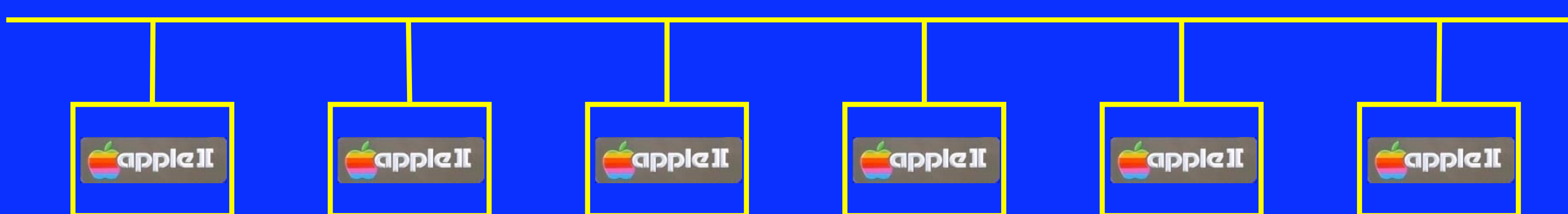
AppleCrate II

Michael Mahon



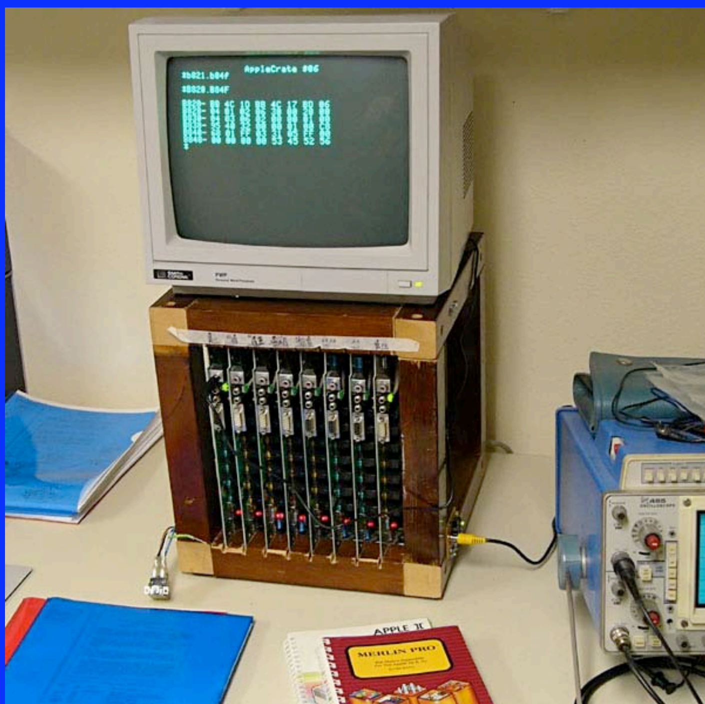
APPLE II - MACINTOSH kansasfest 2008
JULY 22-27
ROCKHURST UNIVERSITY
KANSAS CITY, MISSOURI

Why AppleCrate?

- In the early 1990s, I became interested in “clustered” machines: parallel computers connected by a LAN.
- This interest naturally turned to Apple II computers, and the possibility of creating an Apple II “blade server”.
- A lucky eBay bid in 2003 netted me 25 Apple //e main boards (14 enhanced) for $39, including shipping!

“Because it can be done!”

AppleCrate I



- An 8-machine Apple //e cluster, all unenhanced machines.
- ROMs modified for NadaNet boot (from server)
- Powered by PC power supply
- Fine for a desktop, but quite fragile for travel!

Why AppleCrate II?

- Out of the 25 main boards, 14 were Enhanced //e's.
  - Only 512 bytes of self-test space in ROM required a new "passive" network boot protocol.
- I wanted a 'Crate that was mechanically robust and compact enough to travel.
- I wanted to scale it up to 16 machines, and incorporate a "master" for convenience.
- I wanted better quality sound output.

A machine to support
parallel programming *on Apple II's*.

AppleCrate II

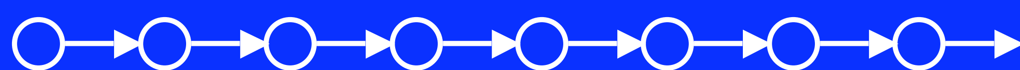
- 17 Enhanced //e boards
  - 1 “master” and 16 “slaves”
  - Self-contained system
- I/O can be attached to the top board, so it is the “master”
- All boards stacked horizontally using standoffs for rigidity
- Total power ~70 Watts
  - ~4.2 Watts per board!
- 17-channel sound
  - External mixer / filter / amplifier
- GETID daisy chain causes IDs to be assigned top-to-bottom

Parallel Programming

- The fundamental problem is maximizing the degree of concurrent computation to minimize *time to completion.*
- To achieve that it is necessary to *decompose* a program into parts that:
  - Require sufficient computation so that communication cost does not dominate TTC
  - Are sufficiently *independent* so that communication does not dominate TTC
  - Do not leave a few large/long sequential tasks whose computation will dominate TTC

Pipeline Parallelism

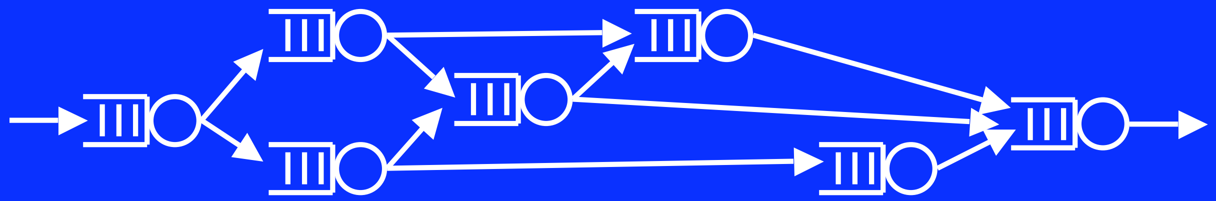
- Processing is divided into “phases” or “stages” that:
  - Require approximately the same time to completion
  - Can be performed essentially independently on many different data sets
- Balancing the times required by each stage independent of the data can be difficult.
  - The pipeline runs at the speed of the slowest stage.
  - A problem in any stage is a problem for the whole pipeline.



This approach can be compared to an assembly line.

Process Parallelism

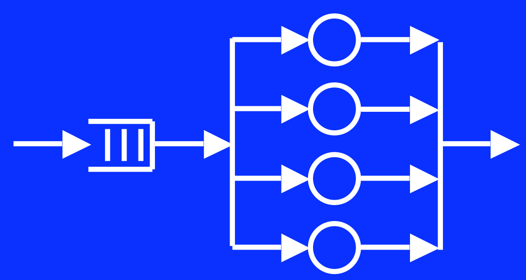
- Processing is divided into separate processes that:
  - Can take any amount of time or resource
  - Can be performed independently on different data sets
  - Any particular data set may have a unique path through the network of processes.
  - Data sets ("jobs") queue for each process
- Balancing processing resources to minimize queueing can be very difficult.



This approach can be compared to scheduling a machine shop floor.

Data Parallelism

- Some problems naturally “fall apart” into many nearly identical independent pieces that:
  - Are sufficiently fine-grained that none dominates TTC
  - Can be easily aggregated to balance computation with communication
- These problems are “made to order” for parallel computation, since decomposition is trivial.



Often so easy it’s known as “embarrassing parallelism”

Examples of Data Parallelism

- Monte Carlo simulations
- Database querys
- Most transaction processing
  - But must still check for independence
- Mandelbrot fractals

BPRUN
(Broadcast Parallel Run)

- RUNs an Applesoft program on all serving machines
- Performs standard AppleCrate initialization
  - Takes census of serving machines
  - Boots any machines awaiting boot
  - Re-takes census
- Starts Message Server if needed
- For each serving 'Crate machine, &POKEs amd &CALLs the BPRUNNER program at $200
- Broadcasts the BASIC program using "boot hack"
- Registers "check-in" of machines running program

Loads and starts all slave machines in parallel.

Parallel Mandelbrot

- Each point is completely independent!
- A “job” could be anything from a single point to *all* 53,760 points!
  - I chose 280 points, or a line, for each “job”
- The master machine queues jobs (in random order)
- Each idle slave machine:
  - Takes the first job in the job queue,
  - Executes the computation, and
  - Enqueues the result for the master to display

The result is an almost linear increase in the speed of execution.

Mandelbrot Master

```
2180  REM  Job parameters
2190 N = 192 : P = 20 : REM 192 jobs, max of 20 at a time
2200 JN = 0:RN = 0:SCH = 0 : REM Start empty
2210 :
2220  REM  Build and maintain job queue
2230  IF JN < N AND SCH < P THEN  GOSUB 2400: REM  Sched another job
2240  IF RN < N THEN  GOSUB 2500: REM  Get result of job
2250  IF RN < N GOTO 2230
2260 :
2270  PRINT  CHR$ (7)"All jobs completed."
2320  END
2330 :
2400  REM  Schedule new job
2410 JN = JN + 1
2430  POKE BUF,LM%(JN - 1): REM  Line number
2440  & PUTMSG (2, JQ, 8, BUF) : REM  Enqueue job in JQ
2460 SCH = SCH + 1
2470  RETURN
2480 :
2500  REM  Receive and display job result
2560  & GET MSG# (2, RQ, LL, BUF) : REM Get result from RQ
2570  IF PEEK(1) THEN FOR I = 1 TO 100: NEXT I: RETURN : REM Delay if no result
2580 SCH = SCH - 1:RN = RN + 1 : REM One less thing to do, one more thing done.
2590 PY =  PEEK (BUF)
2600 H =  INT (PY / 8):L1 = PY - H * 8 : REM Compute start of HGR2 line PY
2604 L3 =  INT (H / 8):L2 = H - L3 * 8
2606 LINE = 4 * 4096 + L1 * 1024 + L2 * 128 + L3 * 40
2607  FOR I = 0 TO 39: POKE LINE + I, PEEK (BUF + 2 + I): NEXT I : REM Display line
2610  RETURN
```

You can see why it's called embarrassing parallelism!

RatRace

- A “pure communication” program
- Each slave is associated with a Message Server input queue
- The queues are “primed” with three messages each
- Each slave machine:
  - Gets the first message from the queue,
  - “Ages” the message by 1, and
  - Puts the message on a random recipient’s queue
    - Until each message has been passed 50 times

2850 messages are sent and received!

RatRace Program

```
600  REM  Message passing loop
610  &  GET MSG#(2,IQ,L,BUF): REM  Receive a message
630  IF  NOT  PEEK (1) GOTO 700
640  PRINT  CHR$ (7);: REM  Delay 100 ms. & flash LED
650 K = K + 1: REM  Timeout counter
660  IF K < 50 GOTO 600
680  END : REM  If 15 seconds w/o message.
690 :
700  REM  Increment message age and pass it on...
710 K = 0: REM  Reset timeout counter
740 S =  PEEK (BUF + 1): REM  Message "age"
750  IF S = 50 GOTO 600: REM  Max trips--it stops here.
760  POKE BUF + 1,S + 1: REM  Inc age by 1 and send it on.
770 D =  INT ( RND (1) * NC) + 3: REM  Random destination, 3..NC+2
800  & PUTMSG#(2,Q + D,20,BUF)
820  IF  NOT  PEEK (1) GOTO 600
830  PRINT "PUTMSG err."
840  END
```

All communication, no computation.

Crate.Synth: Master

- Performs standard AppleCrate initialization
- Reads music file containing voice tables and music streams for each “oscillator” machine
- Loads needed voices and music into each slave
- Loads synthesizer into each slave and starts it (waiting for &BPOKE)
- Starts all slaves in sync when requested

This process could benefit substantially from parallel loading.

Crate.Synth: Slaves

- Waits for master's &BPOKE to start
- Fetches commands from music stream that:
  - "Rest" for T samples (11,025 samples/second), or
  - "Play note N for T samples in current voice, or
  - Change to voice V, or
  - Stop and return to SERVE loop.

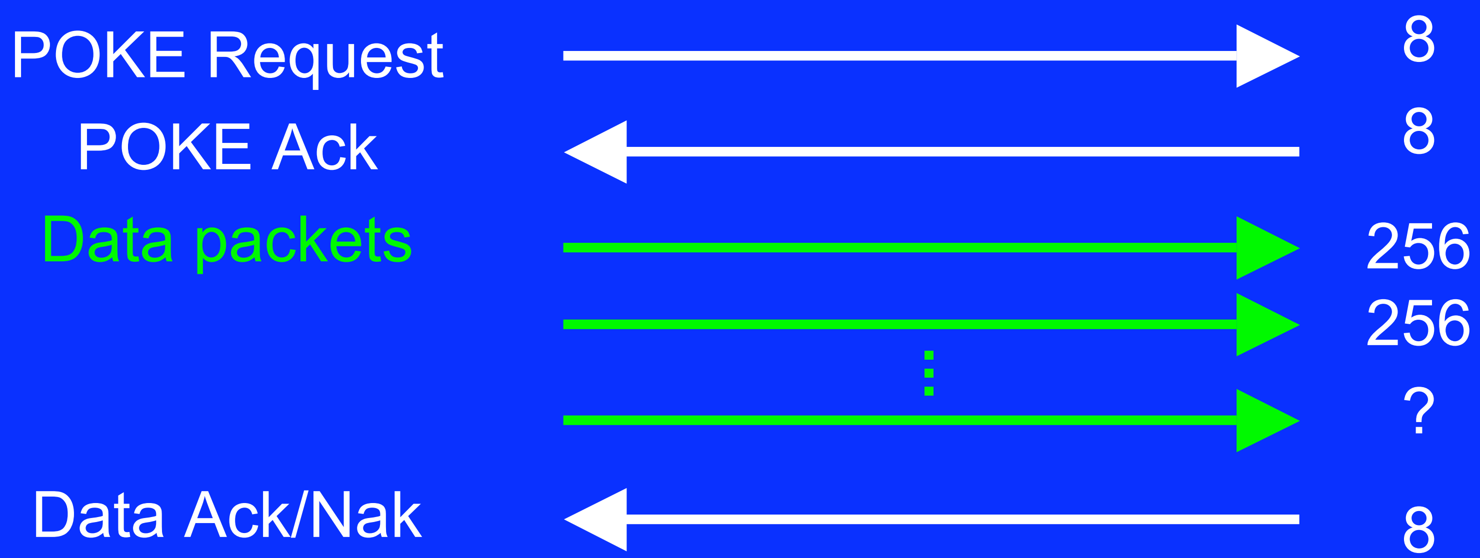
Any oscillator can play any voice at any time.

Questions and discussion...

APPLE II - MACINTOSH kansasfest 2008
JULY 22-27
ROCKHURST UNIVERSITY
KANSAS CITY, MISSOURI

apple II

POKE (A Typical Protocol)



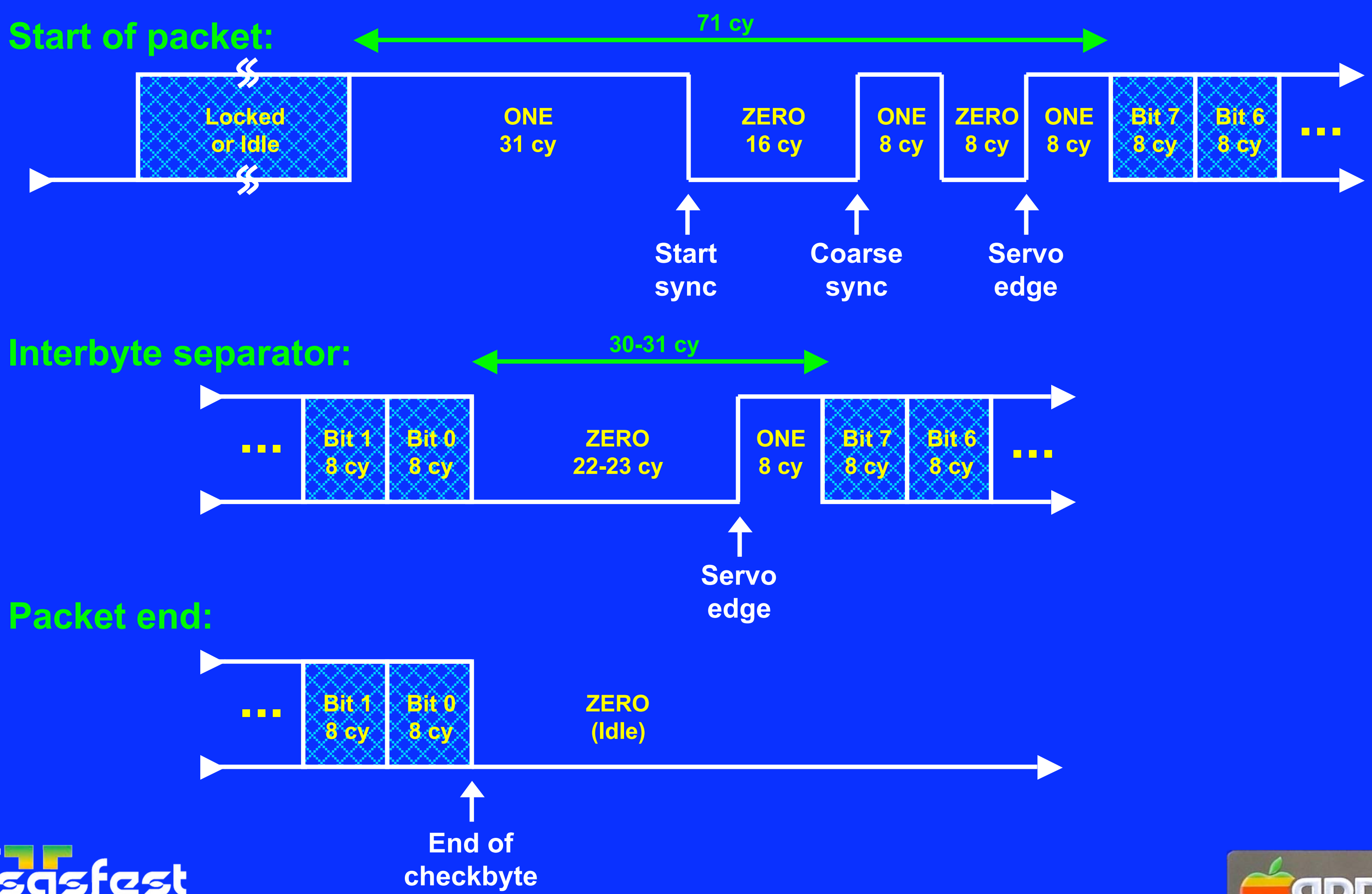
Control Packet Format

Request	Request Modifier	Dest	From	Address	Length	Cksum

- **Request** identifies all control packets of a given request type
  - PEEK, POKE, CALL, etc.
- **Modifier** specifies the role of the packet within the protocol
  - Request, Request Ack, Data Ack, Nak
- **Dest** is the target machine ID
- **From** is the sending machine ID
- **Address** (generally) specifies an address in the target machine
- **Length** (generally) specifies a data length
- **Cksum** is an EOR checksum of all bytes in the packet

Control packets are ~1ms long.

Nadanet Data Format



NadaNet Arbitration

- Always listen before sending
- Wait for net to be idle for 1 millisecond + ID * 22cy
  - Lower ID machines have higher arbitration priority
- Seize net by forcing HIGH state
  - Only 11-cycle sample-to-seize window for idle net collisions

- Consequences:
  - Network is “locked” until it is idle for longer than 1ms.
  - All requests satisfy this requirement and so are atomic.