Trusted Computing in 2025 ?
UT Austin ACL2-Group Analysis Tools and Capabilities

Natahlie Beavers	Cuong Kim Chau	Soumava Ghosh
Shilpi Goel	Marijn J. H. Heule	Warren A. Hunt, Jr.
Matt Kaufmann	Keshav Kini	Robert B. Krug
J Strother Moore	Ben Selfridge	Nathan Wetzler

Spring, 2014

Computer Science Department
1 University Way, M/S C0500
University of Texas
Austin, TX 78712-0233

hunt@cs.utexas.edu
TEL: +1 512 471 9748
FAX: +1 512 471 8885

Rigorous Modeling and Analysis of Commercial Systems

To enable the modeling and analysis of industrial-sized systems:

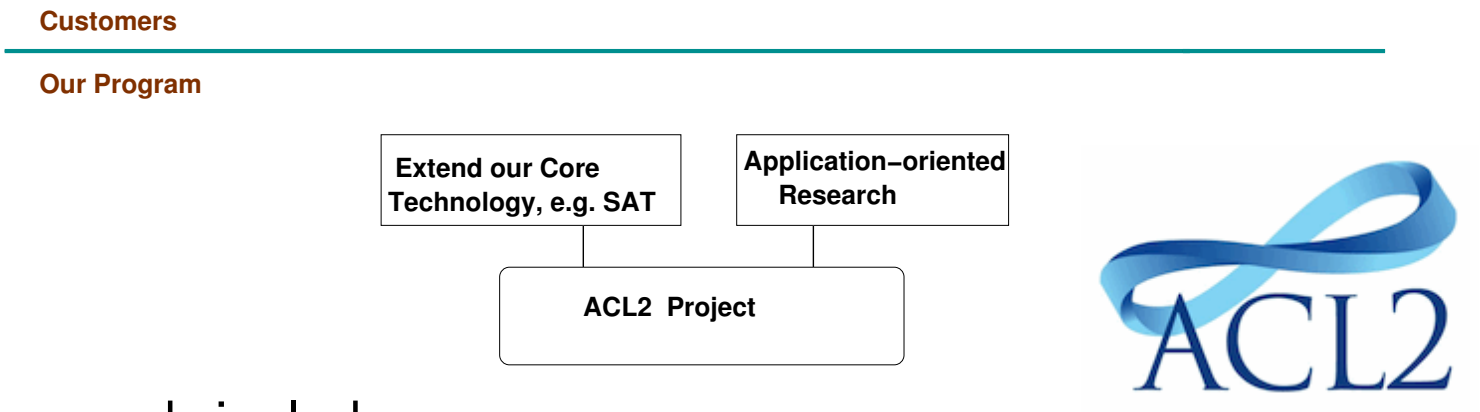
- Centaur is verifying a contemporary (VIA Nano) x86 design.
- We have developed an x86 ISA model suitable for code analysis.
- We are verifying x86 user-level binary programs.
- We are vetting our tools on commercial-sized problems.
- We are not verifying contemporary OSs and system software yet.

Question: Is it reasonable to think that by 2025, we can rigorously analyze a complete computer system's **functional**, **information-flow**, and **security** properties?

Note: Others (e.g., NICTA L4 Micro-Kernel Verification Project, x86 Effort at Harvard, etc.) are also working in this area. For this brief talk, we confine our comments to our efforts.

Ecosystem

We have significant collaboration with industry.

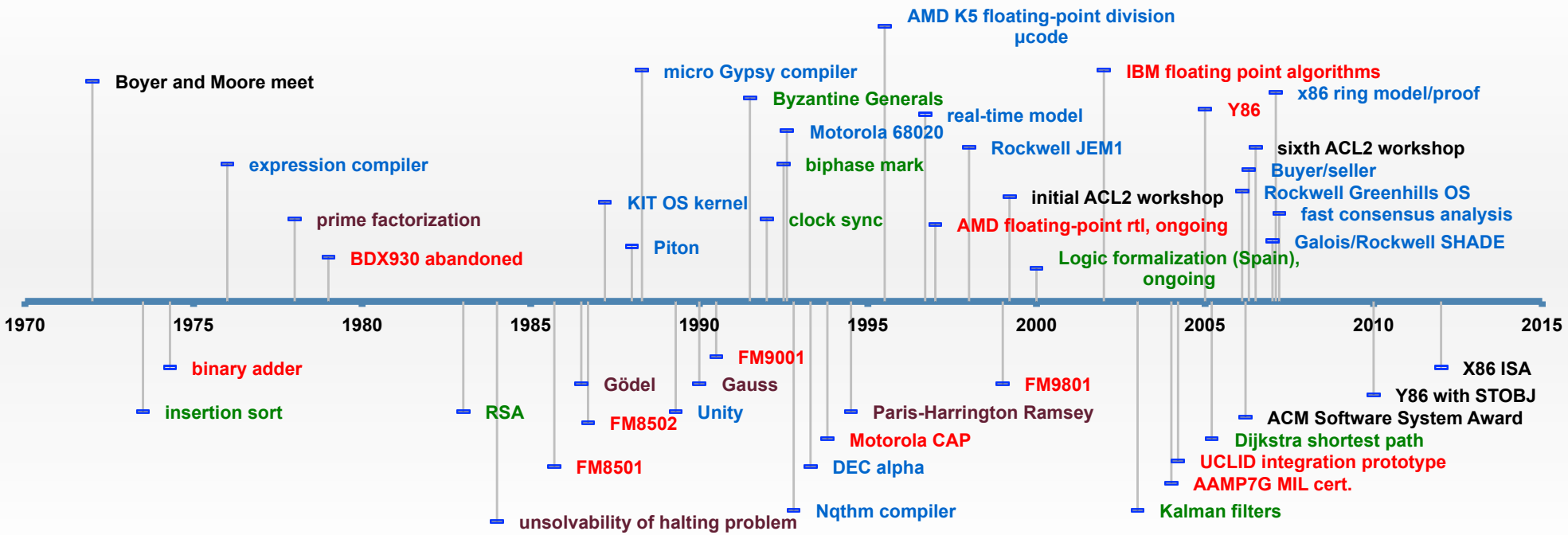


Our own research includes:

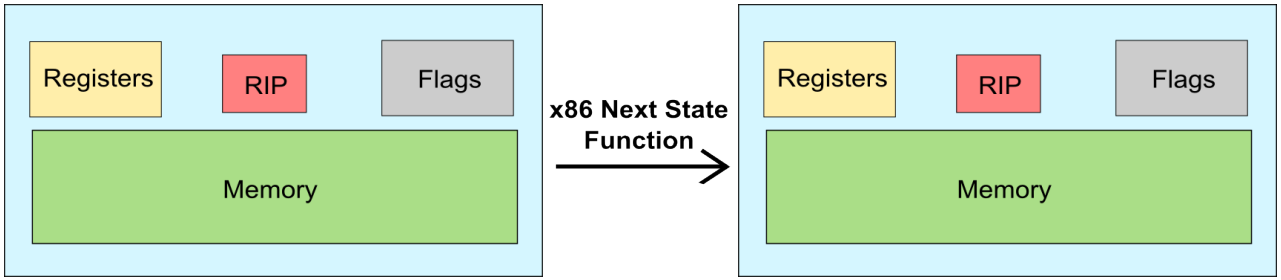
- **Development** of core technologies
- **Application** of these technologies on different verification domains
- **Commercial Driver:** validation for Centaur's x86 design

Timeline

- Our group has been working on the development and deployment of reasoning systems for 40+ years.



x86 ISA Model



We are developing a **formal** and **executable** model of the x86 ISA. Applications of such a model include:

- x86 system emulation
- Binary program verification
- Development of trustworthy programs
- Detecting and analyzing malware and viruses
- Analysis of x86 hardware and software properties
- Build-to (for x86 vendors) and compile-to (for user) specification

Our Progress

Centaur uses our tools to help assure product quality; our tools have displaced a number of commercial tools.

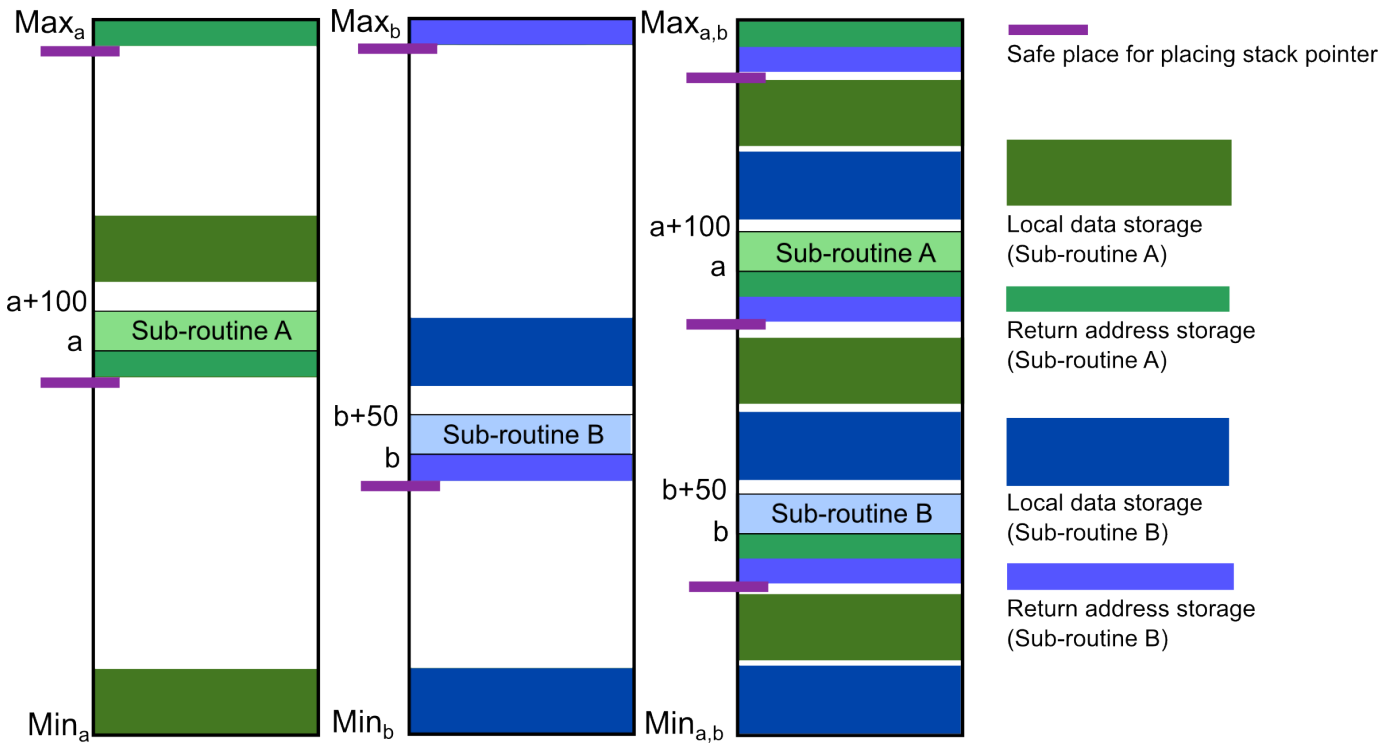
- Centaur models 700,000 lines of Verilog using ACL2
- 3300+ definitions, 4100+ theorems, and 2100+ library references
- Low estimates: macros are heavily used to create such events

Our x86 ISA model can serve as a hardware, build-to specification and as a software, compile-to specification.

- We use our evolving x86 model to verify x86 binary programs
- 400+ definitions, 1500+ theorems (low estimates)
- We mechanically verify small x86 binary programs

Example: Verification of a Simple WC Program

- Simple WC program: reads input from the user and computes the number of characters, words, and lines in it.
- Proof of functional correctness of simple WC also provides **memory guarantees.**



Simple Word-Count Program

```
#define IN 1    /* inside a word  */
#define OUT 0   /* outside a word */
#define EOF '#' /* EOF character  */

#include <stdio.h>

int gc(void) {
  char buf[1];
  int n;
  __asm__ volatile
    (
     "mov $0x0, %%rax\n\t"
     "xor %%rdi, %%rdi\n\t"
     "mov %1, %%rsi\n\t"
     "mov $0x1, %%rdx\n\t"
     "syscall"
     : "=a"(n)
     : "g"(buf)
     : "%rdi", "%rsi", "%rdx");
  return (unsigned char) buf[0];
}
```

```
/* count lines, words, & chars */

int main () {
  int c, nl, nw, nc, state;
  state = OUT;
  nl = nw = nc = 0;
  while ((c = gc()) != EOF) {
    ++nc;
    if (c == '\n')
      ++nl;
    if (c == ' '  ||
        c == '\n' ||
        c == '\t')
      state = OUT;
    else if (state == OUT) {
      state = IN;
      ++nw;
    }
  }
  return 0;
}
```

Simple Word-Count: Proof of Correctness

Helper Lemma:

$$\forall x : pre(x) \implies (run(i, x))$$

Final Correctness Theorem:

$$\forall x : pre(x) \implies halted(run(c, x)) \wedge post(x, run(c, x))$$

where $c = i + l$, where i is the number of steps to reach the beginning of the loop.

Proof checked by ACL2 theorem-proving system.

Other Sources of Non-Determinism

- We can use our framework to simulate and reason about other non-deterministic computations.
- For instance, the `rdrand` instruction provides cryptographically secure random numbers to applications at all privilege levels by loading a hardware-generated random value in a processor register.

Our Vision for Binary-Code Analysis
Memory Guarantees

Goal: Develop a formal verification toolsuite to reason about a program's memory consumption.

- We have a library of ACL2 lemmas that facilitates automated reasoning to provide these memory guarantees in the user-level mode.
- We continue to improve this library by adding reasoning support for additional instructions and other processor modes.

Progress on Analysis Efforts

- We can reason about **system calls** and other **non-deterministic** computations.
- We have libraries to reason about overlapping and disjoint memory regions. For example, we can verify whether the stack and heap overwrite the program or each other during execution or not.

We are working on commercial-size artifacts, but we are far away from complete verification of x86 implementations and large software systems like OS kernels.

Big Impact Bugs

In the last few weeks, there have been some software bugs reported that are having a big impact; see:

`http://heartbleed.com`

`https://technet.microsoft.com/en-us/library/security/2963983.aspx`
`http://www.cnet.com/news/microsoft-fixes-big-ie-bug-on-windows-xp-even/`

A Core Question: Is it reasonable to think that by 2025, we will be able rigorously analyze a complete computer system's **functional**, **information-flow**, and **security** properties?

Re-Consideration of the Core Question

Are we working on a fool's errand?

- By 2025, could we mechanically verify an x86 implementation with "*10%*" of the performance of a commercial design?
- By 2025, could we mechanically verify an Linux-like operating system and applications with "*10%*" of the complexity of today's systems?

If the answer is **No**, then we will not have trustworthy components. We then need a **risk-mitigation strategy**.

If the answer is **Maybe**, then we should embark on a program to make trustworthy computing a reality. We need to remove IP and legal impediments to allow the free and **mathematically-accurate** exchange of hardware/software specifications and analyses.