CS 6110 Software Correctness, Spring 2022
Lec6

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

URL: bit.ly/cs6110s22

Slides for Lec6 : Agenda

- Rozier's paper
  - An amazing intro to FV, model-checking, LTL, CTL, …
  - The fact that Kripke Structures stand out so nicely with the air-traffic example is COOL !!
- U of Rochester has (had?) an UG speciality in SW Engg
  - Model-checking is a required UG class

Scary new world at the edge (with Science of FM, it will be the Exciting New World!)

- Exact quote from paper
  - We assert there is as yet no ``science'' for debating and systematically answering basic questions for how to best facilitate broad, flexible, and effective use of multiple accelerators.
    - https://cacm.acm.org/magazines/2021/12/256949-accelerator-level-parallelism/fulltext

Scary new world at the edge (with Science of FM, it will be the Exciting New World!)

- Exact quote from paper
  - We assert there is as yet no ``science'' for debating and systematically answering basic questions for how to best facilitate broad, flexible, and effective use of multiple accelerators.
    - https://cacm.acm.org/magazines/2021/12/256949-accelerator-level-parallelism/fulltext

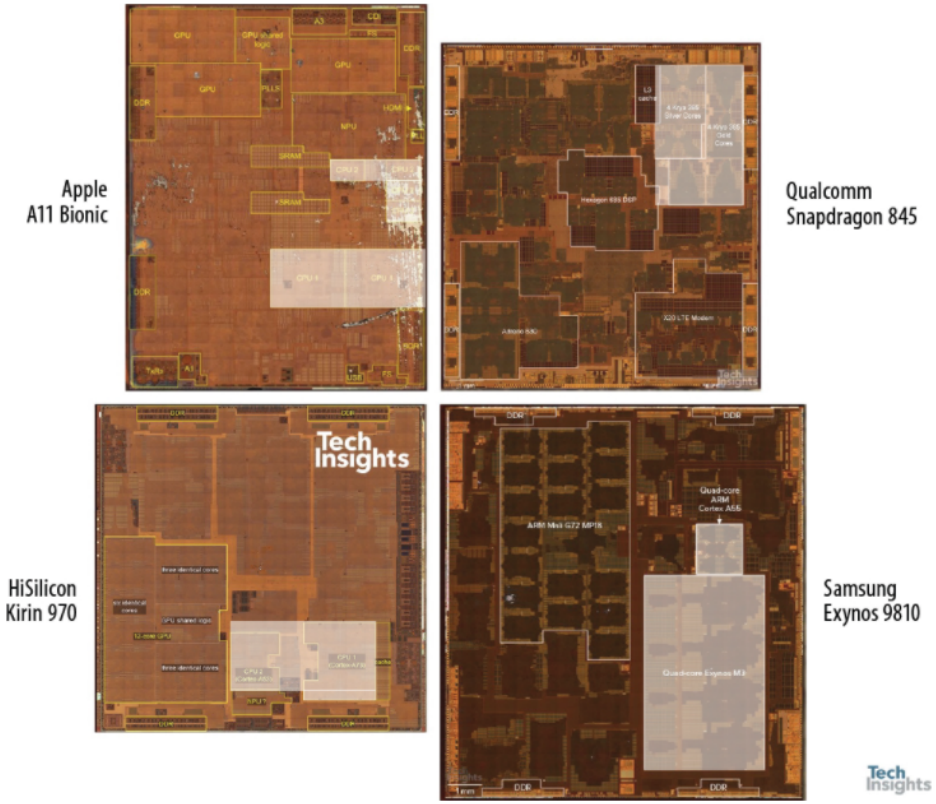

Figure. Modern System-on-Chip (SoC) architectures. The CPUs in modern SoCs (shown in white) occupy only a small percentage of the die area. The rest of the SoC is committed to a potpourri of different accelerators, such as the DSP, GPU, ISP, NPU, video, and audio codecs.

Our
New proposal:
SPACE
Safe
Pervasive
Accelerator
Concurrency
at the Edge

But let’s get back to work!

- Dist termination!
  - Used to know when water-flow ceases
    - Hardy-Cross method (told to me by a Civil Engineer + FM person
      - Courtesy Dr. John Baugh
  - Also in Monte-Carlo simulations in HPC
    - When is the simulation over?
  - We used it for dist. Model-checking!
- Study + design the protocol now!

Dist Term

```
/*---
// Upstream of i is j with j > i. Node numbering clockwise i
// Everything is upstream of root which is 0.
// Node assigning work upstream turns itself B and becomes P
// But that node (that assigns work upstream and turns itsel
// C/E means false or true (C is color, which is W or B)
// State Vector: <NP:I, NS:A, NC:W, HasT: W/B/E, NI: 0..N-1>
// NP=node PC, NS=node state, NC=node color,

R01: <NP:I, NS:A, NC:W, HasT:E, NI: ==0> : tokout! ~~>
     <NP:M, NS:A, NC:W, HasT:E, NI: ==0>

R02: <NP:I, NS:A, NC:W, HasT:E, NI: !=0> :          ~~>
     <NP:M, NS:A, NC:W, HasT:E, NI: !=0>

===

A -> P without work assignment for C color node.
R03: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : silently ~~>
     <NP:M, NS:P, NC:C1, HasT:C2, NI: any>

A -> P with work assignment, upstream
R04: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : workupst! ~~>
     <NP:M, NS:P, NC:B,  HasT:C2, NI: any>

A -> P with work assignment, downstream
R05: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : workdnst! ~~>
     <NP:M, NS:P, NC:C1, HasT:C2, NI: any>
```

Dist Term

```
A -> token being ingested : HasT acquires token color
R06: <NP:M, NS:A, NC:C1, HasT:C2, NI: any> : tokin?C3 ~~>
     <NP:M, NS:A, NC:C1, HasT:C3, NI: any>

A -> does not accept work!
A -> can absorb token but does not send it out till it goes P!

===

P -> A by absorbing work
R07: <NP:M, NS:P, NC:C1, HasT:C2, NI: any> : work? ~~>
     <NP:M, NS:A, NC:C1, HasT:C2, NI: any>

P -> Can circulate token if needed, and the token color depends on no
lor is B
R08: <NP:M, NS:P, NC:B,  HasT:C2, NI: any> : C2 != E / tokout!B ~~>
     <NP:M, NS:P, NC:W,  HasT:E,  NI: any>

Do this if node color is W and NI is not 0
R09: <NP:M, NS:P, NC:W,  HasT:C2, NI: any> : C2 != E / tokout!C2 ~~>
     <NP:M, NS:P, NC:W,  HasT:E,  NI: any>

Do this if node color is W and NI is 0 and local token is B
R10: <NP:M, NS:P, NC:W, HasT:B, NI: 0> :  tokout!W ~~>
     <NP:M, NS:P, NC:W, HasT:E, NI: 0>

Do this if node color is W and NI is 0 and local token is W
R11: <NP:M, NS:P, NC:W,  HasT:W, NI: 0> ~~> Termination

Do this if P and HasT == E
R11: <NP:M, NS:P, NC:C1, HasT:E,  NI: any> :  tokin?C2 ~~>
```

Dist Term

```
Do this if node color is W and NI is 0 and local token is W
R11: <NP:M, NS:P, NC:W,  HasT:W, NI: 0> ~~> Termination

Do this if P and HasT == E
R11: <NP:M, NS:P, NC:C1, HasT:E,  NI: any> :  tokin?C2 ~~>
     <NP:M, NS:P, NC:C1, HasT:C2, NI: any>

---*/
#define Ns      3       /* nr of processes (use 5 for demos)
#define WORK    1       /* does not matter what this is */
mtype = { B, W, E, A, P }; // B,W are for token and node colo

chan workqArray[Ns] = [0] of { bit };   /* rendezvous channel
chan tokqArray[Ns]  = [1] of { mtype }; // really only B,W ; I
mtype ns[Ns]; // really only A,P
bit terminated = 0;

proctype node (chan tokIn, tokOut, workIn; byte myid)
{ mtype nc   = W;
  mtype HasT = E;       /* These xr/xs will throw a false vio
                        /* Suppress this error by turning off
  xr tokIn;  xs tokOut;  byte pick = 0;
  if :: myid == 0 -> tokOut!W :: myid != 0 fi; //--R01

  do
  :: ns[myid] == A ->
  ...fill...

  :: ns[myid] == P ->
  ...fill...
 od;
```

Dist Term

```
 od;
 end:
 //terminated is true here
   assert (...what...)
}

init {
byte i = Ns-1;
        atomic {
        do
        :: i > 0 ->     //--covered by first ND asg-->
           ns[i] = ...figure out...
           run node(tokqArray[i], tokqArray[i-1], workqArray[i], i);
           i--
        :: i == 0 ->
           ns[i] = ...figure out...
           run node(tokqArray[0], tokqArray[Ns-1], workqArray[i], i);
           break
        od
        }
}

//--comment out when doing invalid end-state safety first
never {
do
:: skip
:: terminated &&
  (...what...)
   -> break
od
accept: 1 -> goto accept
```