Tools Seminar

Week 10 - Parallel Computing

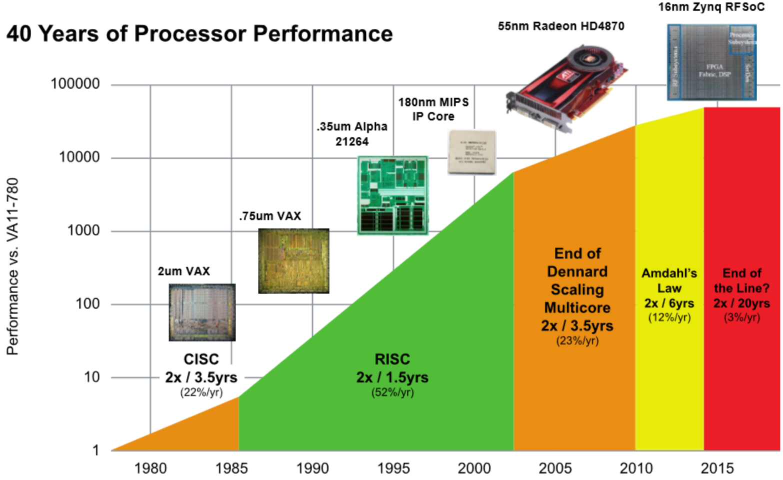
Hongzheng Chen

Mar 30, 2020

1

Introduction

Challenges: The End of Moore's Law and Scaling



Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e 2018

* That's why Intel is called "toothpaste factory" now

The End of Moore's Law and Scaling

This shift toward **increasing parallelism** *is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this plunge into parallelism is actually a* **retreat** *from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures.*

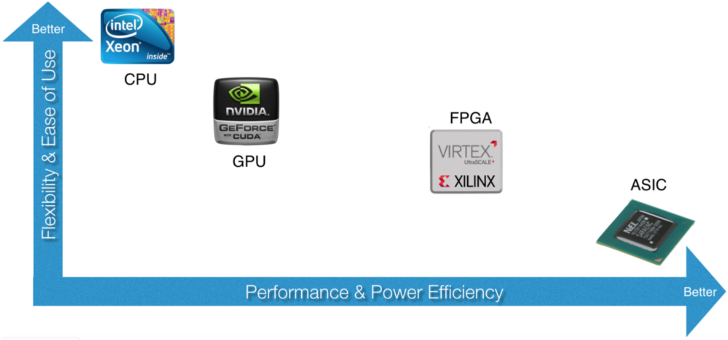
— The Landscape of Parallel Computing Research: A View from Berkeley, 2006

The End of Moore's Law and Scaling

> *This shift toward* **increasing parallelism** *is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this plunge into parallelism is actually a* **retreat** *from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures.*
> *— The Landscape of Parallel Computing Research: A View from Berkeley, 2006*

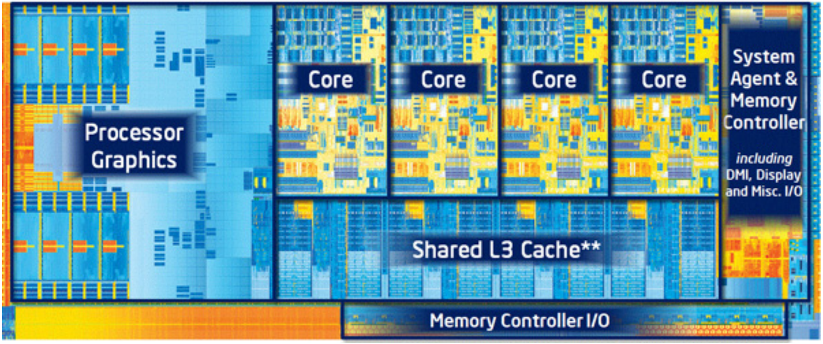
Thus, multicore processors put burdens from hardware to software, which needs programmers to code parallel programs.

Different hardware



- CPU
- GPU (Graphical Processing Unit)
- FPGA (Field-Programmable Gate Array)
- ASIC (Application-Specific Integrated Circuit)

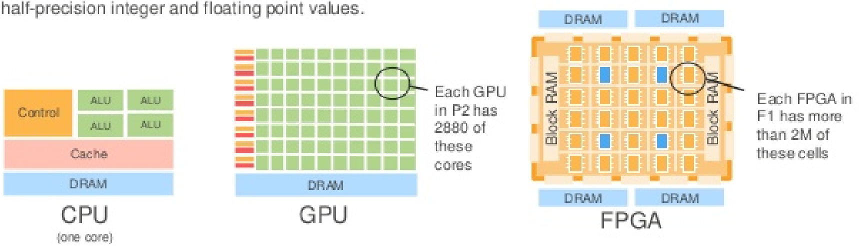
CPU Architecture



Intel core i7 CPU (Ivy Bridge)

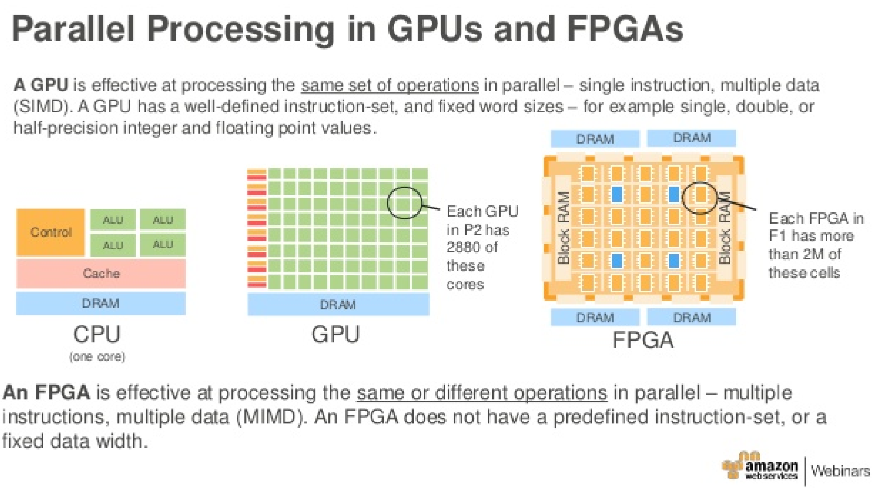
Parallel Processing in GPUs and FPGAs

A GPU is effective at processing the <u>same set of operations</u> in parallel – single instruction, multiple data (SIMD). A GPU has a well-defined instruction-set, and fixed word sizes – for example single, double, or half-precision integer and floating point values.



An FPGA is effective at processing the <u>same or different operations</u> in parallel – multiple instructions, multiple data (MIMD). An FPGA does not have a predefined instruction-set, or a fixed data width.

amazon web services | Webinars



We will focus on CPU parallelization in this seminar

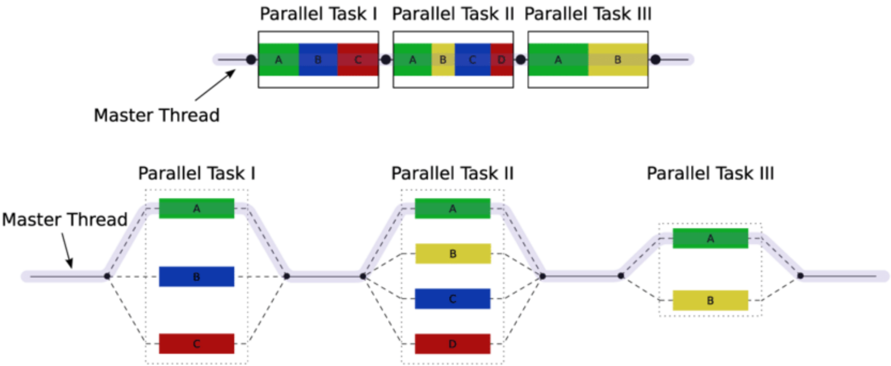
2

Single Machine Parallelization

2.1

Multi-threads

Fork-Join Model



- Fork: Dispatch tasks to each processor / thread
- Join: Synchronization, wait till all threads are done

pthread

POSIX (Portable Opearing System Interface for Unix)

- <pthread.h> is in Linux's system library and can be directly called

```
void *foo(void *arg)
{
    int* id = (int*) arg;
    printf("My id is %d\n", *id);
}

int main()
{
    pthread_t id[4];
    for (int i = 0; i < 4; ++i)
        // pass in function pointer and args
        pthread_create(&id[i], NULL,foo,&i);
    for (int i = 0; i < 4; ++i)
        pthread_join(&id[i], NULL);
    for (int i = 0; i < 4; ++i)
        pthread_exit(&id[i]);
}
```

Need to add `-lpthread` flag when compiling

¡thread¿

C++11 adds initial support for multi-threading in stl

```
#include <iostream>
#include <thread>
using namespace std;

void exec(int n){
    cout << "My id is" << n << endl;
}

int main(){
    thread myThread[4];
    for (int i = 0; i < 4; ++i)
        myThread = thread(exec,i);
    for (int i = 0; i < 4; ++i)
        myThread[i].join();
}
```

Race Condition

Be careful of the shared data

Thread A	Thread B
$\cdots$	$\cdots$
Count++	Count−−
$\cdots$	$\cdots$

Thread A		Thread B	
Load	Count	Load	Count
Add	#1	Sub	#1
Store	Count	Store	Count

- Critical section: That part of the program where the shared memory is accessed
- Need to avoid conflicts and make data consistent

Avoid Race Condition

Two basic methods:

- Corse-grained: Lock/mutex
- Fine-grained: Atomic operations

* There are lots of details about synchronization & consistency, please refer to books of OS

Mutex Operations in pthread

<pthread.h>

- `pthread_mutex_init(&mutex1,NULL)`
- `pthread_mutex_destroy(&mutex1)`
- `pthread_mutex_lock(&mutex1)`
- `pthread_mutex_unlock(&mutex1)`

<thread>

- `std::mutex g_display_mutex`
- `std::lock_guard<std::mutex> guard(g_display_mutex)`

2.2

OpenMP

OpenMP

OpenMP (Open Multi-Processing): Shared-memory programming model

- Set of parallel commands, library, and routines
- Simplify multi-threading programming
- A spec suitable for different devices from desktop to supercomputer
- gcc has initial support for OpenMP

OpenMP API

#include <omp.h> and only need to write compilation directives

```
#pragma omp <directive-name> [clause,...]
```

- omp_get_thread_num
- omp_get/set_num_procs
- omp_get/set_num_threads
- #pragma omp parallel for: The most commonly used!
- #pragma omp ... private (<variable list>)
- #pragma omp ... reduction (op:list)

OpenMP Example (Matrix Multiplication)

```
# pragma omp for
for ( i = 0; i < n; i++ )
{
  for ( j = 0; j < n; j++ )
  {
    c[i][j] = 0.0;
    for ( k = 0; k < n; k++ )
    {
      c[i][j] = c[i][j] + a[i][k] * b[k][j];
    }
  }
}
```

OpenMP Example (Summation)

```
float sum(const float *a, size_t n)
{
    float total = 0.;

    #pragma omp parallel for reduction(+:total)
    for (size_t i = 0; i < n; i++) {
        total += a[i];
    }
    return total;
}
```

2.3

Cilk

Intel Clik Plus

Intel Cilk Plus: A extremely light-weighted parallel framework

- `#include<cilk/cilk.h>`
- gcc 5.0+: `g++ -O3 -fcilkplus -lcilkrts <source>`
- Or compiled by Intel Compiler (icpc) — Better choice!
    - But from icpc 18.0, Intel uses Thread Building Block (TBB)

Only three keywords

- `cilk_spawn`: fork
- `cilk_sync`: join
- `cilk_for`: `parallel for`

Clik Example (Fibbonacci)

```
int fib(int n)
{
    if (n < 2)
        return n;
    int x = fib(n-1);
    int y = fib(n-2);
    return x + y;
}

int fib(int n)
{
    if (n < 2)
        return n;
    int x = cilk_spawn fib(n-1);
    int y = fib(n-2);
    cilk_sync;
    return x + y;
}
```

Cilk Runtime

The most powerful thing is Cilk runtime deploys **work-stealing** scheduling strategy, which greatly outperforms OpenMP's runtime

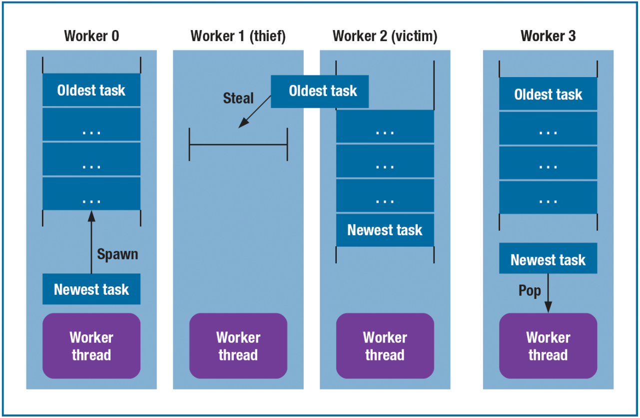


Fig source: Intel TBB

2.4

SIMD

ILP

- Pipelining
- Hyperscalar
- Very Long Instruction Word (VLIW)
- Vector processing
- Out-of-Order (OoO) execution
- Spectacular execution

3

Distributed Parallelization

MPI

4

Parallel Computing Frameworks

Frameworks

- MapReduce
- Spark
- Ray

5

Summary

Summary

- Introduction
- Shared-memory: `<pthreads>`, OpenMP, Cilk, AVX
- Distributed-memory: MPI
- Parallel computing frameworks: MapReduce