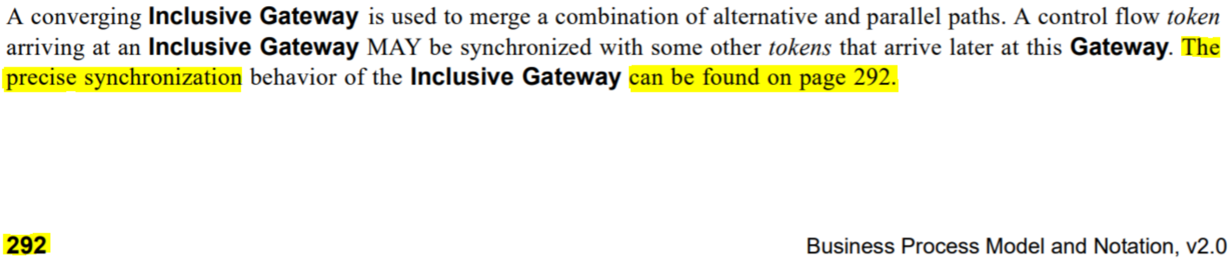
Immigration Course
on
Formal Methods

Academic year 2023/2024

Emilio Tuosto
https://cs.gssi.it/emilio.tuosto/

A couple of resasons to be rigorous

[https://www.omg.org/spec/BPMN/2.0/]

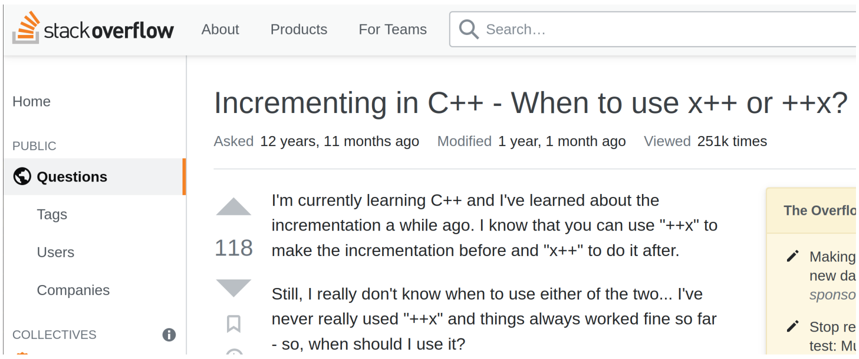


A converging **Inclusive Gateway** is used to merge a combination of alternative and parallel paths. A control flow *token* arriving at an **Inclusive Gateway** MAY be synchronized with some other *tokens* that arrive later at this **Gateway**. The precise synchronization behavior of the **Inclusive Gateway** can be found on page 292.

292

Business Process Model and Notation, v2.0

[https://stackoverflow.com/questions/1812990/incrementing-in-c-when-to-use-x-or-x]



Incrementing in C++ - When to use x++ or ++x?

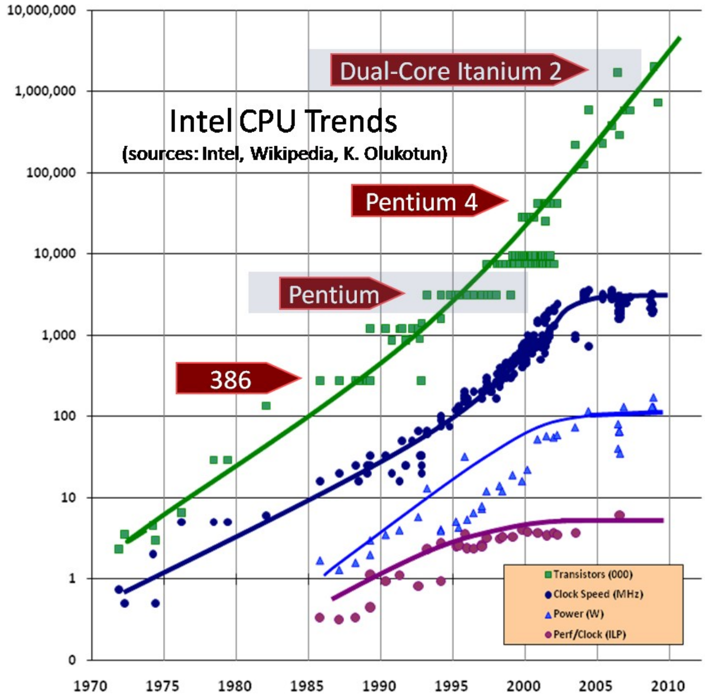
Asked 12 years, 11 months ago Modified 1 year, 1 month ago Viewed 251k times

I'm currently learning C++ and I've learned about the incrementation a while ago. I know that you can use "++x" to make the incrementation before and "x++" to do it after.

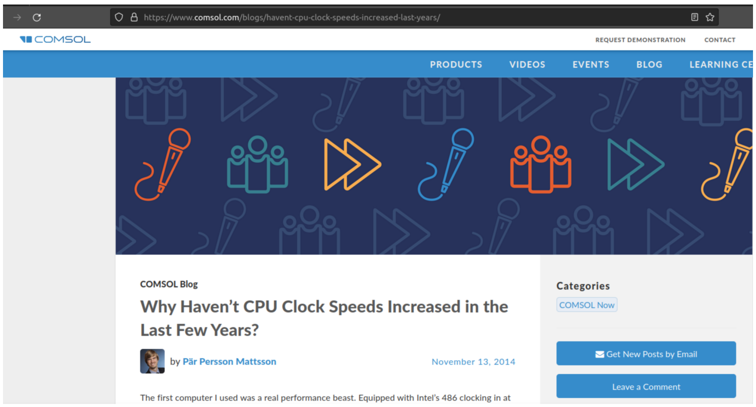
Still, I really don't know when to use either of the two... I've never really used "++x" and things always worked fine so far - so, when should I use it?

A reson to go concurrent

[https://i.extremetech.com/imagery/content-types/03zc6ghfKswe41smvPXi8Zh/images-6.jpg]



[https://www.comsol.com/blogs/havent-cpu-clock-speeds-increased-last-years]



COMSOL Blog

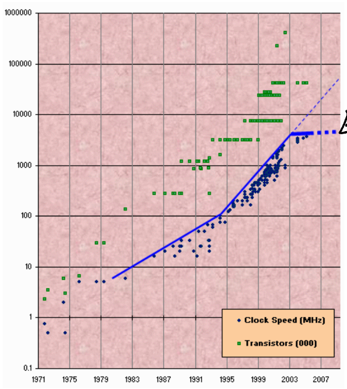
Why Haven’t CPU Clock Speeds Increased in the Last Few Years?

by Pär Persson Mattsson November 13, 2014

The first computer I used was a real performance beast. Equipped with Intel’s 486 clocking in at

The art _ multi-processor programming

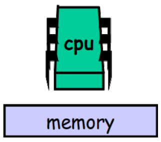
Hw $\xrightarrow{\text{Efficiency is no longer on hw thing}}$ Sw



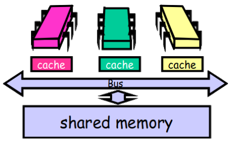
clock Speed

transistors grows by a factor of 10 every 10 years

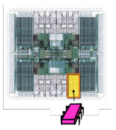
CPU speed is plotooing



uniprocessor



shared memory processor



multicore

- programming constructs in ALL languages
  - "new" languages
    - Go
    - Scala
    - Elixir / Erlang
    - Ballerina
    - Concurnas
- supporting library, AKKA
- Modelling languages
  - BPEL
  - BPMN

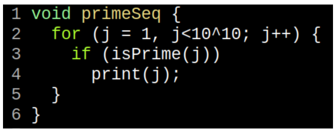
Job interviews and prime numbers

"On the first day of your new job, your boss asks you to find all primes between 1 and 10^10 (never mind why), using a parallel machine that supports ten concurrent threads. This machine is rented by the minute, so the longer your program takes, the more it costs. You want to make a good impression. What do you do?"

[Herlihy, Shavit: The Art of Multiprocessor Programming. Elsevier, 2012.]

An example of shared memory concurrency

Print all prime integer between 1 & 10^{10}

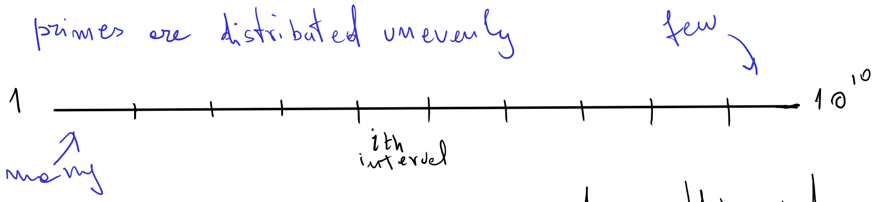


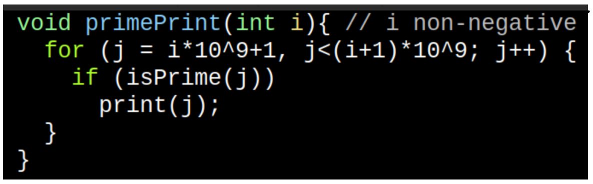
```
void primeSeq {
  for (j = 1, j<10^10; j++) {
    if (isPrime(j))
      print(j);
  }
}
```

Now let's try concurrently

Split the interval & lauch a thread on each position

primes are distributed unevenly

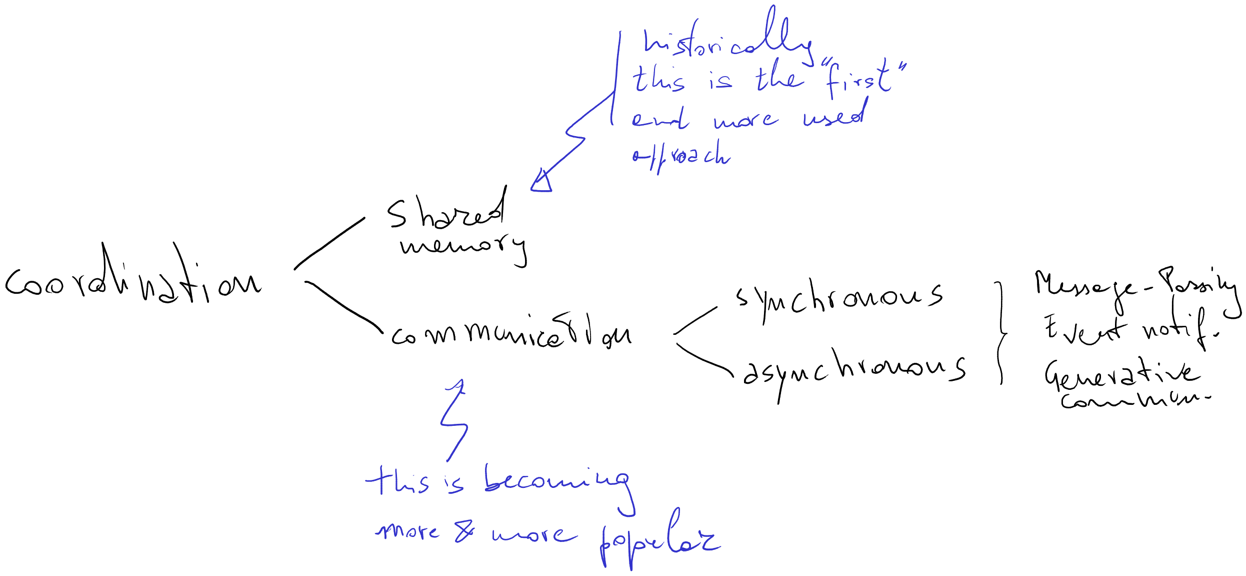




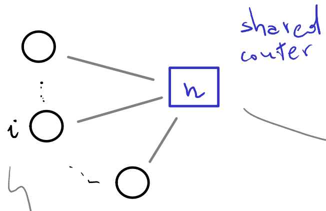
```
void primePrint(int i){ // i non-negative
  for (j = i*10^9+1, j<(i+1)*10^9; j++) {
    if (isPrime(j))
      print(j);
  }
}
```

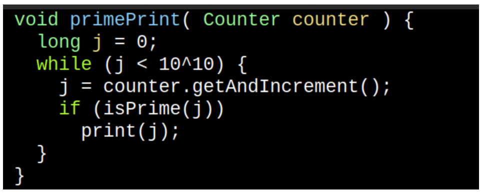
How good is this idea?

- Uneven load
- Is there an "optimal" split?


coordination
Shared memory
communication
synchronous
asynchronous
Message-Passing
Event notif.
Generative commun.
historically this is the "first" and more used approach
this is becoming more & more popular

Exercise 0 Find a better multi-threaded program for the primality test



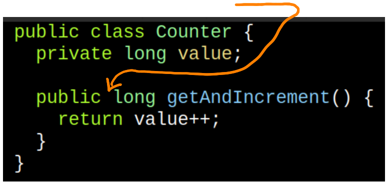


```
void primePrint( Counter counter ) {
  long j = 0;
  while (j < 10^10) {
    j = counter.getAndIncrement();
    if (isPrime(j))
      print(j);
  }
}
```

THIS IS NOT GOOD!

RACES

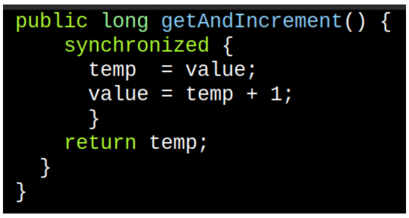
synchronised!



```
public class Counter {
  private long value;

  public long getAndIncrement() {
    return value++;
  }
}
```

temp := value
value ++
return temp



```
public long getAndIncrement() {
    synchronized {
      temp  = value;
      value = temp + 1;
      }
    return temp;
  }
}
```

even better

WHY?

REFLECT about why this solution is better than splitting

Some terminology

Concurrency vs Parallelism

compose "independent" stuff

~

deal with a lot of stuff
AT ONCE

GOAL: "good" composition

run stuff symoltaneously

~

do a lot of stuff
AT ONCE

GOAL: "good" execution

DESIGN!

PERFORMANCE

break down problems
&
Compose the pieces

A Choreographic Formal Model of Communicating Systems

—Immigration Course on Formal Methods—

Emilio Tuosto @ GSSI

So far...

- An idea of FMs

Leonardo da Vinci

> “ Ma prima farò alcuna esperienza avanti ch'io più oltre proceda, perché mia intenzione è allegare prima l'esperienzia e poi colla ragione dimostrare. ”

eM's (bad) translation

> “ Before proceeding further, I will first get some experiment, because my intention is to first understand the experiment and then to explain it with the intellect. ”

- Concurrency vs Parallelism
- Shared-memory

Message-passing

Pink Floyd

“Is there anybody out there?”

A glimpse of Erlang

```erlang
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);


pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

Semantics

- Message passing
- FIFO buffers ⟦mailboxes in Erlang's jargon⟧
- Spawn of threads

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);


pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

Semantics

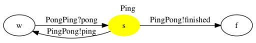
- Message passing
- FIFO buffers ⟦mailboxes in Erlang's jargon⟧
- Spawn of threads

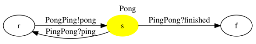
Asynchrony by design

Erlang is an embodiment of the well-known actor model of Hewitt and Agha...dates back to '73!

Friendlier representations

Local behaviour: communicating machines

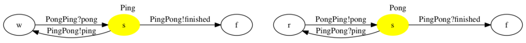




CFSMs (Brand & Zafiropulo 1983!): FIFO buffers as well

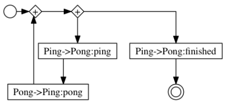
Friendlier representations

Local behaviour: communicating machines



CFSMs (Brand & Zafiropulo 1983!): FIFO buffers as well

Choregraphy: global graph



…“synchronous” distributed workflow (Deniélou and Yoshida 2012)

A glimpse of Erlang

```erlang
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);
```

```erlang
pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);


pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

Q:

Is this program correct?

A glimpse of Erlang

```
ping(N, Pong_PID) ->
  Pong_PID ! {ping, self()},
  receive
    pong ->
      io:format("Ping received pong~n", [])
  end,
  ping(N - 1, Pong_PID).

ping(0, Pong_PID) ->
  Pong_PID ! finished,
  io:format("ping finished~n", []);


pong() ->
  receive
    finished ->
      io:format("Pong finished~n", []);
    {ping, Ping_PID} ->
      io:format("Pong received ping~n", []),
      Ping_PID ! pong,
      pong()
  end.
```

Q:

Is this program correct?

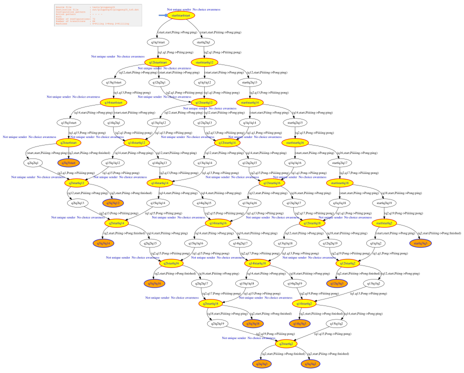
A:

No!

Exercise:

find the bug

Send ping-pong to shell !!! ... I mean, use ChoSyn



```
Source file             : tests/pingpong35
Destination file        : out/pingpong35/pingpong35_ts9.dot
Configuration pattern   :
Action pattern          : - - - -
Path                    :
Number of configurations: 72
Number of transitions   : 84
Machines                : 0=Piiing 1=Pong 2=Piiiiing
```

Not unique sender No choice awareness
start|start|start
(start,start,Piiing->Pong:ping)
(start,start,Piiiiing->Pong:ping)
q1|q1|start
start|q2|q1
(q1,q1,Pong->Piiing:pong)
(q2,q1,Pong->Piiiiing:pong)
q12|start|start
start|start|q12
(q12,start,Piiing->Pong:ping)
(start,start,Piiiiing->Pong:ping)
q13|q1|start
q12|q2|q1
q1|q1|q12
start|q2|q13
(q1,q13,Pong->Piiing:pong)
(q2,q1,Pong->Piiiiing:pong)
(q1,q1,Pong->Piiing:pong)
q14|start|start
q12|start|q12
start|start|q14
(q14,start,Piiing->Pong:ping)
(q12,start,Piiing->Pong:ping)
(q12,start,Piiiiing->Pong:ping)
q15|q1|start
q14|q2|q1
q13|q1|q12
q12|q2|q13
q1|q1|q14
(q1,q15,Pong->Piiing:pong)
(q2,q13,Pong->Piiiiing:pong)
q2|start|start
q14|start|q12
q12|start|q14
(q2,start,Piiing->Pong:finished)
(q14,start,Piiing->Pong:ping)
(q14,start,Piiiiing->Pong:ping)
q2|q2|q1
q3|q3|start
q15|q1|q12
q14|q2|q13
q13|q1|q14
q12|q2|q15
(q2,q15,Pong->Piiiiing:pong)

- Brand, D. and Zafiropulo, P. (1983).
  On Communicating Finite-State Machines.
  JACM, 30(2):323–342.

- Guanciale, R. and Tuosto, E. (2016).
  An abstract semantics of the global view of choreographies.
  In *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, pages 67–82.

- Tuosto, E. and Guanciale, R. (2018).
  Semantics of global view of choreographies.
  Journal of Logic and Algebraic Methods in Programming, 95:17–40.
  Revised and extended version of [Guanciale and Tuosto, 2016].