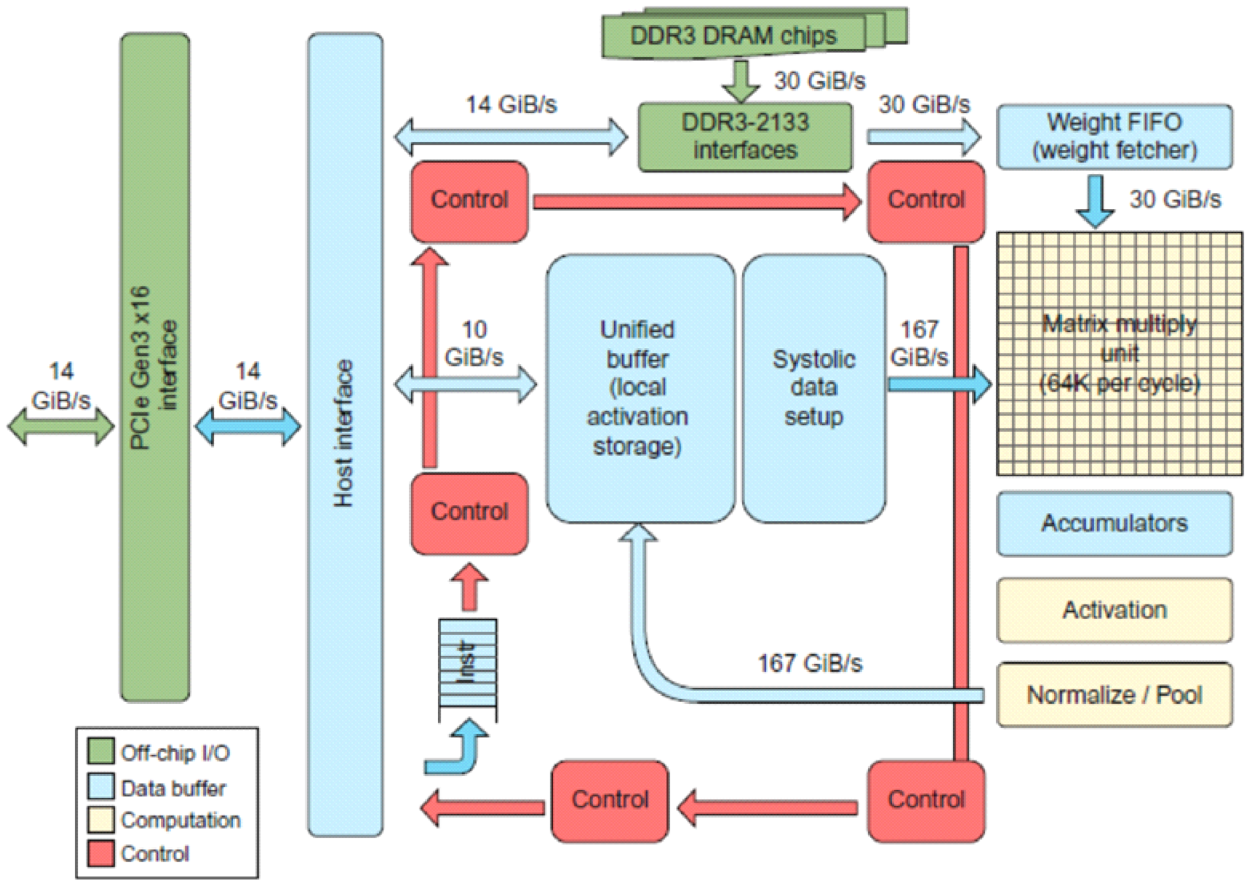
```
void daxpy(double *X, double *Y, double alpha, const int N)
{
    for (int i = 0; i < N; i++)
    {
        Y[i] = alpha * X[i] + Y[i];
    }
}
```

scalar

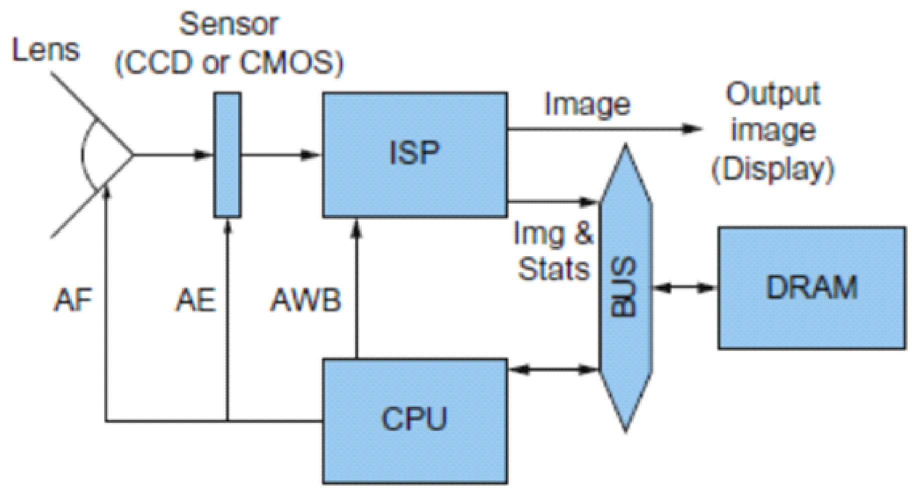
```
void daxpy_simd(double *X, double *Y, double alpha, const int N)
{
    __m256d *vx = (__m256d*)X;
    __m256d *vy = (__m256d*)Y;
    __m256d va = {alpha, alpha, alpha, alpha};

    for (int i = 0; i < N/4; i++)
    {
        __m256d mult = _mm256_mul_pd(vx[i], va);
        __m256d add = _mm256_add_pd(vy[i], mult);
        vy[i] = add;
    }
}

void daxpy_fmad(double *X, double *Y, double alpha, const int N)
{
    __m256d *vx = (__m256d*)X;
    __m256d *vy = (__m256d*)Y;
    __m256d va = {alpha, alpha, alpha, alpha};

    for (int i = 0; i < N/4; i++)
    {
        vy[i] = _mm256_fmadd_pd(va, vx[i], vy[i]);
    }
}

void daxpy_threads(double *X, double *Y, double alpha, const int N)
{
    int num_threads = std::thread::hardware_concurrency();

    std::thread threads[num_threads];

    int chunk = N / num_threads;

    for (int i=0; i<num_threads; i++) {
        threads[i] = std::thread(daxpy, X+i*chunk, Y+i*chunk, alpha, chunk);   ←
    }

    for (int i=0; i<num_threads; i++) {
        threads[i].join();
    }
}

void daxpy_omp(double *X, double *Y, double alpha, const int N)
{
    #pragma omp parallel for
    for (int i = 0; i < N; i++)           openMP
    {
        Y[i] = alpha * X[i] + Y[i];
    }
}

void daxpy_opencl(double *X, double *Y, double alpha, const int N)
{
    int tid = get_global_id(0);                                  GPU
    for (int i = tid; i < N; i += get_global_size(0)) {
        Y[i] = alpha * X[i] + Y[i];
    }
}
```

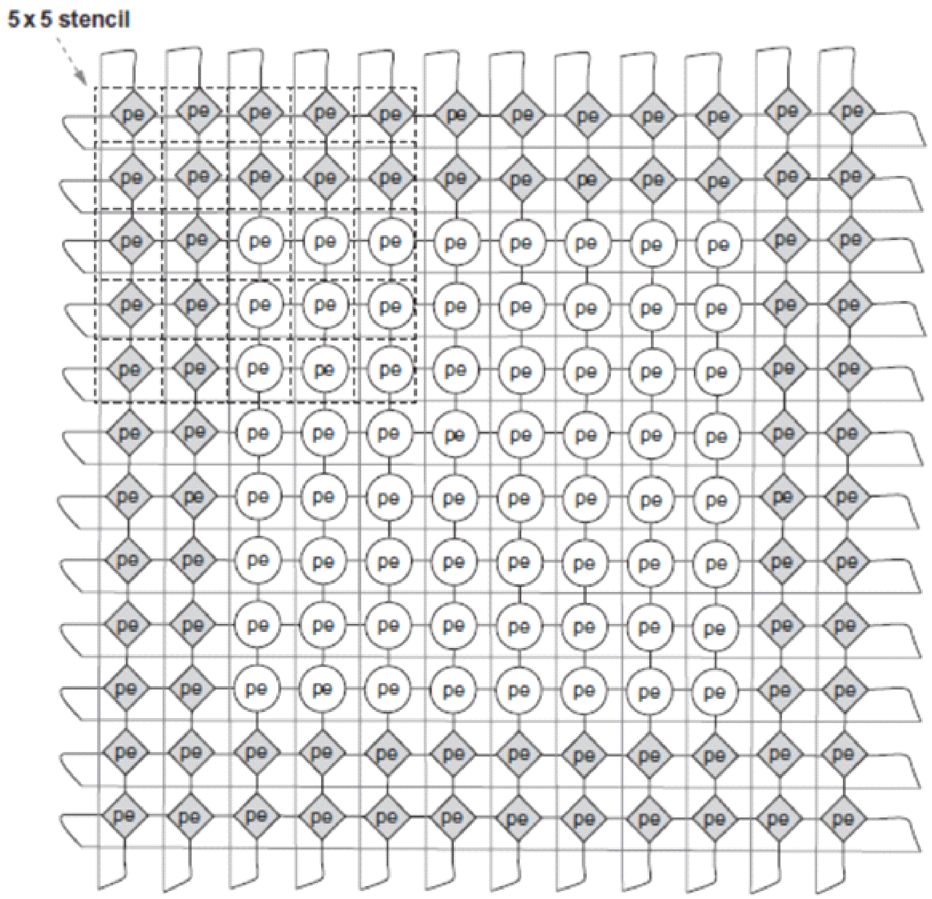
New parallel architectures

Google's TPU



Pixel visual core




5 x 5 stencil
pe