Speeding Up Thread-Local Storage Access from Dynamic Libraries

Alexandre Oliva

http://www.lsd.ic.unicamp.br/~oliva/

aoliva@redhat.com oliva@lsd.ic.unicamp.br

Red Hat **University of Campinas**

March, 2008

Summary

- TLS?!?
- Dynamic libraries
- Thread-Local Storage
- Optimizations
- Performance numbers
- ARM Port
- Relaxations

Background

- Per-thread data
- Stack, automatic variables
- `pthread_[gs]etspecific`
- TLS: `__thread` variables

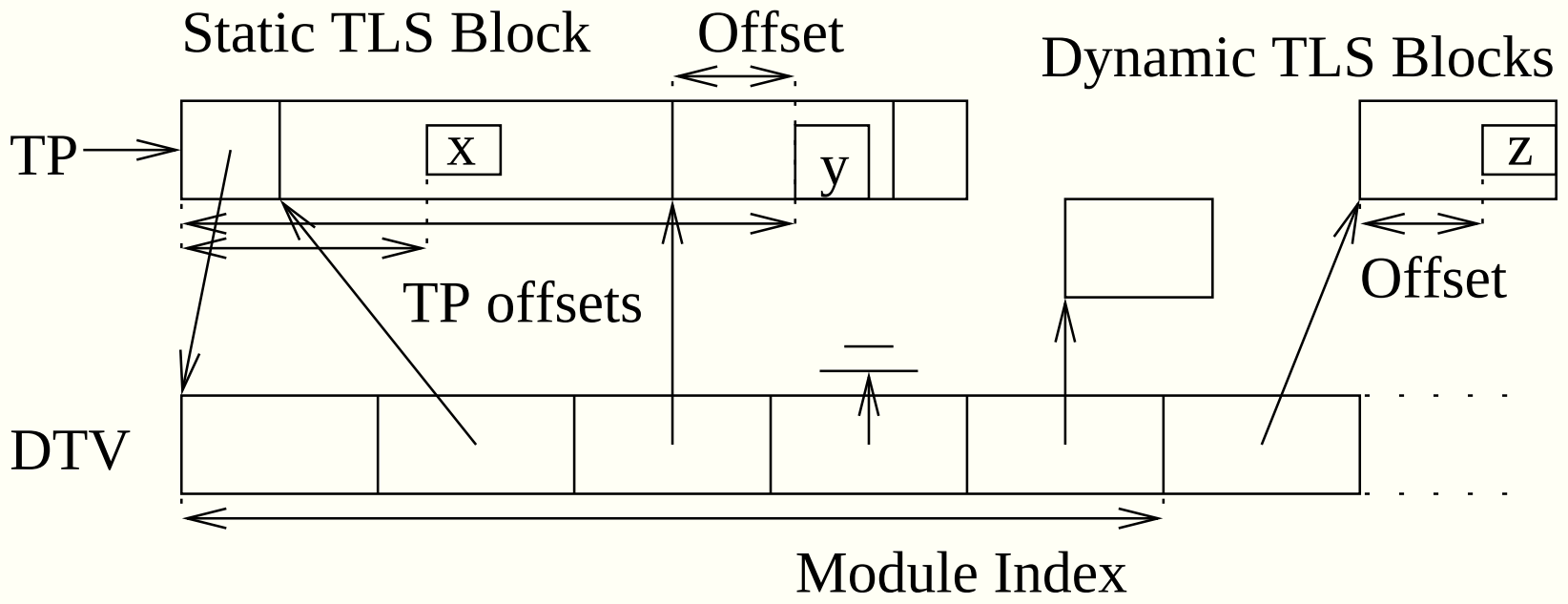
Dynamic Libraries

extern int i, g(void); int f(void) { return g() + i; }

PDC (exec)	PIC (shared lib)
	copy next pc to %ebx
	addl $_G_O_T_ - ., %ebx
call g	call g@PLT
	movl i@GOT(%ebx), %edx
addl i, %eax	addl (%edx), %eax

Thread-Local Storage

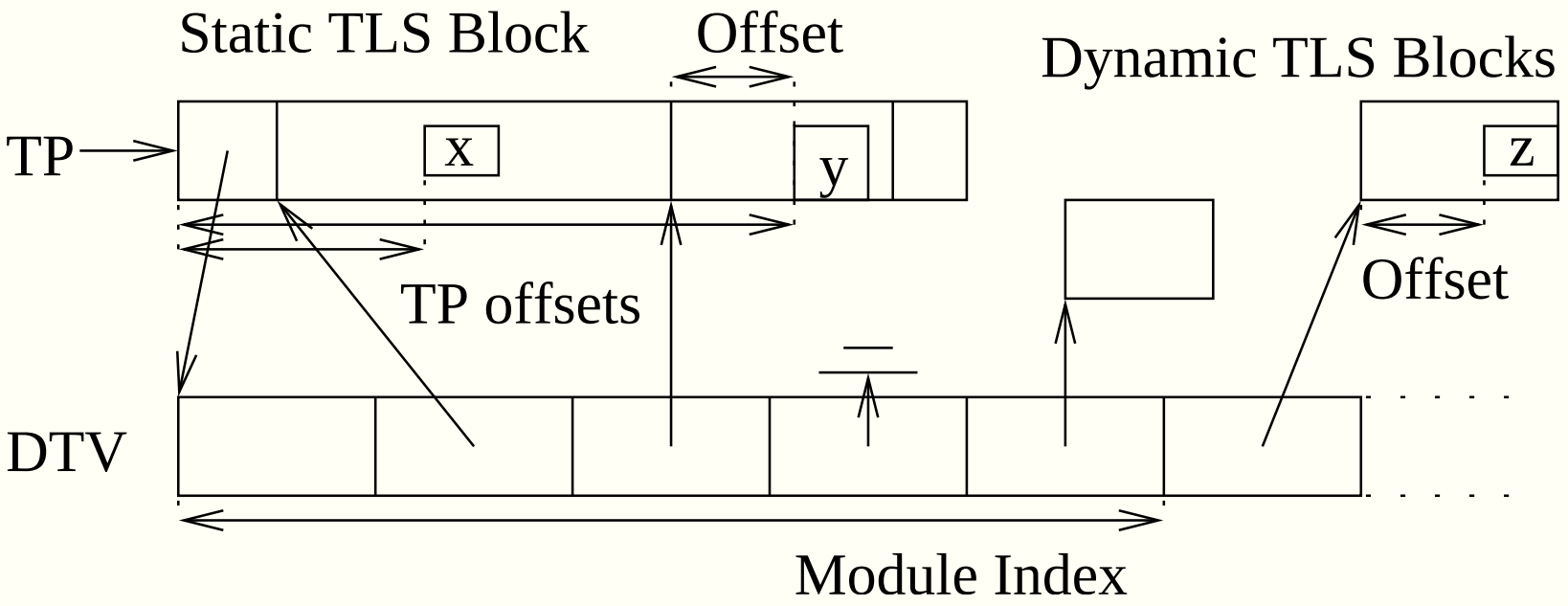
__thread int x; extern __thread int y;



Local Exec

__thread int x; int getx() { return x; }

- movl %gs:x@NTPOFF, %eax



Initial Exec

extern __thread int y; int gety() { return y; }

- movl y@GOTNTPOFF(%ebx), %eax
- movl %gs:(%eax), %eax

_G_O_T_ + y@GOTNTPOFF:

- .word y@NTPOFF

General Dynamic

__thread int z; int getz() { return z; }

- leal z@TLSGD(,%ebx,1), %eax
- call ___tls_get_addr@PLT
- movl (%eax), %eax

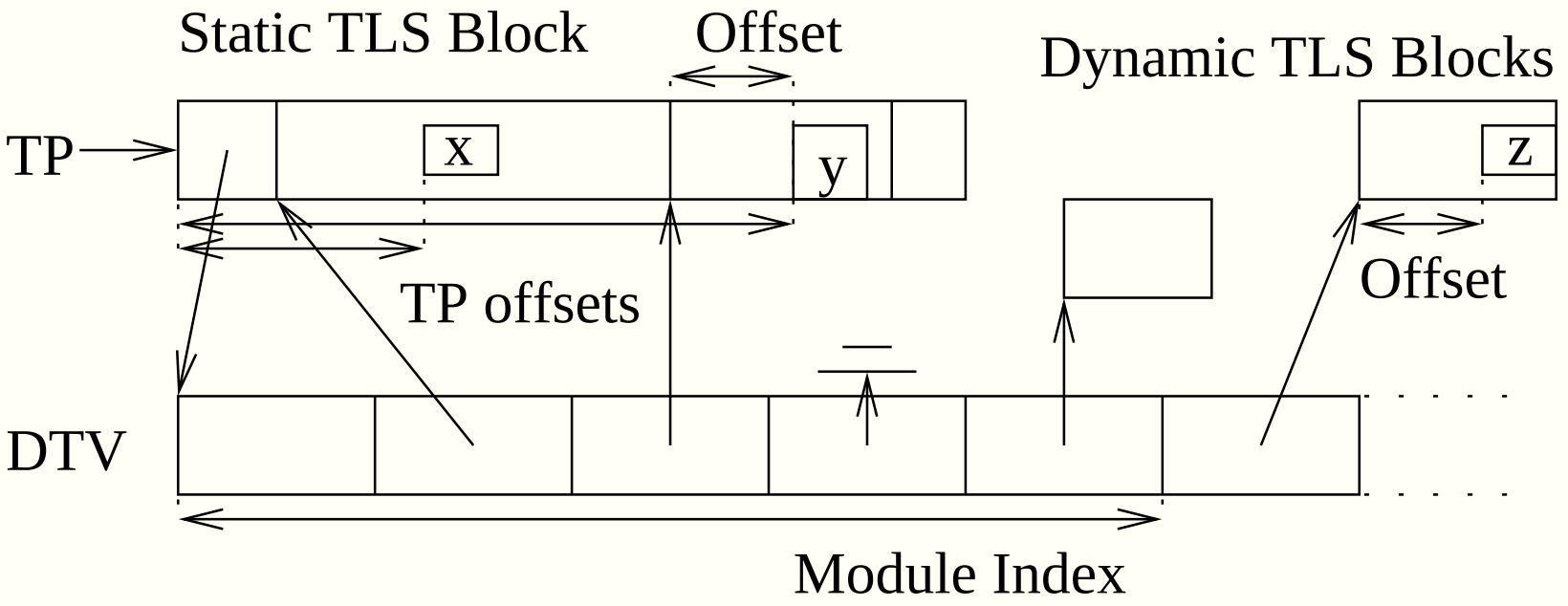
void *___tls_get_addr(struct { long index, offset; } *);

_G_O_T_ + z@TLSGD:

- .word index, offset

__tls_get_addr

- If generation count is not current, update() DTV
- If dtv[index] not allocated, allocate() it
- Return dtv[index] + offset



Local Dynamic

```
static __thread int z1, z2;
int getz() { return z1 + z2; }
```

- leal z1@TLSLDM(%ebx), %eax
- call ___tls_get_addr@PLT
- movl %eax, %esi
- movl z1@DTPOFF(%eax), %eax
- addl z2@DTPOFF(%esi), %eax

TLS Descriptor-based General Dynamic

__thread int yz; int getyz() { return yz; }

- leal yz@TLSDESC(%ebx), %eax
- call *yz@TLSCALL(%eax) ;; == call *(%eax)
- movl %gs:(%eax), %eax

_G_O_T_ + yz@TLSDESC:

- .word resolver, argument

Static Descriptor

_G_O_T_ + y@TLSDESC:

- .word sresolver, y@NTPOFF

sresolver:

- movl 4(%eax), %eax
- ret

Dynamic Descriptor

_G_O_T_ + z@TLSDESC:

- .word dresolver, dyndesc(z)

dresolver:

- If GC is current **enough** and dtv[index] is allocated, return dtv[index] + offset - TP
- Call __tls_get_addr preserving registers, subtract TP

dyndesc(z): (allocated by the dynamic loader)

- .word index, offset, generation

Lazy Descriptor

_G_O_T_ + yz@TLSDESC:

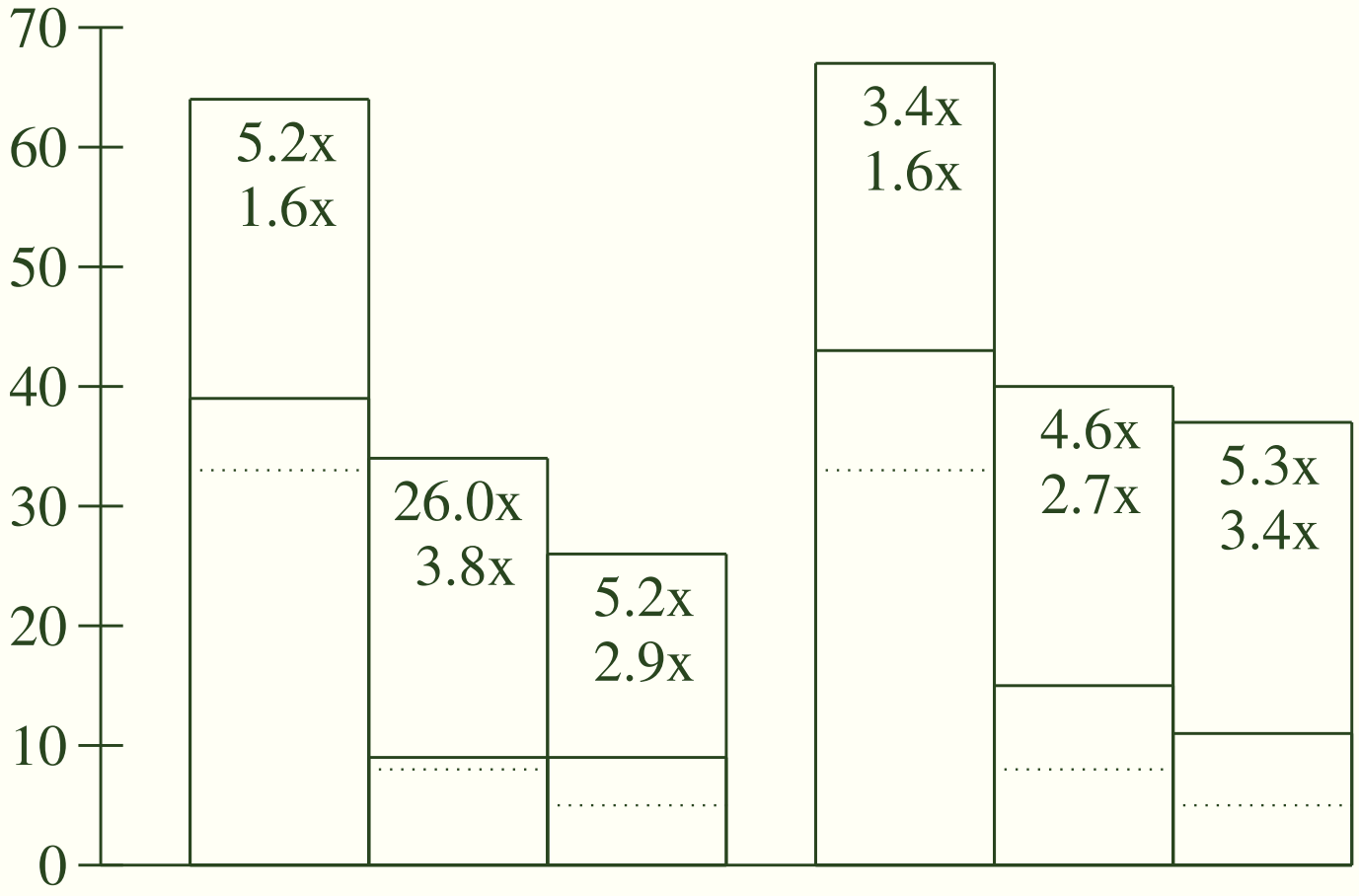
- .word lresolver, reloc

lresolver:

- Acquire loader lock
- If not resolved yet,
  - Apply relocation preserving registers
- Release lock
- Return into final resolver

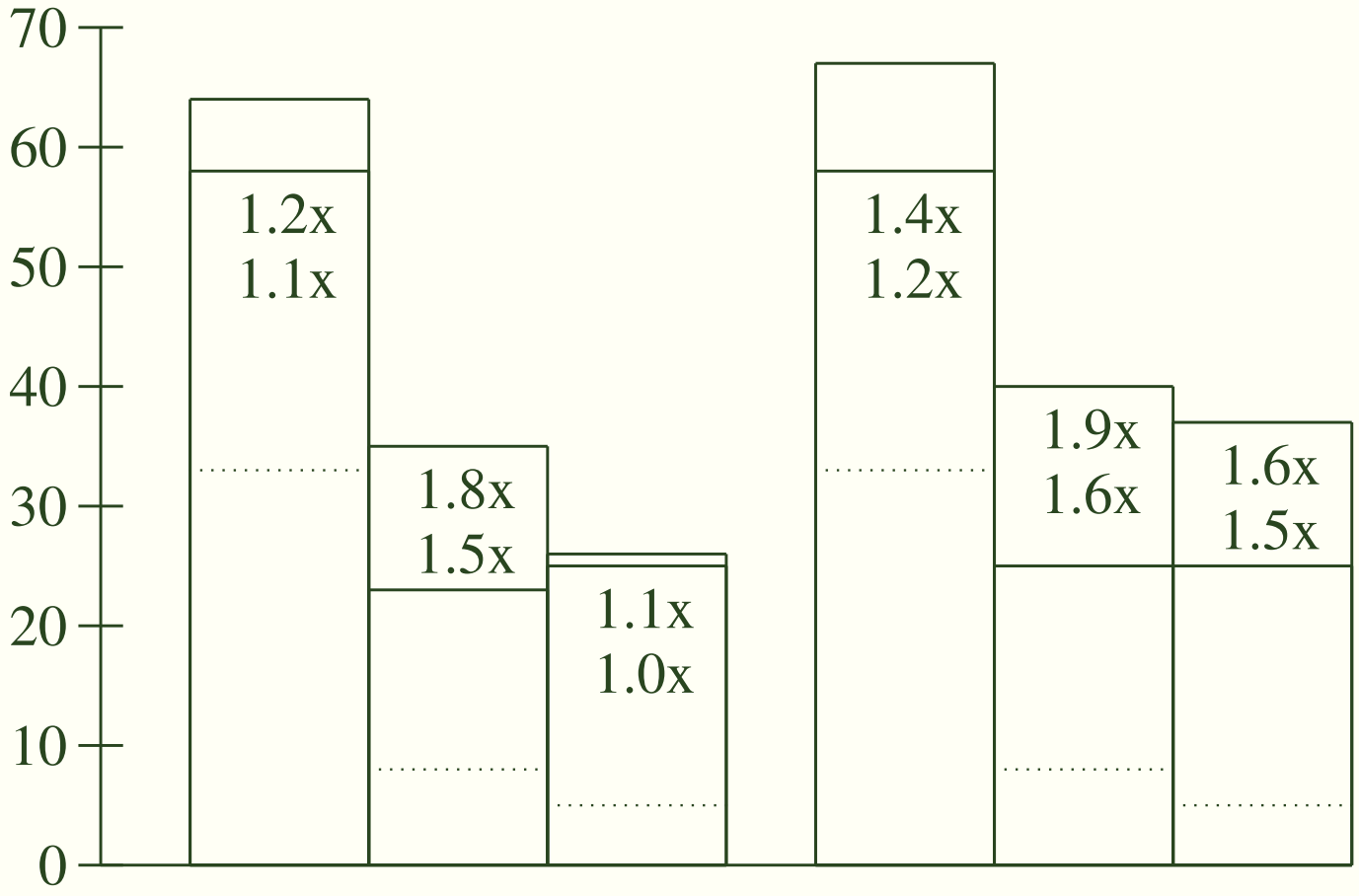
Speedups: Static

t (CK) × ((MinSt, MaxSt) × (P3, A64/32, A64/64))



Speedups: Dynamic

t (CK) $\times$ ((MinSt, MaxSt) $\times$ (P3, A64/32, A64/64))



ARM Port

	Original	Optimized
	ldr r0, .Lt0	ldr r0, .Lt0
.L1:	add r0, pc, r0	
	bl __tls_get_addr(PLT)	bl foo(tlscall)
	ldr r0, [r0]	ldr r0, [$tp, r0]
.Lt0:	.word foo(tlsgd) \	.word foo(tlsdesc) \
	+ (. - .L1 - 8)	+ (. - .L1)

ARM Port (cont)

Original	Optimized
bl tga(PLT)	bl tramp
.word foo(tlsgd) \ + (. - .L1 - 8)	.word foo(tlsdesc) \ + (. - .L1 - **4**)

add ip, pc, tga(got)[24:31]
add ip, ip, tga(got)[16:23]
ldr pc, [ip, tga(got)[0:15]]!
	add r0, lr, r0
	ldr r1, [r0, #4]
	bx r1

Relaxations

GD	IE	LE
ldr r0, .Lt0 bl foo(tlscall)	ldr r0, .Lt0 ldr r0, [pc, r0]	ldr r0, .Lt0 nop
.word foo(tlsdesc) \ + (. - .L1 - 4)	foo(gottpoff) \ + (. - .L1 - 8)	foo(tpoff) + 0

Inlining the Trampoline

ldr rt, .Lt1	ldr rt, .Lt1	ldr rt, .Lt1
add rx, pc, rt	add rx, pc, rt	mov rx, rt
ldr ry, [rx, #4]	ldr ry, [rx]	nop
mov r0, rx	mov r0, rx	mov r0, rx
blx ry	mov r0, ry	nop
.word foo(tlsdesc) \	foo(gottpoff) \	foo(tpoff)
+ (. - .L1 - 8)	+ (. - .L1 - 8)	+ 0

Conclusions

- Major speedups in the most common case
- Small speedups even in the dlopen case
  - Compiler improvements could reduce them
  - Generation count
  - Calling conventions
- Smaller code, same data space in static case
- Lazy relocation
- Ported to x86, x86_64, ARM and FR-V