Stack Machines

Towards Scalable Stack Based Parallelism

Tutorial Organizer:

Dr Chris Crispin-Bailey
Department of Computer Science
University of York
Today’s Speakers

Dr Chris Crispin-Bailey

Dr Mark Shannon

Dr Huibin Shi
Stack Machines

Towards Scalable Stack Based Parallelism

Dr Chris Crispin-Bailey
Department of Computer Science
University of York
Tutorial Plan

1. Exploding the Myths  
   (Chris Bailey)

2. Compilers & Code Optimization  
   (Mark Shannon)

<< Coffee Break >>

3. Measuring Parallelism  
   (Huibin Shi)

4. Parallel Architectures  
   (Chris Bailey)
Core Motivation for this tutorial …

Can stack machines exploit parallelism?

Can stack machines offer anything useful for future architectures?

First, let's try to define the problem of **performance** in suitable terms …
Performance – A Complex Equation

\[ \text{TLP} \times \text{ILP} \]

Technology Scaling
Performance – A Complex Equation

- multicores
- SMT
- virtualisation
- microthreading
- Instruction Level Parallelism
- Raw Instruction Expression
- Power Consumption
- Wire Scaling
- Power Density
- Dark Silicon
Performance – A Complex Equation

Sounds great?

But how much of this is well understood in the context of stack machines?

Well, we think we know about this bit …

So there is a lot to be done … !
But First .. A little history

Where did stack machines come from?

What happened next …
Stack Machines in a nutshell…

1920 ‘POLISH NOTATION’ was invented by Jan Łukasiewicz

1957 C. Hamblin invents Reverse Polish Notation

first commercial Stack Machines – KDF9 - 1964

Burroughs machines B5000 etc 1961 onward
KDF9
English Electric Company

First Used 1964
Still in use in 1980!
1970’s Integrated circuits overtook the stack machine advantage of simplicity

1980’s RISC revolution cemented Register dominance

1990’s INMOS Transputer – Stack Microprocessor

2000’s Java and Java Processors – limited impact

2010’s CISC Approach In trouble
– can stack machines come back?
What If Development of Stack Machines Had Continued?

What If Development of Stack Machines Had Continued?

What should have been done:

- Address ‘inferior’ stack machine limitations
What If Development of Stack Machines Had Continued?

What should have been done:

- Address ‘inferior’ stack machine limitations
- Develop advanced code generation comparable to register machines
What If Development of Stack Machines Had Continued?

What should have been done:

- Address ‘inferior’ stack machine limitations
- Develop advanced code generation comparable to register machines
- Invent methods to exploit general parallelism
Back to the Tutorial Plan...

1. Exploding the Myths (Chris Bailey)

2. Code Optimization (Mark Shannon)

<< Coffee Break >>

3. Measuring Parallelism (Huibin Shi)

4. Parallel Architectures (Chris Bailey)
Part-1

Stack Machines
Exploding the Myths
Stack Machines: Exploding the Myths

i. The ‘case’ against stack machines
1960’s A Turning Point For stacks

IBM 360 systems designers chose Register-Based systems, citing key points that influenced their decision.

After 50 years we still hear the same evidence, increasingly mis-quoted.

Let's start by examining the folklore …
Stack Machines Myth and Misconception …

Stack machines – some common responses …

“Didn’t they stopped using those in the 60’s?…”
“Nobody is looking at this area so it must be a waste of time”
“Everyone knows stack machines don’t do parallelism”
“They were shown to be inferior a long time ago…”

However – where is the research to back this up?
- In truth there is very little evidence to write off stack machines after 50 years of trying we still rely upon assumption…
Stack Machines Myth and Misconception …

What P&H say about stack machines …

1. “Performance derives from fast registers, and not the way they are used”

2. “Stack organization requires many swap and copy operations”

3. “The bottom of the stack (when placed main memory) is slow”

This is a highly influential text book which students rely upon heavily, but is it right?
Stack Machines Myth and Misconception …

Papers Quoted :-

• Amdahl et al 1964
• Bell et al 1970
• Hauck and Dent 1968
• Myers 1978

A little out of date perhaps? …
Stack Machines Myth and Misconception …

Major Changes since 1970:-

- INTEL invented the microprocessor!
- We stopped using core store for memory
- Nearly all modern programming languages were invented
- Register allocation, virtual register sets, superscalar issue, cache, burst-dram, multi-level memory, 1-billion transistor chips
- Multicore, scaling, power density wall, Moores law, etc
Stack Machines Myth and Misconception …

- So at the very least we should take an objective look at this key research before allowing our minds to be made up for us?

- Were these claims true in 1970?
- Are they still true now?
Stack Machines Myth and Misconception …

1. Performance derives from fast registers, and not the way they are used

Amdahl 1964 as quoted by P&H

What Amdahl et al actually said …

“the performance advantage of stack processors comes from fast registers, not how they are used”

Amdahl 1964

This doesn’t seem to support any idea of stack machines being inferior, indeed Amdahl highlights the advantage of stack machines offering fast registers – an issue that is still critical in today’s CPU designs.
2. Stack organization requires many swap and copy operations

Amdahl 1964 as quoted by P&H

In the paper it says …

“about 50% of operands appear at the top of stack when required”

This is true if one relies upon ‘naïve’ compiler output with no relevant optimization. A register based machine would have the same problem without register allocation and graph coloring.

But if stack code is inefficient then the solution is to optimize it, not abandon the architecture as a useful design

(Part 2 will expand on this – much recent work has been done)
Stack Machines Myth and Misconception …

3. The bottom of the stack (when placed in main memory) is slow.

Bell 1970

…

This is true, but so is the following :-

“A register file is very slow if half of it is placed in main memory …”

But we do not have to put anything in main memory !! …

- On chip stack buffers (perfectly possible in 1960)
- On chip cache (a ‘recent’ innovation …

so the problem is a false proposition in terms of today’s hardware Systems and was probably avoidable even 40 years ago.
Stack Machines Myth and Misconception …

1. Performance derives from fast registers, and not the way they are used.

2. Stack organization requires many swap and copy operations

3. The bottom of the stack (when placed main memory) is slow.

The ‘case against stack machines’ is no longer convincing …
Stack Machines Myth and Misconception …

THE EVIDENCE IS IN THE RESEARCH …

• ‘New’ Code optimization techniques
  - Koopman, Bailey, Shannon, Maierhofer ….

• Stack Buffering and Cache Hierarchy
  - Many examples, many alternative technologies

• Instruction sets that optimise operand management
  - Bailey … orthogonal stack management
Stack Machines Myth and Misconception …

**Conclusion**

- **Perceptions of stack machine technology is severely outdated**

- **We have been diverted from more interesting questions …**
  - Can they fully exploit parallelism?
  - Do they make multicore easier?
  - Can they address power-density or wire-scaling issues?
  - Can they approach register file performance gate for gate?
Stack Machines Myth and Misconception …

**Conclusion**

- But, to get closer to answering these problems we need to show …

1. Stack machines can effectively manage stack spill traffic
2. Stack machines can optimize code to eliminate ‘poor efficiency’
3. Stack Machines can exploit useful degrees of parallelism

This will be shown to be feasible in the tutorial.
Stack Machines:

Stack Buffering Methods
Stack Buffering Schemes

‘The stack has a bottom, and it is inefficient when placed in memory’

Making this statement today ignores fundamental computer architecture, as well as more advanced techniques,

It is useful to understand the possible methods by which the problem can be addressed.
Stack Buffering – The problem as it was presented …

Bell perceived the problem in terms of 1960’s computer architecture

We had a register bank on chip and a slow main memory

Where else could the stack go?

There are lots of options … some were perfectly feasible even in the 1960’s.
Stack Buffering – Some of the options

1. Simplest Solution:-

- Don’t spill stack to memory
- Stack has finite depth
- INMOS TRANSPUTER
- Intel 4004 program stack
- Has no ‘memory’ problem …
Stack Buffering – Some of the options

2. Just ignore the problem :-

- always spill stack to memory
- very inefficient as Bell Observed

- 1960’s stack machines
- This is the worst possible implementation
- Not a very fair comparison!
Stack Buffering – Some of the options

Location Location!
- Stack spill traffic has high degree of locality
- Pushes and pops tend to cancel each other

Push Effect
- Spill traffic repeatedly references the same memory addresses (i.e. often redundant)
Stack Buffering – Some of the options

3. Cache Will eliminate redundant Accesses
   • caching will absorb most spills
   • may reduce cache bandwidth

There are many variations ....
   • Write Through
   • Write Back
   • Line based read/write policies

We have had these components for decades!
Stack Machines – Towards Scalable Parallelism

Stack Buffering – Some of the options

The way in which we implement cache for stack spilling is open to many permutations.

Single Shared Cache

External Cache

Main Memory

Cpu core

stack
Stack Buffering – Some of the options

Stack Machines – Towards Scalable Parallelism

Dr Chris Crispin-Bailey
Advanced Computer Architectures Group

Single Shared Cache

Multi-Level Cache

External Cache

Main Memory

Cpu core

On chip cache

External Cache

Main Memory
Stack Buffering – Some of the options

Stack Machines – Towards Scalable Parallelism

Single Shared Cache

Multi-Level Cache

Split Cache schemes
Stack Buffering – Some of the options

But even standard cache is a poor solution

We can use simpler buffer schemes to gain Benefit with much fewer gates/transistors
Stack Buffering – Dedicated Buffers

- Many buffer schemes exist
- Much more compact than equivalent cache
  - Less silicon area
  - Lower power consumption
Stack Buffering – Dedicated Buffers

- Many buffer schemes exist
- Much more compact than equivalent cache
  - Less silicon area
  - Lower power consumption
- Typically Very sharp fall off curve :-

![Graph showing spill traffic % vs buffer capacity]

External Cache

Main Memory

Cpu core

buffer
cache
Far from every stack operation creating a spill, these events are rare even with a small **hysterisis** buffer.

This can be implemented with a simple pointer – something Bell could have utilized even in the limited 1960’s hardware.
Stack Buffering – Dedicated Buffers

Buffer Schemes can be very simple:-

- Single pointer
- Double pointer
- Valid and Dirty Bits

Hugely simplified by comparison to an addressable memory area or a cache

Even small buffers (4 or 8 elements) give large reductions in stack spill traffic.
Stack Buffering – Dedicated Buffers

Why so effective?

- **Locality** of Reference
- Stack spills & fills are tightly coupled

- Buffers ‘soak up’ repeated reads and writes to the same location
  - this is a valuable point for superscalar stack machines (see later)

- Downside is that buffers can increase task switch latency …. 
  - Alternatives include traditional cache or memory queues
  - these approaches have reduced context switching latency
  - but they also introduce non-determinism and ‘warm up’ effects
Conclusions?

‘Stack machines are inefficient when the stack bottom is in memory’ X?

- The claim is correct, but the scenario is ‘contrived’ …

- The answer is do not put the stack in memory !!

  - Use a cache, multiple caches, and/or a buffer
  - this is standard in most CPU’s today

  - Any future comparison of Stack vs Register MUST reflect this simple solution to be taken seriously.
Pause For Reflection ....

1. Performance derives from fast registers, and not the way they are used

2. Stack organization requires many swap and copy operations

3. The bottom of the stack (when placed in main memory) is slow.
Pause For Reflection ....

1. Performance derives from fast registers, and not the way they are used

2. Stack organization requires many swap and copy operations

We will address problem 2 next, and come back to problem 1 later on.
End of Part 1

Questions?
Tutorial Plan

1. Exploding the Myths (Chris Bailey)

2. Code Optimization (Mark Shannon)

<< Coffee Break >>

3. Measuring Parallelism (Huibin Shi)

4. Parallel Architectures (Chris Bailey)