ICOS:

Support for “Bare Metal” Computer Architecture Assignments

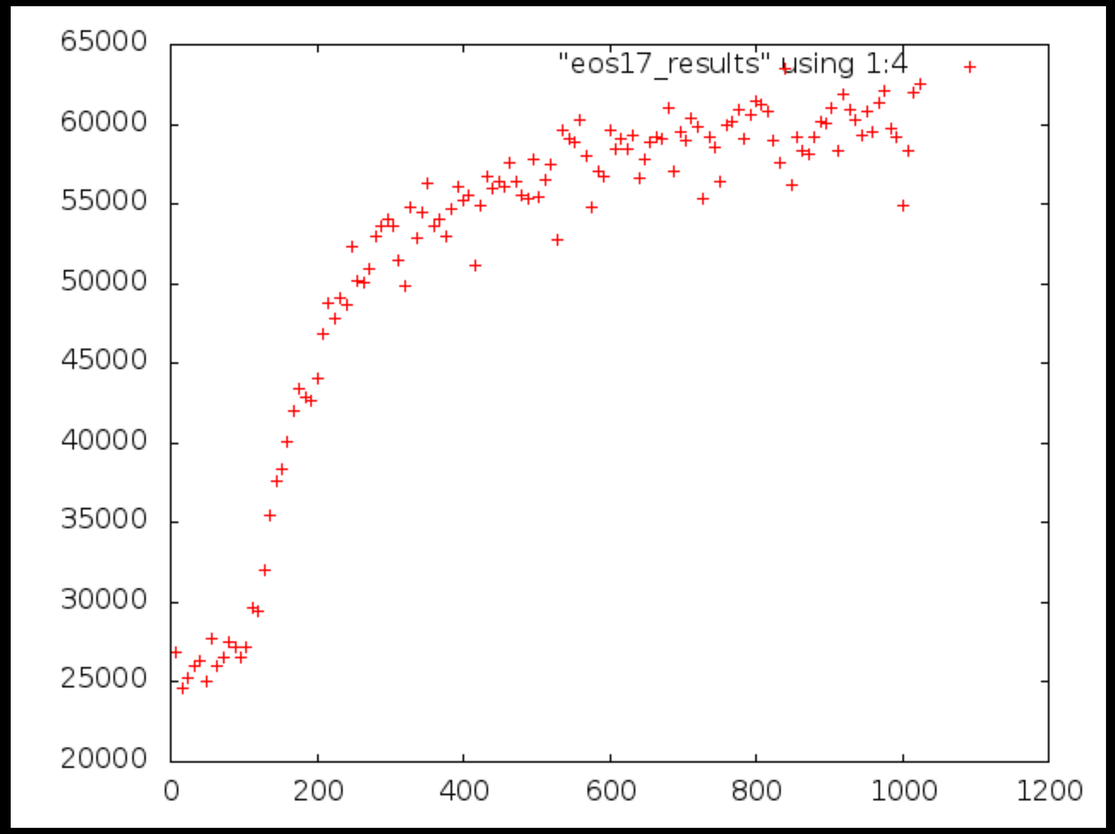
Zachary Kurmas
kurmasz@gvsu.edu

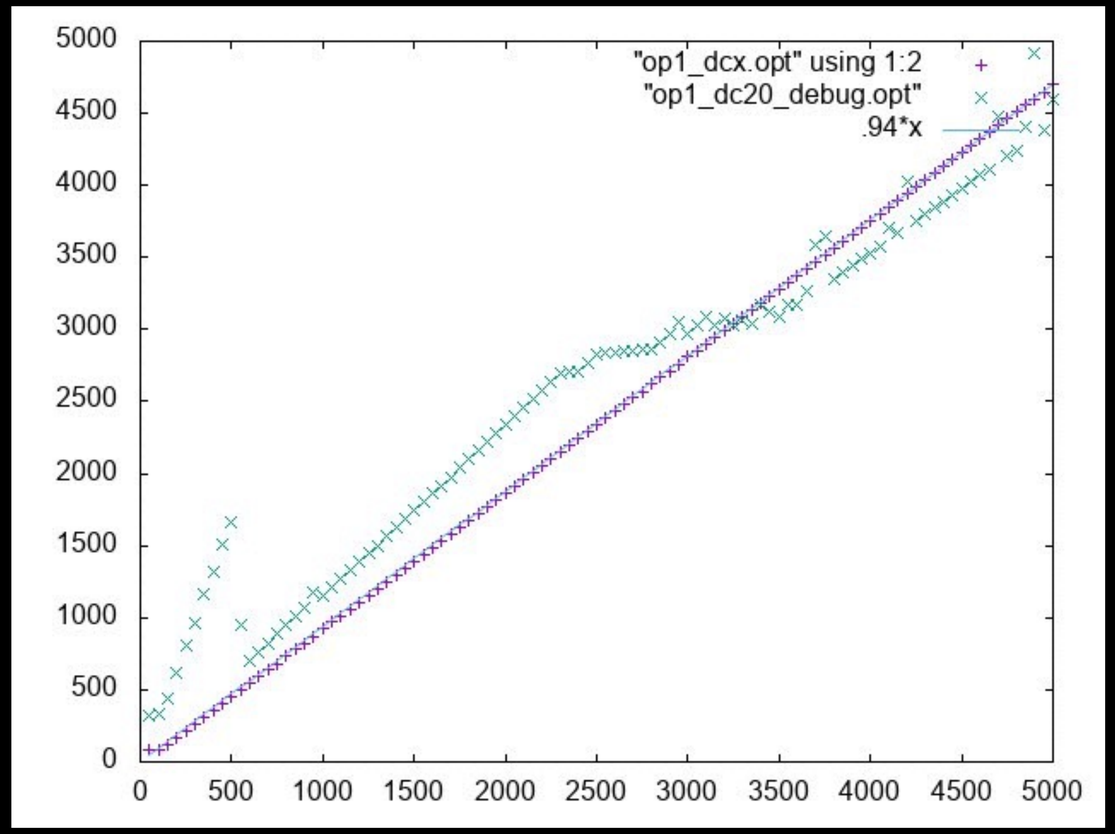
The Story

- GVSU offers two hardware courses
  - CIS 251, Computer Organization, 3 hours
  - CIS 451, Computer Architecture, 4 hours (including a 2 hour lab)
- “Woke up” around 2013 and realized no HW in HW courses
  - Wrote “user space” labs trying to measure branch prediction and superscalar
    - Mostly successful; but noisy.
    - I could see the answer, but some students focused on the noise.

`https://github.com/kurmasz/ICOS/`

Example Noise





The Story

- I assumed noise came from OS (interrupts, context switches. etc.)
- “How hard could it be to boot right into the code for the lab?”
- *<Pause for laughter>*
- 4 years later ….

ICOS

- Framework to run code on “bare metal”
  - Students write C code and
  - Compile it into a bootable image

Consistent performance measurement
- No interrupts
- No virtual memory
- No context switches

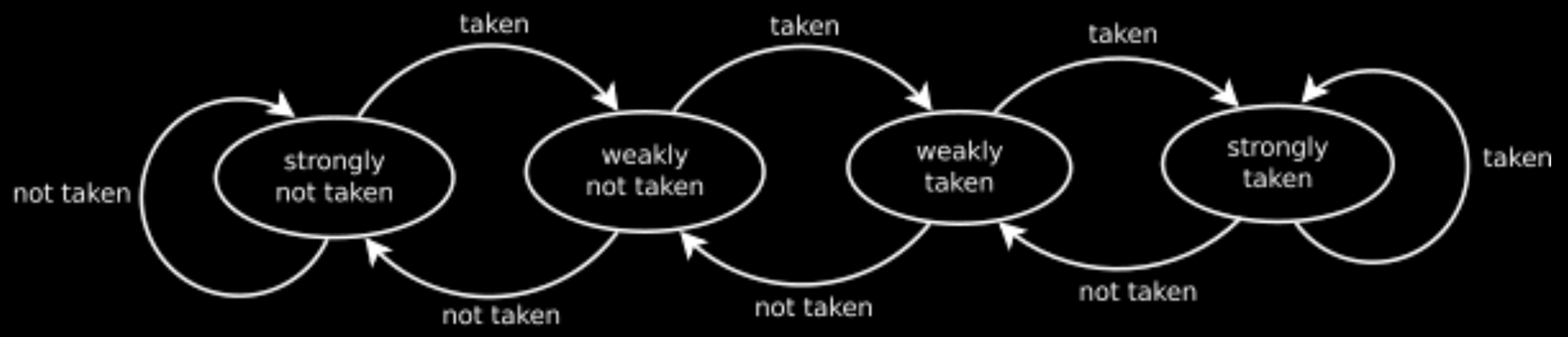
Cost
- No standard C library
- No device drivers
- Very limited I/O
  - 80x25 VGA terminal
  - data buffer dumped back to disk when OS halts

Branch Predictors “In the Wild”

```
int main(int argc, char*argv[])
{
  /* Array Initialization Loop: Initialize the array that determines whether the branch is taken. */
  for (int i = 0; i < SIZE; i++) {
    bool which = random() %2;
    if (i < pattern_length) {
      values[i] = which;  /* Or true or false, depending on the experiment */
    } else {
      values[i] = values[i % pattern_length];
    }
  }

  long unsigned sum1 = 34038, sum2 = 34037; /* Give loop something to do*/

  long unsigned start = rdtsc(); /* start the timer*/
  for (int i = 0; i < SIZE; i++) {
    if (values[i]) {
      sum1 *= 30943; sum1++;
    } else {
      sum2 *= 22891; sum2++;
    }
  }
  long  unsigned stop = rdtsc(); /* start the timer*/

  return stop - start;;
}
```

Key Idea:

Time code that provides evidence that CPU has a branch predictor



Grand Valley State University

Bare Hardware vs. User Space

	Always				
	Min	Average	Max	variance	% Outliers
i7 User Space	51,633	51,927	285,120	4.4×10^6	0.12%
i7 Bare Metal	54,327	54,776	58,236	1.4×10^6	0.00%
i7 Virtual Machine	73,491	77,134	1,241,814	2.5×10^8	5.33%

	Random				
	Min	Average	Max	variance	% Outliers
i7 User Space	85,518	88,211	326,565	8.6×10^6	0.12%
i7 Bare Metal	91,158	95,365	100,269	1.1×10^6	0.00%
i7 Virtual Machine	135,237	160,218	726,528	9.5×10^9	7.97%

Max is less than 110% of average

Key Observations

- User Space and Bare Metal results similar
  - User space version of ICOS much less noisy than early versions
- Difference come from occasional large measurements
- Virtual Machine was surprisingly different

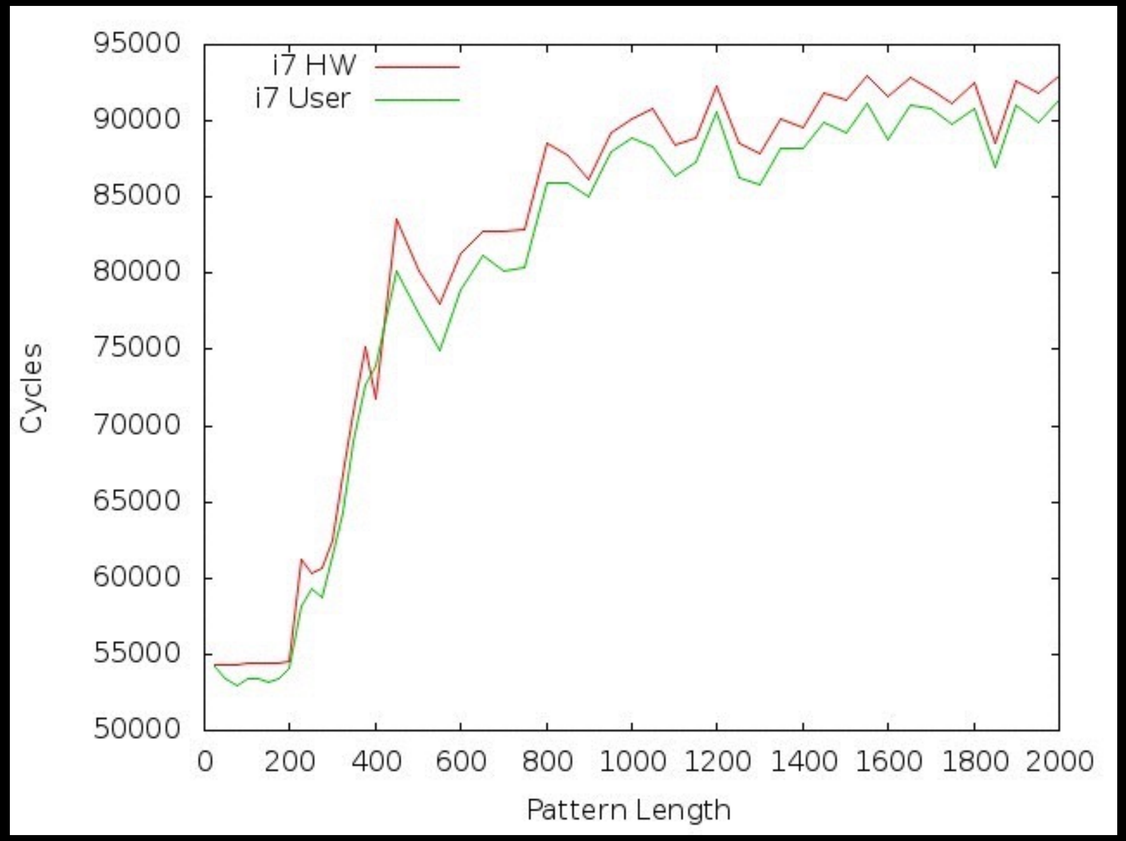
How “Powerful” is Branch Predictor?

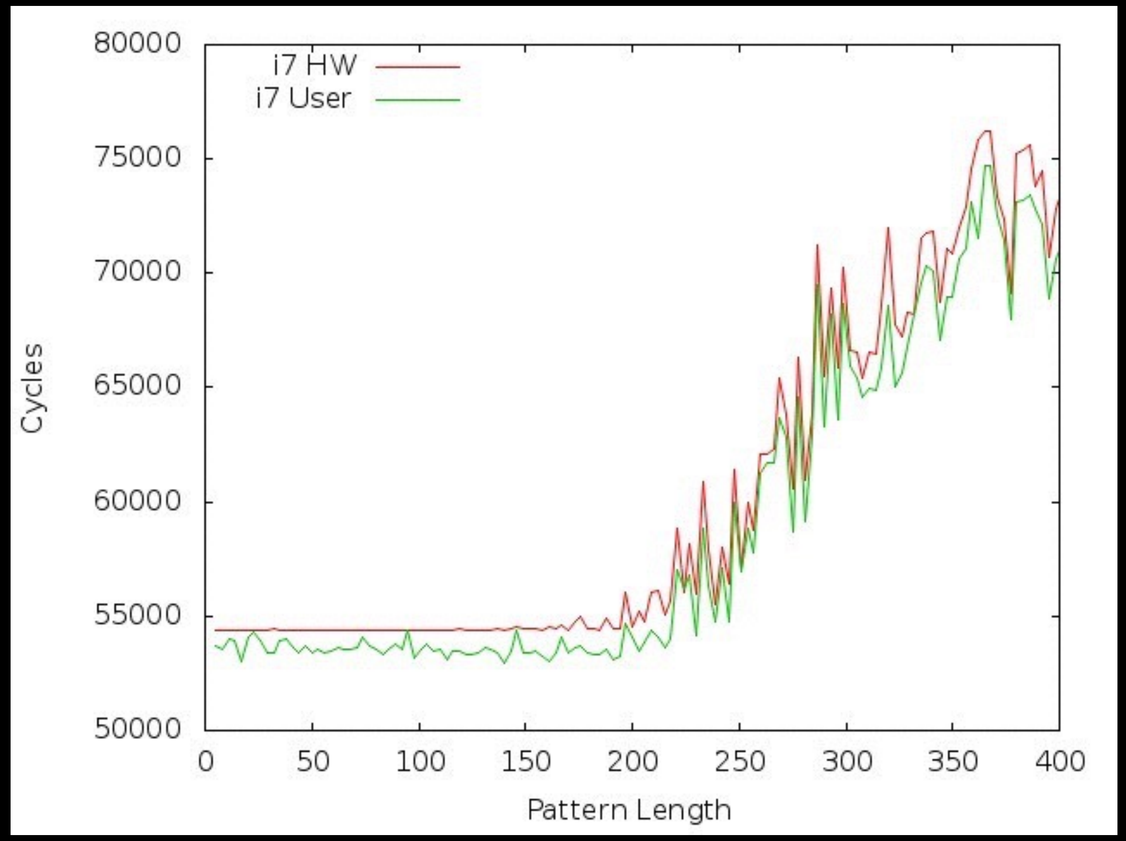
- This repeating sequence of length 5 should be predicted correctly

10110 10110 10110 10110 10110 10110 …

- How long can the sequence get before
  - the predictor accuracy begins to decline?
  - the predictor accuracy is nearly as bad as for a completely random sequence?

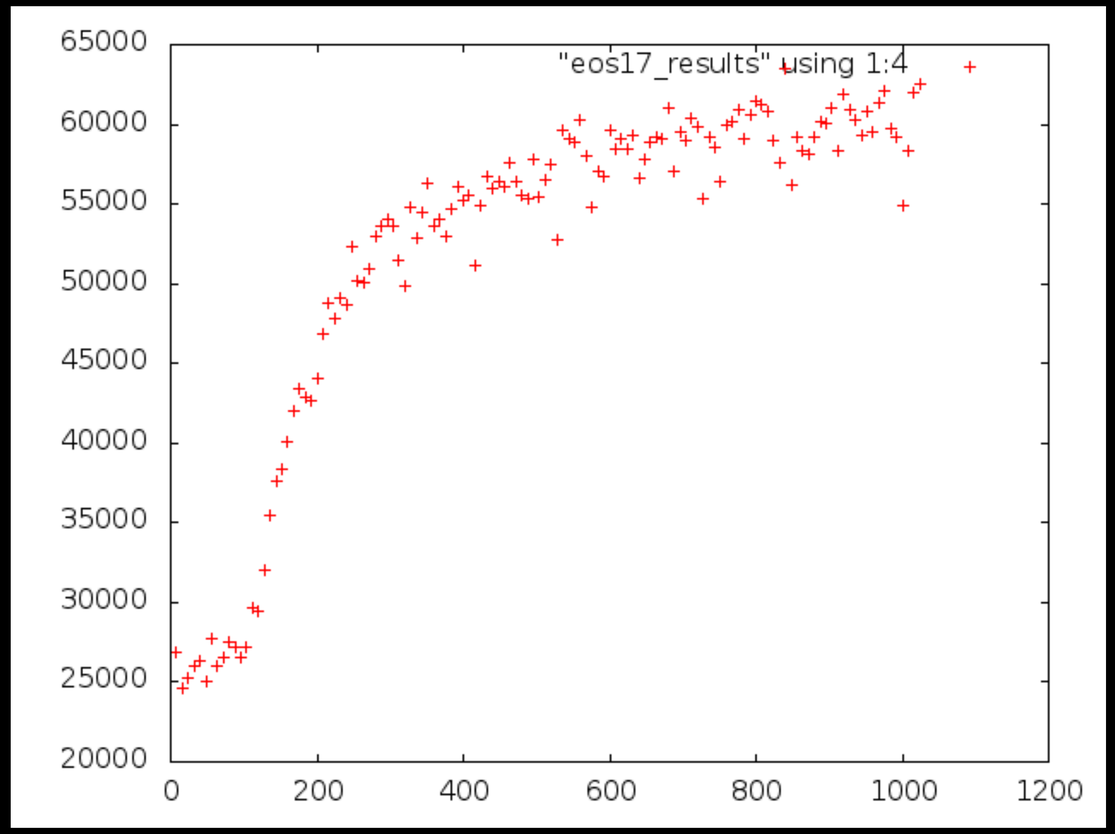
Bare Hardware vs. User Space

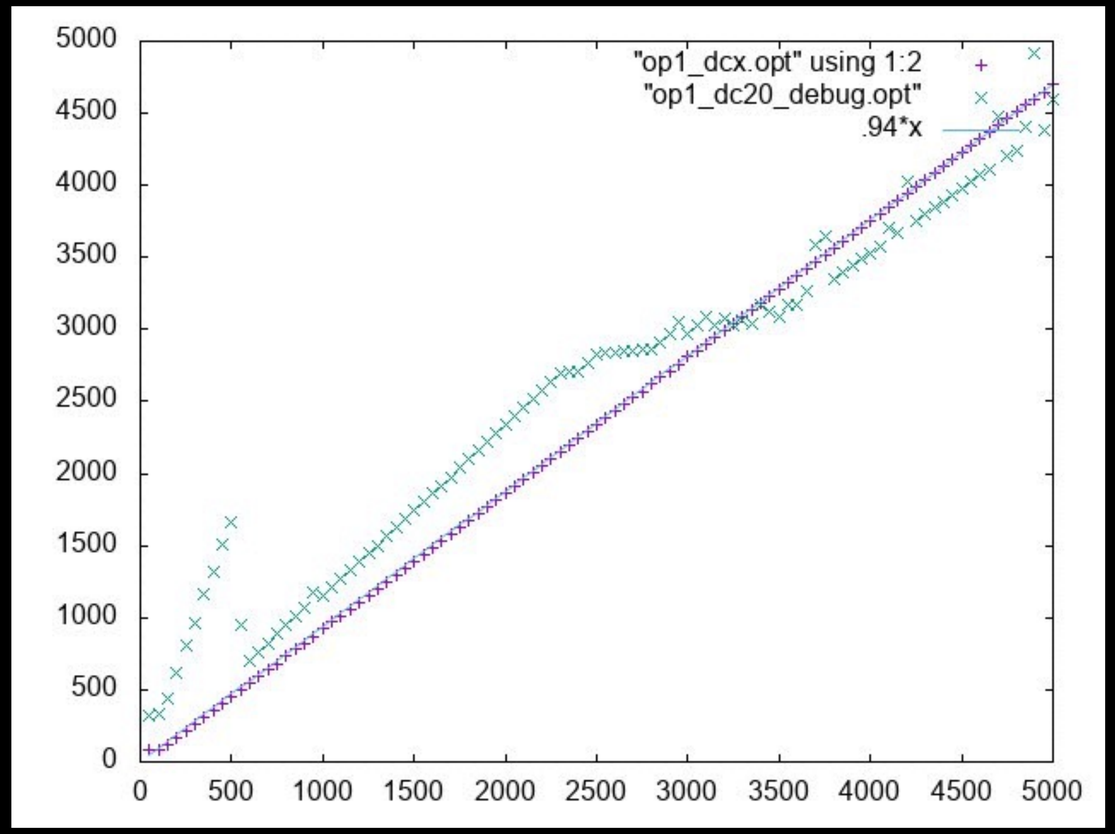




Graphs tell the same story; but, “bare metal” is less noisy

Example Noise





Superscalar

```
rdtsc
push %eax

addl $1, %ecx
addl $1, %ecx
addl $1, %ecx
addl $1, %ecx
addl $1, %ecx
addl $1, %ecx
addl $1, %ecx
… # n total
rdtsc
pop %ebx
subl %eax, %ebx
ret
```

Repeat addl instructions until there are `n` total

- Goal is to estimate the number of functional units in CPU
  - (More accurately, to find the maximum IPC.)
- Count cycles elapsed to execute `n` instructions.
- Choice of `n` is important
  - `rdtsc` has overhead
  - Some `addl` will overlap with rdtsc
- As `n` grows, answer should trend toward true IPC.

Superscalar

- To observe larger IPC, test code with more parallelism
- Question for students: How high can you get the IPC?

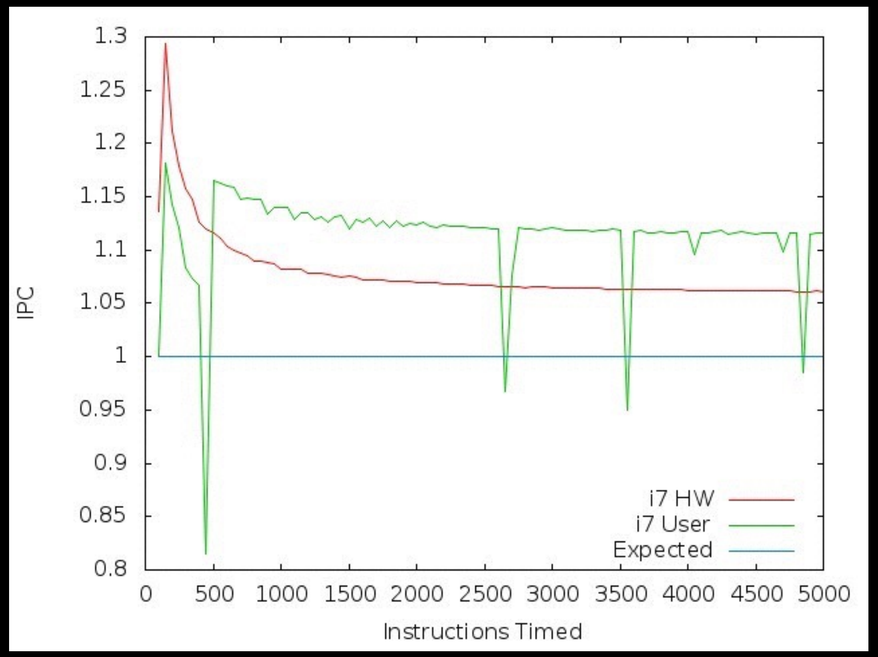
```
addl $1, %eax
addl $1, %eax
addl $1, %eax
addl $1, %eax
addl $1, %eax
addl $1, %eax
…
```

```
addl $1, %eax
addl $1, %ecx
addl $1, %eax
addl $1, %ecx
addl $1, %eax
addl $1, %ecx
…
```

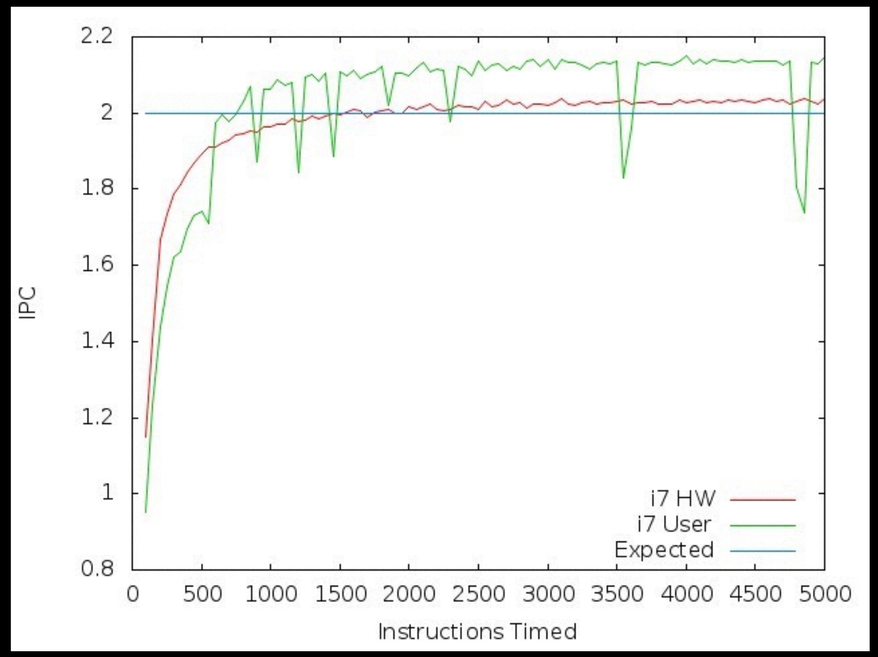
```
addl $1, %eax
addl $1, %ecx
addl $1, %edx
addl $1, %eax
addl $1, %ecx
addl $1, %edx
…
```

Bare Metal vs. User Space

One parallel instruction



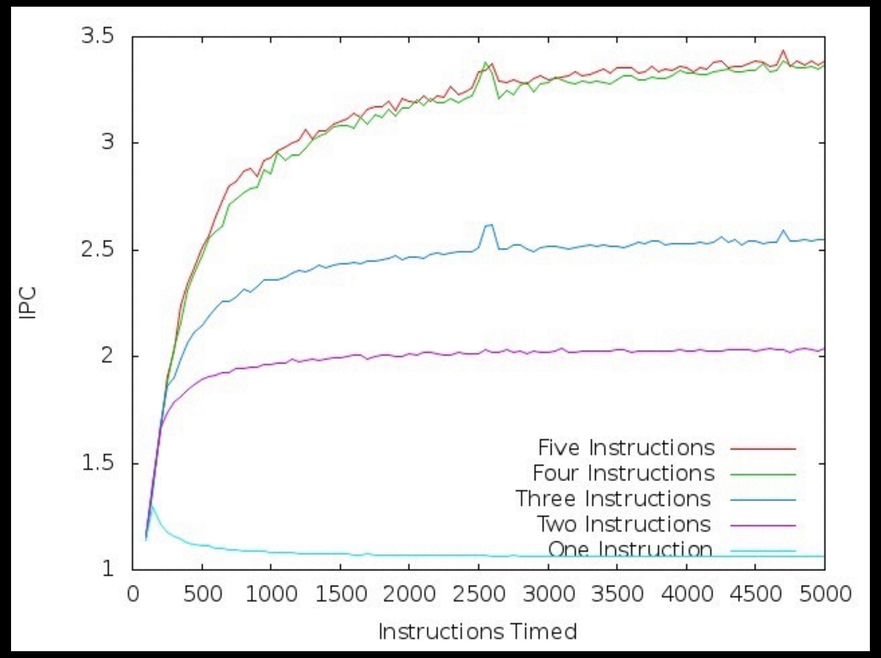
Two parallel instructions



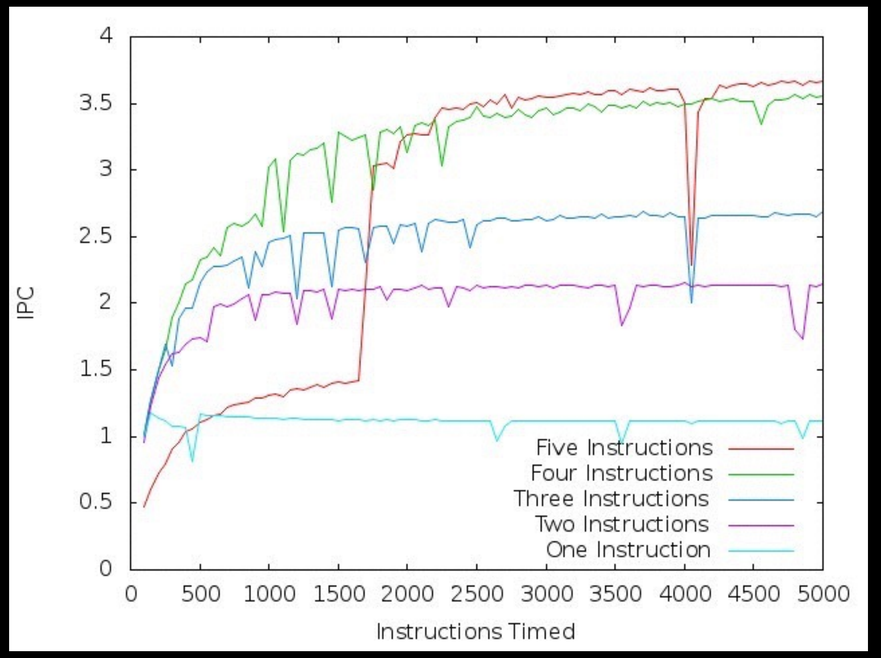
Graphs tell the same story; but, “bare metal” is less noisy

Bare Metal vs. User Space

Bare Metal



User Space



Graphs tell the same story; but, “bare metal” is less noisy

Use in Operating Systems

- Even pedagogically motivated OSes like Minix are very complex
  - Not possible to follow from boot to halt
  - Many now use grub or other standard boot loader

- Would looking at ICOS first help students better understand Minix?

Future Work

- How is the reduced noise from bare metal beneficial to students?
  - Improved Understanding?
    - (Probably not)
  - Improved interest in the course and/or hardware in general?
    - Possible ITiCSE paper. Who's interested?

- Improved standard library
  - printf-style output

Summary

- ICOS makes it easy to run code on bare metal
- Improvements over user space programs are small but noticeable
- Key benefit may be in the “cool factor”
- Potentially useful in Operating Systems courses also

`https://github.com/kurmasz/ICOS/`

ICOS:
Support for “Bare Metal” Computer Architecture Assignments

Zachary Kurmas kurmasz@gvsu.edu http://www.cis.gvsu.edu/~kurmasz

https://github.com/kurmasz/ICOS/