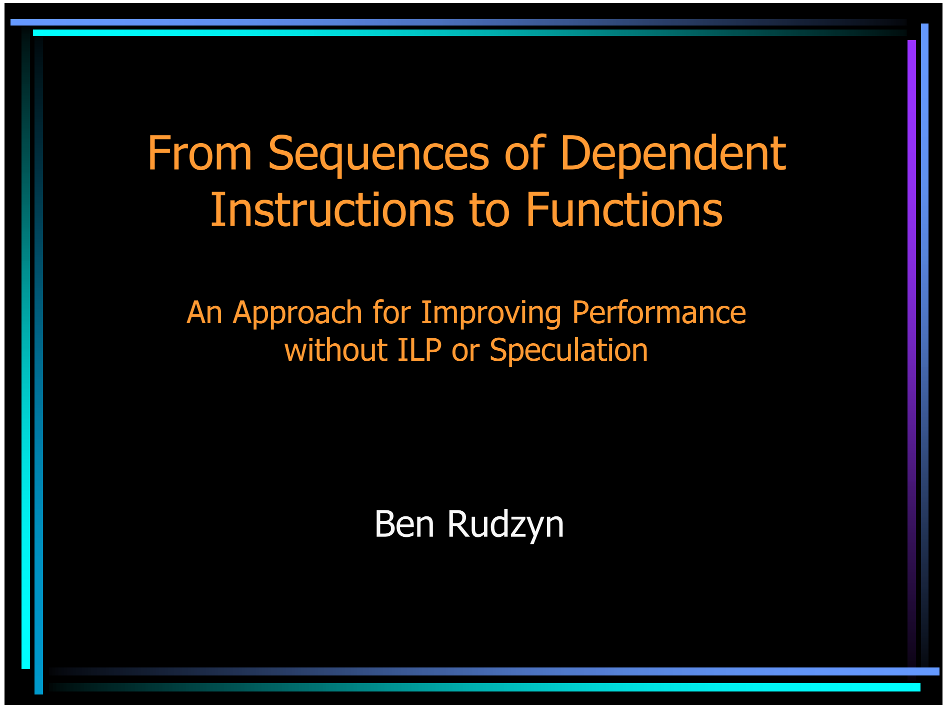
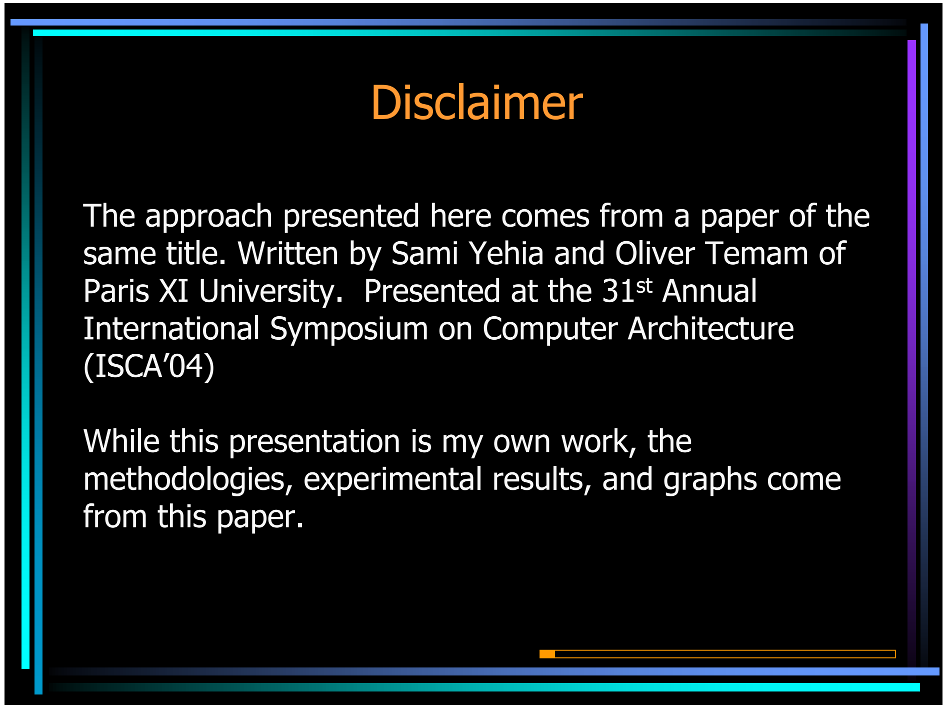


From Sequences of Dependent Instructions to Functions

An Approach for Improving Performance without ILP or Speculation

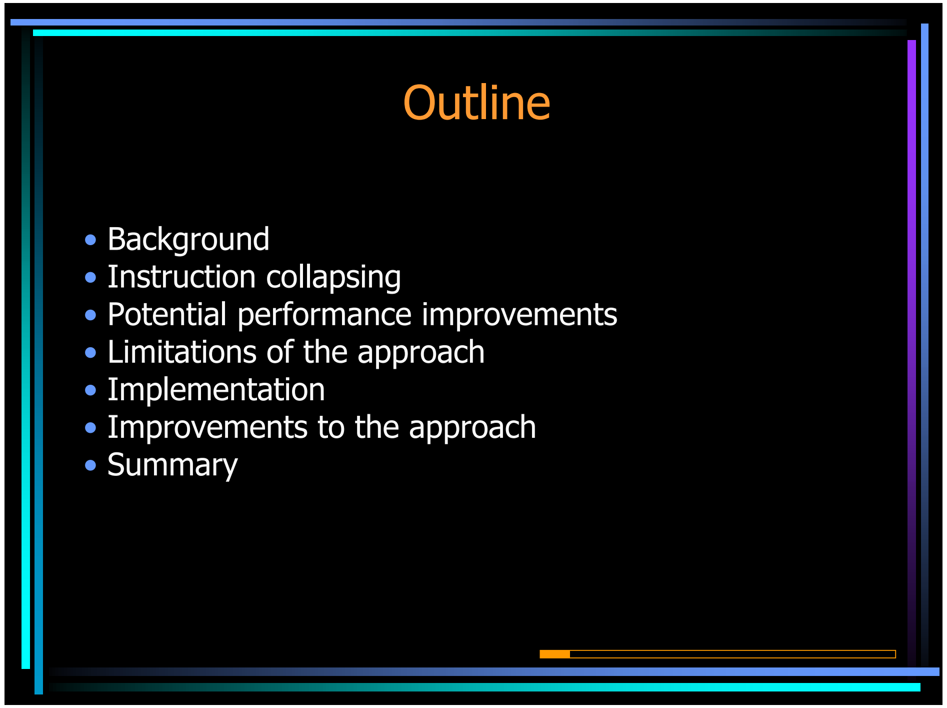
Ben Rudzyn



Disclaimer

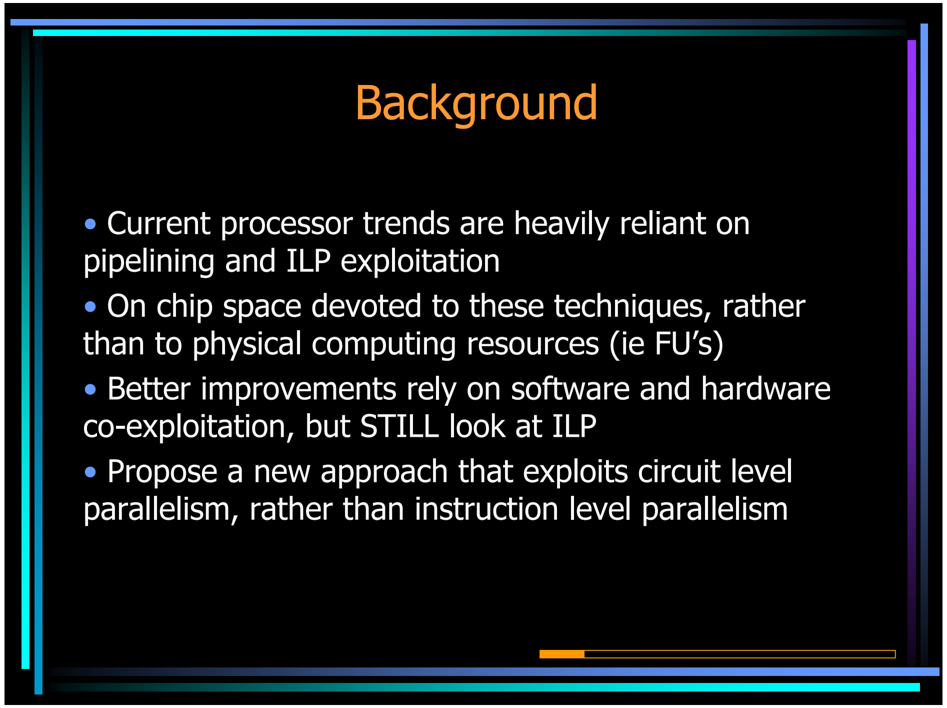
The approach presented here comes from a paper of the same title. Written by Sami Yehia and Oliver Temam of Paris XI University. Presented at the 31st Annual International Symposium on Computer Architecture (ISCA'04)

While this presentation is my own work, the methodologies, experimental results, and graphs come from this paper.



Outline

- Background
- Instruction collapsing
- Potential performance improvements
- Limitations of the approach
- Implementation
- Improvements to the approach
- Summary



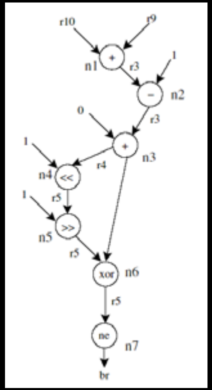
Background

- Current processor trends are heavily reliant on pipelining and ILP exploitation
- On chip space devoted to these techniques, rather than to physical computing resources (ie FU's)
- Better improvements rely on software and hardware co-exploitation, but STILL look at ILP
- Propose a new approach that exploits circuit level parallelism, rather than instruction level parallelism

Instruction collapsing

- Take a sequential set of dependent instructions
  - ILP exploitation useless
- Could implement as a *Function*
  - Can be collapsed to a combinational 2 level sum of products (OR's of AND's) or LUT

```
i1: addq r10,r9,r3     ; hdL+hdR
i2: subq r3,0x1,r3     ; hdL + hdR –1
i3: addl r31,r3,r4     ; ov=(int) result;
i4: sll r4,0x1,r5      ; ov <<1
i5: sra r5,0x1,r5      ; ((ov<<1)>>1)
i6: xor r5,r4,r5       ; ((ov<<1)>>1)==ov
i7: bne r5, continue
```

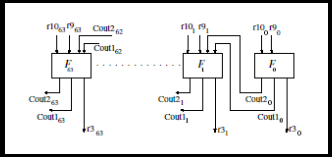


Instruction collapsing

- Exploit CLP at the cost of redundant operations

$$f_{r5}(r9,r10) = ((r9+r10-1) << 1) >> 1$$
$$f_{br}(r9,r10) = (r9+r10-1) \oplus ((r9+r10-1) << 1) >> 1)$$

- Cost: 2 64 bit inputs → 2^{128} bit truth table!!!
- One solution: implement as a set of n 1 bit operators with multiple carry propagation

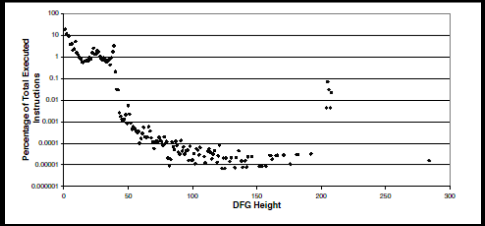


Progress

- Background ✓
- Instruction collapsing ✓
- Potential performance improvements
- Limitations of the approach
- Implementation
- Improvements to the approach
- Summary

Potential performance improvement

- Potential speedup determined by the number of collapsible dependent instructions
  - Need to identify all disjoint DFG's in the program trace
- Speed up is the average height of all DFG's
  - Avg of 1.5 for instruction traces with 1024 window

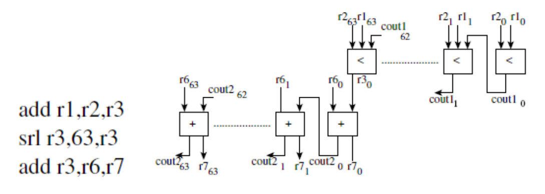


Limitations of the approach

- Number of physical inputs (register + carries)
  - Hardware operator size fixed
- Load instructions
  - Cannot be combined with dependent instructions
  - Still semi collapsible
  - Avg 24.4% of instructions
- Non collapsible instructions
  - Eg syscalls, FP divide
  - Avg 15% of instructions
- Result: only consider integer add/sub, constant shift, bit operations/manipulations and conditional branches

Limitations of the approach

- Significant bit carries



```
add r1,r2,r3
srl r3,63,r3
add r3,r6,r7
```

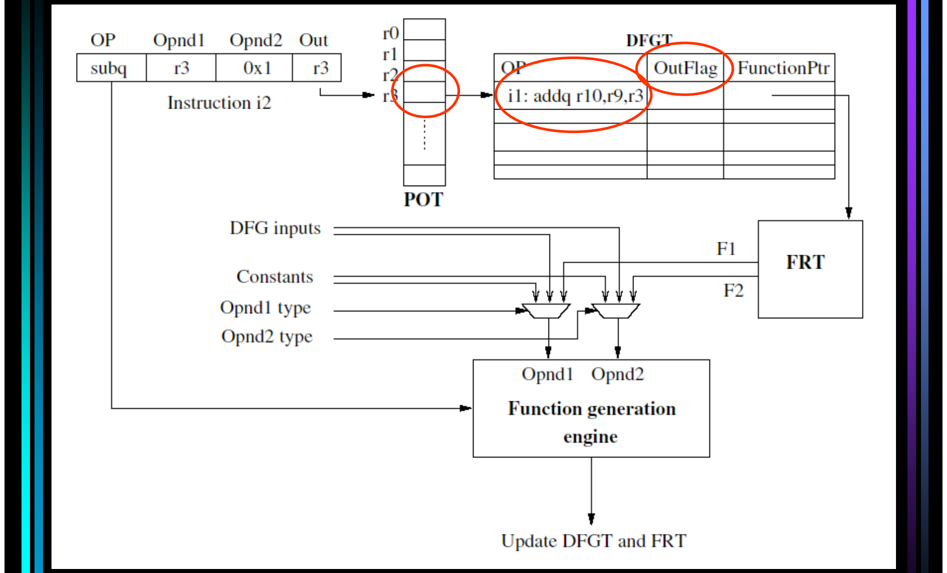
- Height limitation
  - Consider only those DFG's with height greater than the Function unit latency
  - Allows better utilisation of all FU's

Progress

- Background ✓
- Instruction collapsing ✓
- Potential performance improvements ✓
- Limitations of the approach ✓
- Implementation
- Improvements to the approach
- Summary

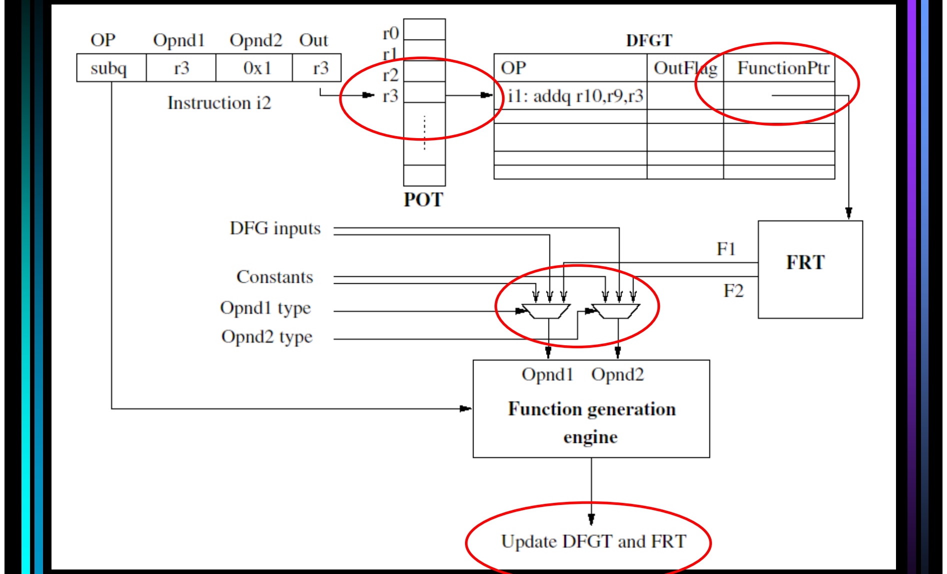
Implementation

- 4 main components
  - DFGT : Data Flow Graph Table
  - POT : Producing Output Table
  - FGE : Function Generation Engine
  - FRT : Function Repository Table



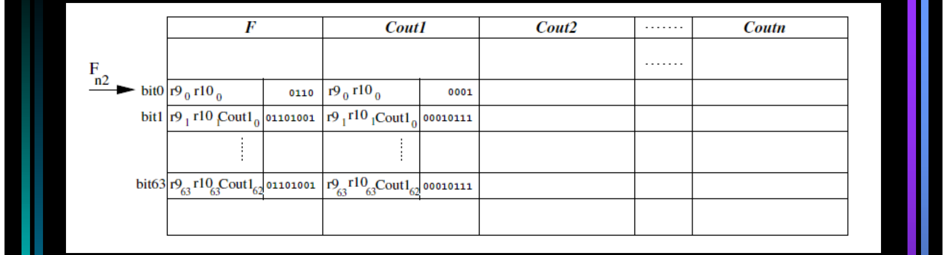
Implementation

- Output flag set to indicate which instruction is an output of the DFGT
- Use the POT to keep track of data dependencies
  - Each entry has an index into the DFGT to the instruction that produces the result for that register
  - The combination of the POT and DFGT is similar to that of the ROB, except that it is done offline
- Once an instruction is loaded into the DFGT, compose this operation with the functions producing its source operands, thus creating a more complex function



Implementation

- FGE (Function Generation Engine)
  - Three types of inputs
  - If the operand is a result of a previous instruction, send the function producing this operand as a truth table
- FRT (Function Repository Table)
  - Stores the result of each function as a 64 bit truth table (6 inputs for each function)
  - One truth table for EACH bit of the input word
  - Each entry in the DFGT contains an index to the corresponding function results in the FRT
  - Also stores the number of inputs of the truth table

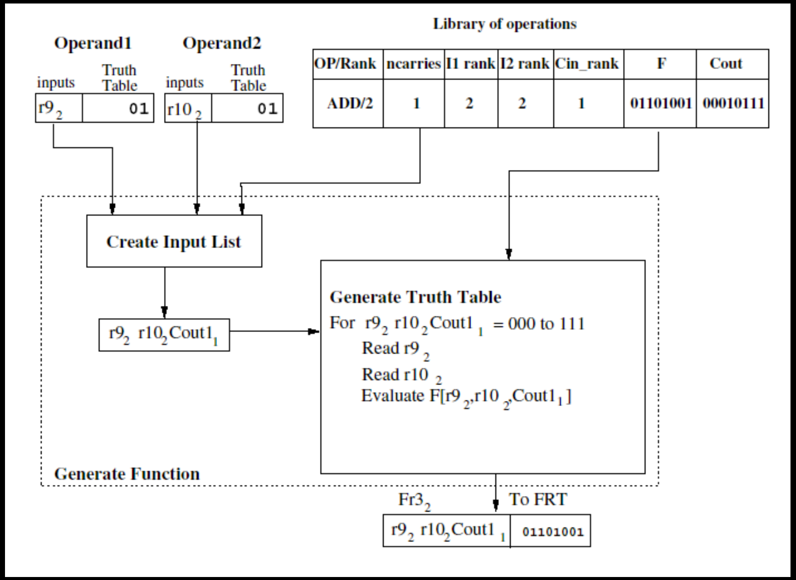


	F		*Cout1*		*Cout2*	....	*Coutn*
						....	
bit0	$r9_0$ $r10_0$	0110	$r9_0$ $r10_0$	0001			
bit1	$r9_1$ $r10_1$ $Cout1_0$	01101001	$r9_1$ $r10_1$ $Cout1_0$	00010111			
	⋮		⋮				
bit63	$r9_{63}$ $r10_{63}$ $Cout1_{62}$	01101001	$r9_{63}$ $r10_{63}$ $Cout1_{62}$	00010111			

r9	r10	Ci	F	Co
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Implementation

- How does the FGE create a new truth table from the previous ones?
  - For each combination of the inputs, it looks up the truth table of the operands
  - Uses the result to look up the truth table of the operation itself (this is stored in an additional library of operations)
  - The library also indicates if additional variables (ie carries) must be introduced
- The final function truth table is stored back into the FRT, and linked through the DFGT again



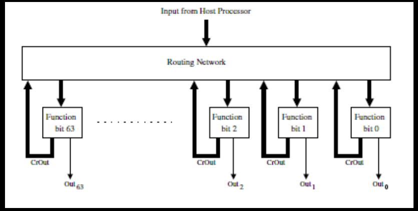
Library of operations

OP/Rank	ncarries	I1 rank	I2 rank	Cin_rank	F	Cout
ADD/2	1	2	2	1	01101001	00010111

```
Generate Truth Table
For  $r9_2$ $r10_2$ $Cout1_1$ = 000 to 111
    Read $r9_2$
    Read $r10_2$
    Evaluate F[$r9_2$,$r10_2$,$Cout1_1$]
```

Hardware implementation

- From truth table to reconfigurable Function Unit
- Function unit advantages
  - Combinational logic only
  - Single row
  - No complex interconnections
- Disadvantages
  - Significant number of inputs → large logic blocks



Hardware implementation

- Major issue
  - Overhead of dynamically building DFG's and functions on the fly
  - Assembling large traces
  - Trace → DFG → Function truth table → Macro

- rePLay framework
  - Not going into details
  - Speed up of branch resolution
  - Effect of Function delay (reconfig and process)

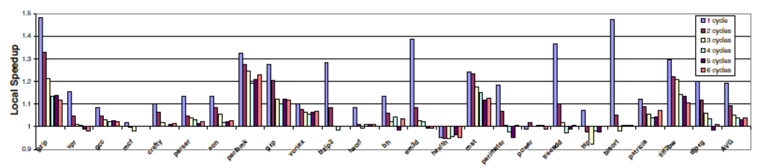


Figure 19. Local Speedup.

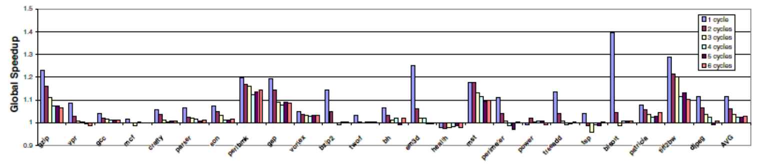


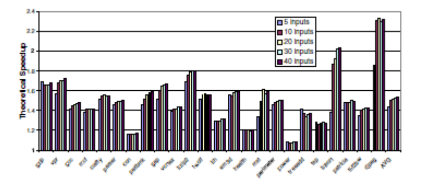
Figure 20. Global Speedup.

Progress

- Background ✓
- Instruction collapsing ✓
- Potential performance improvements ✓
- Limitations of the approach ✓
- Implementation ✓
- Improvements to the approach
- Summary

Improvements to the approach

- Increase the number of inputs?



- Increase trace window for frames?
- Alleviate load cuts through address prediction?
- Combine Functions with existing ILP techniques
  - Best of both worlds

Summary

- Exploits circuit level parallelism
- Collapse dependent instructions into 2 level combinational circuits
- Works independently of ILP
  - Targets a different set of instructions