The Performance Effects of Virtual-Machine Instruction Pointer Updates

M. Anton Ertl, TU Wien

Example: `matrix.fs`

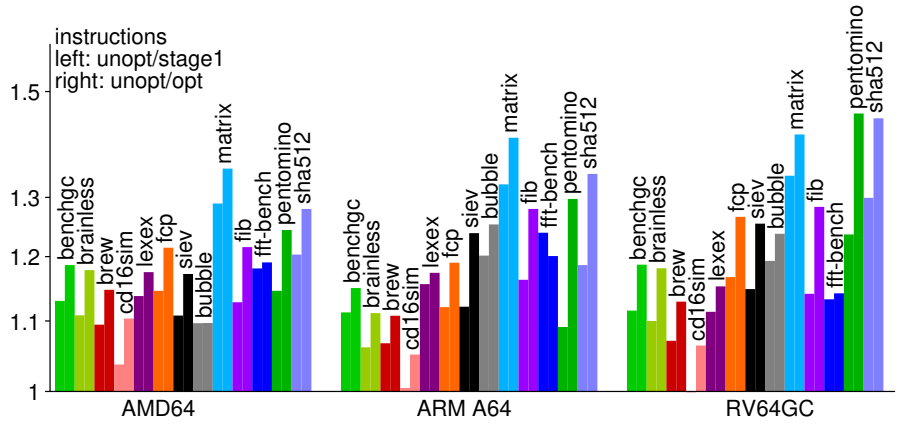
```
: innerproduct ( a[row][*] b[*][column] -- int)
  0 row-size 0 do
    >r over @ over @ * r> + >r
    swap cell+ swap row-byte-size +
    r>
  loop
  >r 2drop r>
;
```


threaded code		unopt.	opt.	threaded code		unopt.	opt.
>r	1->1	sd s7,-8(rp)	sd s7,-8(rp)	cell+	2->2	*addi ip,ip,8*	
		ld s7,8(sp)	ld s7,8(sp)			addi s0,s0,8	addi s0,s0,8
		addi sp,sp,8	addi sp,sp,8	swap	2->1	addi sp,sp,-8	addi sp,sp,-8
		addi rp,rp,-8	addi rp,rp,-8			*addi ip,ip,8*	
		addi ip,ip,8				sd s0,8(sp)	sd s0,8(sp)
over	1->2	ld s0,8(sp)	ld s0,8(sp)	lit+ 1600	1->1	ld a5,*0(ip)*	ld a5,*104(ip)*
		addi ip,ip,8				*addi ip,ip,16*	
@	2->2	ld s0,0(s0)	ld s0,0(s0)			add s7,s7,a5	add s7,s7,a5
		addi ip,ip,8		r>	1->1	sd s7,0(sp)	sd s7,0(sp)
over	2->3	mv s3,s7	mv s3,s7			addi sp,sp,-8	addi sp,sp,-8
		addi ip,ip,8				ld s7,0(rp)	ld s7,0(rp)
@	3->3	ld s3,0(s3)	ld s3,0(s3)			*addi ip,ip,8*	
		addi ip,ip,8				addi rp,rp,8	addi rp,rp,8
*	3->2	mul s0,s0,s3	mul s0,s0,s3	(loop) start	1->1	ld a5,0(rp)	ld a5,0(rp)
		addi ip,ip,8				ld a4,8(rp)	ld a4,8(rp)
r>	2->3	ld s3,0(rp)	ld s3,0(rp)			ld a3,*0(ip)*	ld a3,*128(ip)*
		addi ip,ip,8				addi a5,a5,1	addi a5,a5,1
		addi rp,rp,8	addi rp,rp,8			*addi a6,ip,8*	sd a5,0(rp)
+	3->2	add s0,s0,s3	add s0,s0,s3			beq a4,a5,end	beq a4,a5,end
		addi ip,ip,8				sd a5,0(rp)	
>r	2->1	addi rp,rp,-8	addi rp,rp,-8			ld a4,0(a3)	ld a4,0(a3)
		addi ip,ip,8				*addi ip,a3,8*	*mv ip,a3*
		sd s0,0(rp)	sd s0,0(rp)			jr a4	jr a4
swap	1->2	ld s0,8(sp)	ld s0,8(sp)			end:	end:
		addi ip,ip,8				*addi ip,a6,8*	
		addi sp,sp,8	addi sp,sp,8			sd a5,0(rp)	

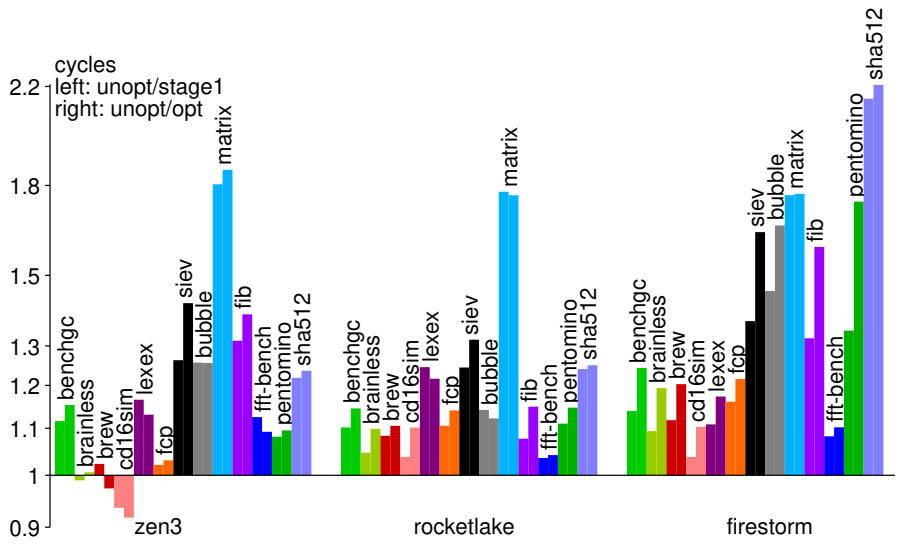
Optimize away most `ip` updates

- Normal case: Don't insert ip-update
- Remember which threaded-code cell `ip` points to
- If `ip` must be up-to-date, insert ip-update
  (Taken branch)
  Superblock end
  Calls
  non-relocatable native code
  immediate arguments (in some cases)
- Versions of ip-updates for 1–24 cells
- Versions of primitives with immediate arguments with varying `ip` offsets
  `lit call ?branch lit@ branch (loop) lit-perform lit+ does-xt`
- No architecture-specific code

Instructions



Performance (modern high-performance cores)



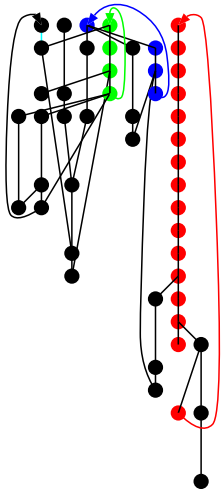
Questions

- Why is the speedup much higher than instruction reduction (for some benchmarks)?
- Why only for some benchmarks?
- Why is the speedup of sha512 bigger on Firestorm?

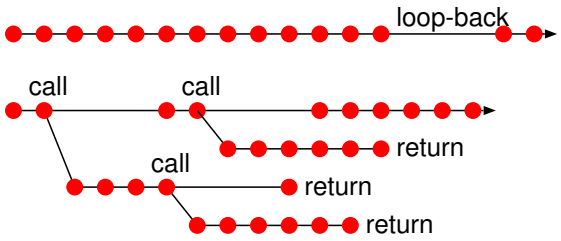
Performance components

- Branch mispredictions (solved)
- Resource limitations
- VM data dependences
- sp-update dependences
- rp-update dependences
- *ip-update* dependences

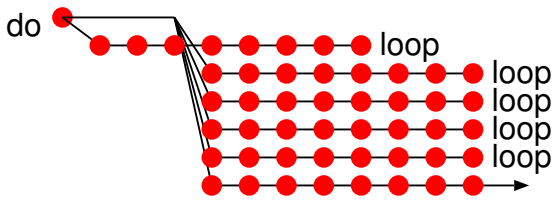
Data flow graph of `innerproduct`



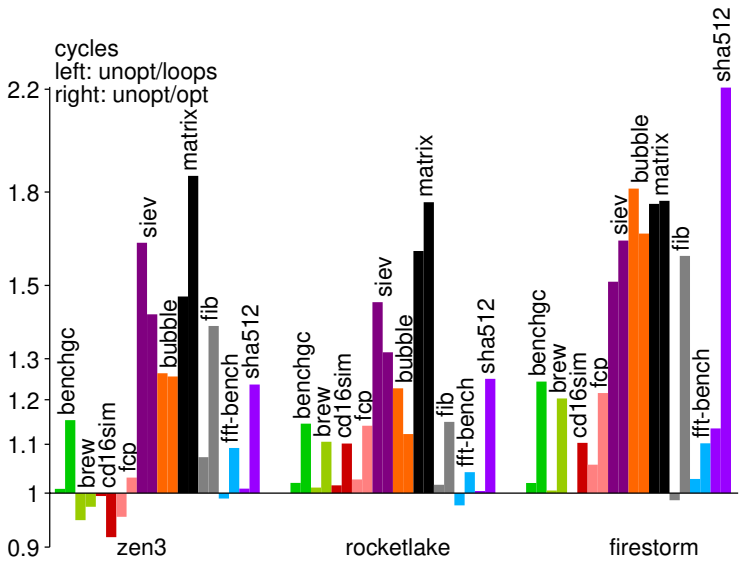
Why do we not see such speedups for all benchmarks?



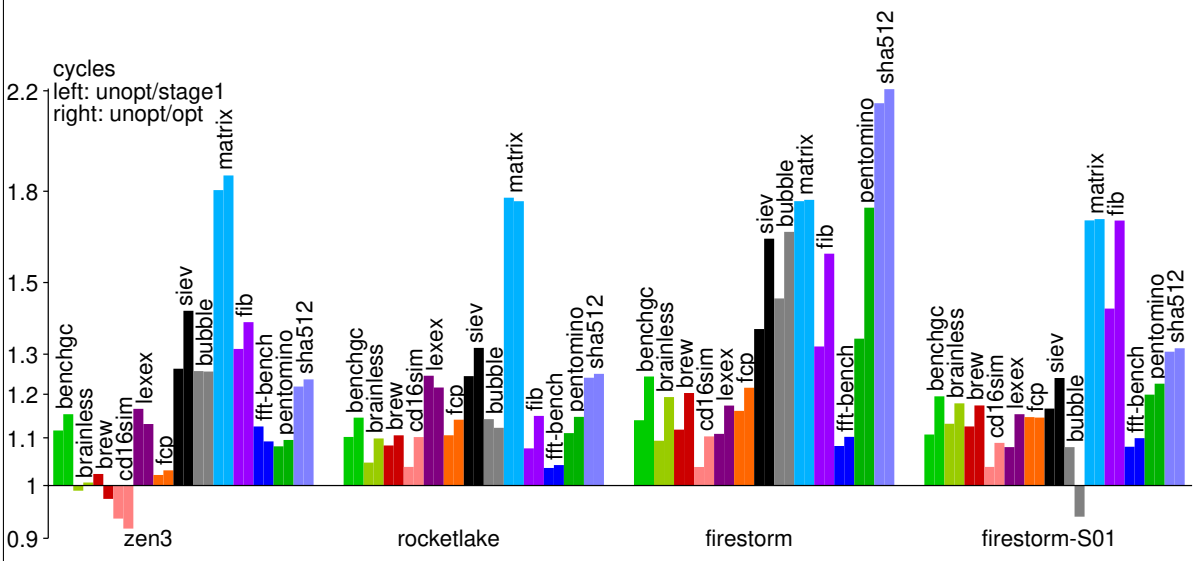
Alternative: Keep `do` loop-back address on return stack



Performance of `do` loop alternative vs. ip-update optimization



Why is the speedup of sha512 bigger on Firestorm? Stack caching!



Conclusion

- Optimizing ip updates can be done portably
- Reduces executed instructions by $\approx 1.2\times$
- Increases performance by up to $2.2\times$
  because ip-updates are the critical path in looping benchmarks
  Alternative: optimize loops
- Synergy between stack caching and ip-update optimization