Easy Parallel Computing in Perl5 for Multi-Core CPUs

Darko Obradovic

My history:

15 years of linear programming in Perl5

April 2007:

First dual-core CPU

October 2012:

Xeon E5

(4x3,2GHz, 4x3MB Cache, 8 logical cores)

+ 64GB of RAM

since then:

linear programming

= bad conscience

August 2018:

32-core CPU?

... finally made me thinking about parallel programming

A search for "parallel"
on CPAN returns 727 results
modules for different purposes

parallel

distributed

async

master/slave

parallel
simultaneous calculation
usually splits calculations to different processes

typically uses forking

`Parallel::ForkManager`

async

non-blocking execution

usually to overcome I/O latency

and for event-driven programming

typically uses callbacks

`LWP::Parallel`

master/slave

parallel server implementation

usually to handle many requests,
event-driven dispatching

typically uses workers/preforking

`Parallel::Prefork`
`Parallel::ScoreBoard`

distributed

simultaneous distributed calculation

distributes larger chunks of work to different machines

typically uses MapReduce

`Parallel::MapReduce`

parallel
simultaneous calculation
usually splits calculations to different processes

typically uses forking

`Parallel::ForkManager`

processes *vs* **threads**

process
program + data + resources

can be forked
= clone of the process

very robust
no shared address space

UNIX thread

"lightweight" process

shared address space & resources

→ more efficient, more fragile

create, join, yield, ...

perl thread

no shared address space

so how to start in Perl5?

TIMTOWTDI

CPAN modules suitable for simultaneous calculation on Multi-Core CPUs

subs::parallel
Parallel::Loops
Parallel::Simple::Dynamic
Parallel::ForkManager
Parallel::Simple
Parallel::Iterator
Parallel::parallel_map
Parallel::Runner
Parallel::QueueWorker
Parallel::Queue
Parallel::Jobs
Parallel::Forker

Proc::Parallel
Parallel::DataPipe
Parallel::Workers
Job::Manager
Parallel::SubFork
Parallel::SubArray
Parallel::MapReduce
Parallel::Supervisor
Proc::ParallelLoop
Parallel::Fork::BossWorker
Parallel::ForkControl
Parallel::ThreadContext

laziness...
we want to keep it simple!
no process/thread management
or synchronization

“get it parallelized in a minute!”

Parallel::Simple

demo example

Parallel::Simple

- no return values
- changes in memory are lost

  → only parts not changing state
- makes sense with 2+ cores

`Parallel::parallel_map`

demo example

Parallel::parallel_map

- only maps without side-effects
- efficiently with N forks

  → from %ENV or /proc/cpuinfo
- merges return values

subs::parallel

demo example

subs::parallel

- one fork per sub call
- waits when return values are used
- based on Perl threads
- seems to have issues

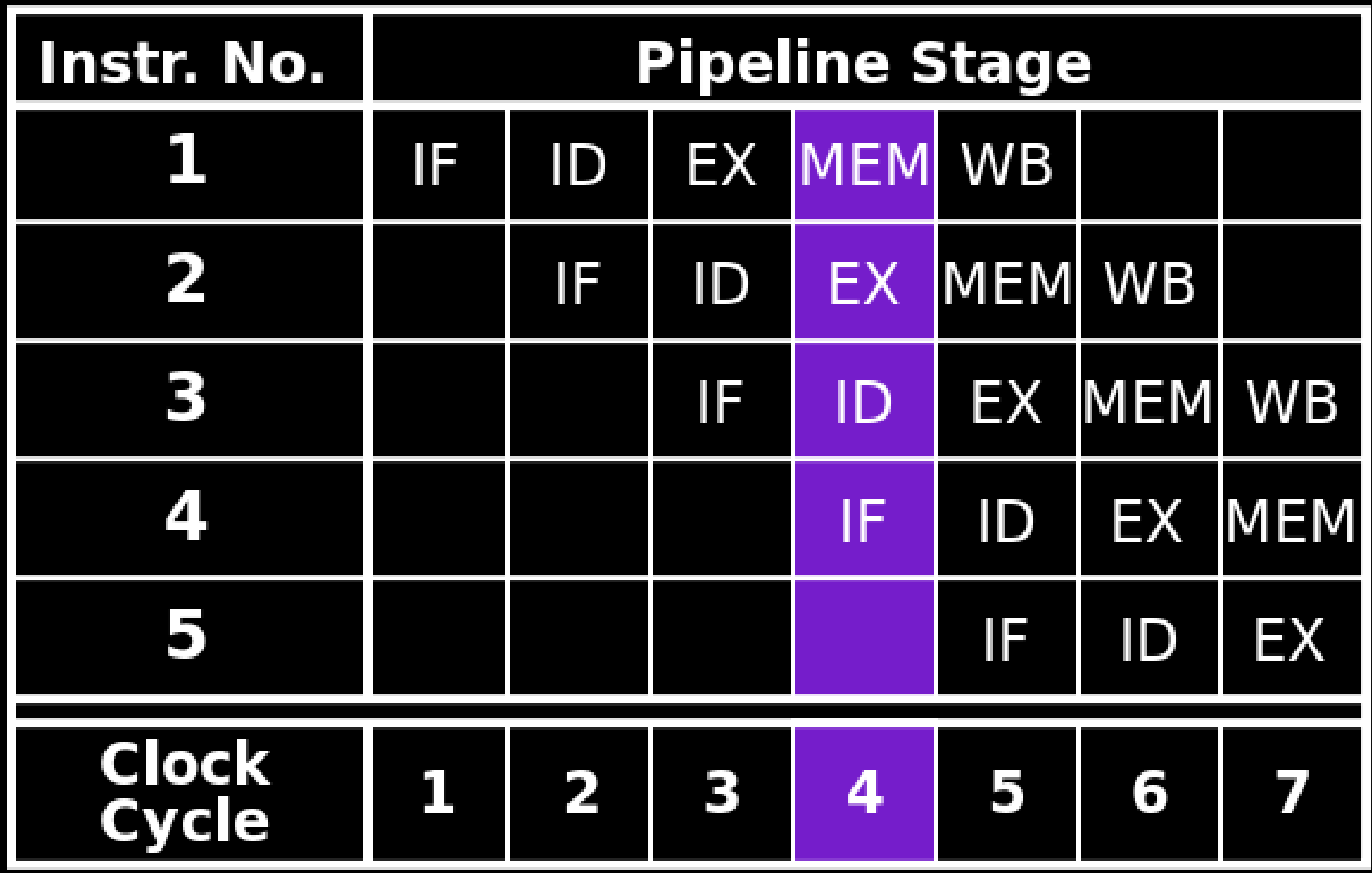
on my Xeon:

4 processes = speedup 3

what about logical cores?
(Intel's *HyperThreading*)

the problem:

CPU pipelining


Instr. No.
Pipeline Stage
1
IF ID EX MEM WB
2
IF ID EX MEM WB
3
IF ID EX MEM WB
4
IF ID EX MEM
5
IF ID EX
Clock Cycle
1 2 3 4 5 6 7

the issue:

pipeline is partly empty

Intel PIII:

11 pipeline stages

Intel P4:

20 pipeline stages

the solution:

share a physical pipeline for two logical threads

on my Xeon:

4 processes = speedup 3

8 processes = speedup 4

... so parallelizing is very simple in Perl5 for some problems

… so use it and enjoy the extra free time ;-)