浅聊并发

创研技术分享论坛

易居创研中心

2017年05月

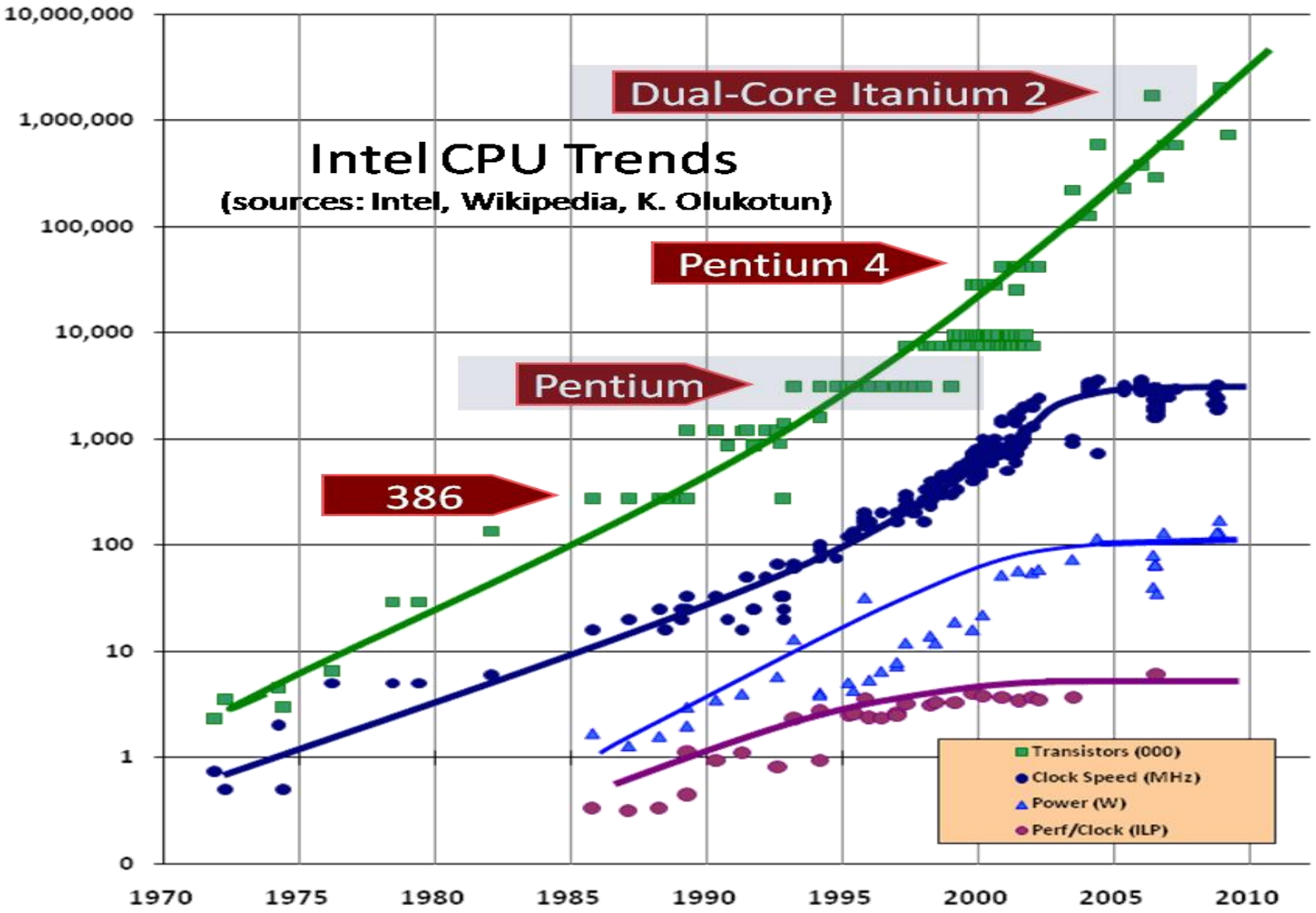
目 录

CONTENTS

引言

- 软件 VS. 硬件
- 硬件（CPU）结构发生变化
- 自 OOP 之后又一次重大“**revolution**”

转折点



转折点

之前:

– clock speed
– execution optimization
– cache

之后:

– hyperthreading
– multicore
– cache

转折点

- more cores in a DIE
- more DIEs in a server
- more servers for a system
- systems are getting more complicated

并发 VS. 并行

“In programming, concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations. Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.” - Rob Pike

并发 VS. 并行

- 并发是一种构建软件的方式，是一种组织软件的方式；
- 并行是指同时执行；
- 两者相关但不是一回事情；

The gopher example from Rob Pike:

https://talks.golang.org/2012/waza.slide#12

并发 VS. 并行

- A complex problem can be broken down into easy-to-understand components.
- The pieces can be composed concurrently.
- The result is easy to understand, efficient, scalable, and correct.
- Maybe even parallel.

几种并发模型

- Lock based concurrency
- CSP
- Actor
- Asynchronous callback/promise

Locked based concurrency

- 共享可变状态;
- Low-level primitives, such as lock.

- 锁太大或太多会造成性能问题;
- 锁太少会不能保护 critical section;
- 锁的顺序不对导致死锁;

Communicating sequential processes

- golang
- do not communicate by sharing memory ; instead, share memory by communicating
- Go primitives: goroutines, channels, and the select statement.

Goroutines

- Goroutines are like lightweight threads;
- They start with tiny stacks and resize as needed;
- Go programs can have hundreds of thousands of them;
- Start a goroutine using the go statement:

  go f(args);
- The Go runtime schedules goroutines onto OS threads;
- Blocked goroutines don't use a thread;

Channels

Channels provide communication between goroutines.

```
1  c := make(chan string)
2
3  // goroutine 1
4  c <- "hello!"
5
6  // goroutine 2
7  s := <-c
8  fmt.Println(s) // "hello!"
9
```

Select

A select statement blocks until communication can proceed.

```
1 select {
2 case n := <-in:
3   fmt.Println("received", n)
4 case out <- v:
5   fmt.Println("sent", v)
6 }
7
```

Actor - Erlang

- everything is a process.
- process are strongly isolated.
- process creation and destruction is a lightweight operation.
- message passing is the only way for processes to interact.
- processes have unique names.
- if you know the name of a process you can send it a message.
- processes share no resources.
- error handling is non-local.
- processes do what they are supposed to do or fail.

-- Joe Armstrong

afile_server.erl

```erlang
-module(afile_server).
-export([start/1, loop/1]).

start(Dir) -> spawn(afile_server, loop, [Dir]).

loop(Dir) ->
    receive
        {Client, list_dir} ->
            Client ! {self(), file:list_dir(Dir)};
        {Client, {get_file, File}} ->
            Full = filename:join(Dir, File),
            Client ! {self(), file:read_file(Full)}
    end,
loop(Dir).
```

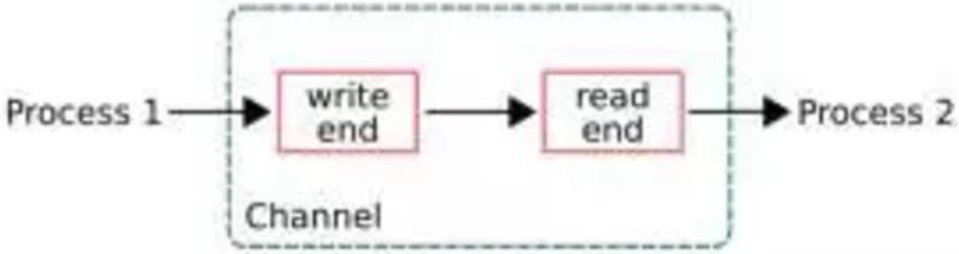
afile_client.erl

```erlang
-module(afile_client).
-export([ls/1, get_file/2]).

ls(Server) ->
    Server ! {self(), list_dir},
    receive
        {Server, List} ->
            List
    end.

get_file(Server, File) ->
    Server ! {self(), {get_file, File}},
    receive
        {Server, Content} ->
            Content
    end.
```

Eshell 模拟调用

```
Erlang/OTP 17 [erts-6.0] [64-bit] [smp:4:4] [async-threads:10]

Eshell V6.0  (abort with ^G)
1> c(afile_server).
{ok,afile_server}
2> c(afile_client).
{ok,afile_client}
3> Server = afile_server:start(".").
<0.44.0>
4> afile_client:ls(Server).
{ok,["afile_client.beam","afile_client.erl",
     "afile_server.beam","afile_server.erl","Hello.txt",
     "include","lib","min.beam","min.erl","reverse.beam",
     "reverse.erl"]}
5> afile_client:get_file(Server, "Hello.txt").
{ok,<<"Hello,Erlang.">>}
```

CSP VS. Actor



CSP

- Communication through channels
- Processes are "anonymous"

Actor Model

- Point-to-point communication
- No anonymity

Javascript/Node

- 单线程;
- 异步I/O;
- 事件驱动;

Javascript/Node

callback hell

```
const data = {...}
AwesomeDBAdapter.open(config, (err, db) => {
  if (err) error-handling...
  db.getOrCreateTable((err, table) => {
    if (err) error-handling...
    table.insert(data, (err, result) => {
      if (err) error-handling...
      // actual code
    })
  })
})
```

Javascript/Node

```
// promise
AwesomeDBAdapter.open(config).then(db => {
  return db.getOrCreateTable()
}).then(table => {
  return table.insert(data)
}).then(result => {
  // actual code
}).catch(err) {
  // error handling
}
```

- 可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数；
- Promise对象提供统一的接口，使得控制异步操作更加容易

其他几种

- STM(Software Transactional Memory)
- Observable

引用

- The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software
- Concurrency is not Parallelism
- https://www.youtube.com/watch?v=cN_DpYBzKso
- An Introduction to Programming with Threads.
- Javascript异步编程的4种方法
- Promise: 给我一个承诺，我还你一个承诺
- ECMAScript 6 Promise 对象
- 部落图鉴之Go：爹好还这么努力？
- Erlang程序设计（第2版）

Thank You !