Copyright © 2004 Intel Corporation. All Rights Reserved.

Compiler Optimizations for TPC-C workloads on Linux*

Knud Kirkegaard
Intel Compiler Lab, Santa Clara

Gerolf Hoflehner, Daniel M Lavery,
Yong-fong Lee, Wei Li, Rod Skinner

March 21, 2004

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Agenda: Compiler Technology

- Code Scheduling and Instruction Selection
- Function Layout Optimizations
- Data Layout Optimizations
- RSE Traffic Reduction
- Setjmp() Overhead Reduction
- Linux* Software Conventions

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Code Scheduling and Instruction Selection

- Schedule for code size
- Tune control- and data speculation
  - no use speculation
- Reduce/Remove synchronization overhead
  - Unordered volatile data
  - _memory_barrier() intrinsic

Copyright © 2004 Intel Corporation. All Rights Reserved.

Unordered volatile data ("volatile lite")

- Volatile qualifier
  - Store local data in memory
  - Memory instructions can't move
- Suppress .rel/.acq completers
- New compiler option(s):
  - Linux*: -mno-serialize-volatile

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Compile Barrier Intrinsic

- Prevents both upward and downward motion of memory instructions.
- Non-memory instructions may be scheduled across the barrier.
- Automatic data references may be allocated in a register across the barrier.
- This intrinsic does not generate code.
- Needs changes in source code.
- Prototype: void __memory_barrier()

Copyright © 2004 Intel Corporation. All Rights Reserved.

Agenda: Compiler Technology

- Code Scheduling and Instruction Selection
- Function Layout Optimizations
- Data Layout Optimizations
- RSE Traffic Reduction
- Setjmp() Overhead Reduction
- Linux* Software Conventions

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Function Layout Optimizations

- Function inlining
  - Use >200 hottest modules for inter-procedural function inlining
- Function splitting
  - Partition a function into hot and cold section based on feedback profiling data
- Function grouping
  - Layout based on execution frequencies

Better I-Cache Utilization

Copyright © 2004 Intel Corporation. All Rights Reserved.

Data Layout Optimizations

- Use read-only data section
  - Move strings and constants to read-only section instead of data section
- Local data layout
  - Sort local data on the memory stack based on frequency and size
- Global data layout
  - Sort global data based on "closeness" derived from execution feedback profiling

Better D-Cache Utilization

Copyright © 2004 Intel Corporation. All Rights Reserved.

Agenda: Compiler Technology

- Code Scheduling and Instruction Selection
- Function Layout Optimizations
- Data Layout Optimizations
- RSE Traffic Reduction
- Setjmp() Overhead Reduction
- Linux* Software Conventions

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

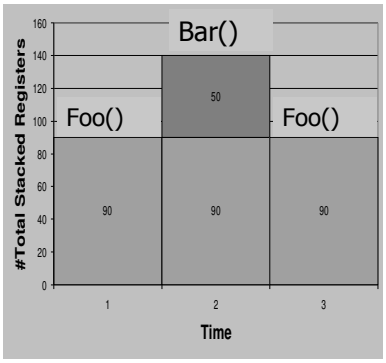
Register Stack Engine (RSE) traffic reduction

- Multiple alloc for calls
- Share register stack with callee
- Overlap results in overall decrease of RSE traffic
- Cost: Increase in code size is negligible

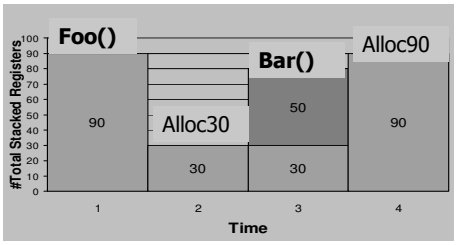
Copyright © 2004 Intel Corporation. All Rights Reserved.

Optimize the RSE Usage

Unoptimized RSE



Optimized RSE



Register Stack for Caller and Callee Overlap

Copyright © 2004 Intel Corporation. All Rights Reserved.

Agenda: Compiler Technology

- Code Scheduling and Instruction Selection
- Function Layout Optimizations
- Data Layout Optimizations
- RSE Traffic Reduction
- Setjmp() Overhead Reduction
- Linux* Software Conventions

* Other names and brands may be claimed as property of others

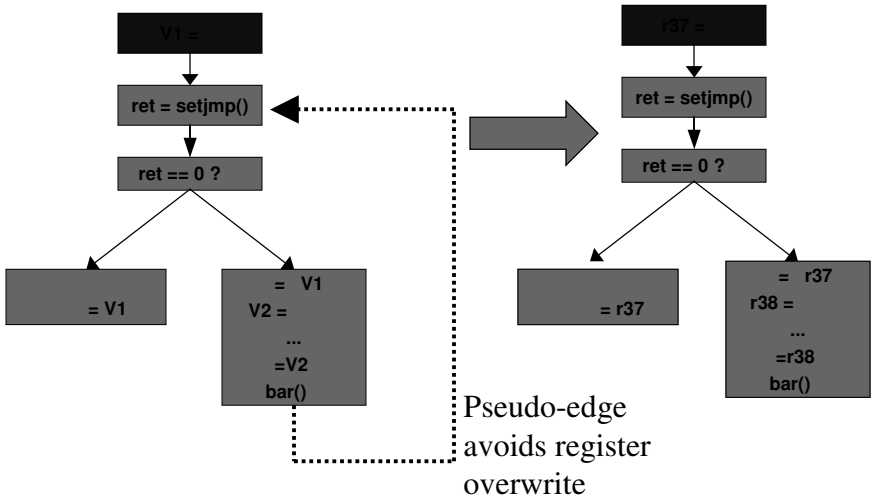
Copyright © 2004 Intel Corporation. All Rights Reserved.

Setjmp() Overhead Reduction

- Limit floating-point register usage to non-rotating scratch floating-point registers f6-f15
  - Avoid saving/restoring of rotating and preserved floating-point registers in jmp_buf buffer
- Assign integer stacked registers r32-r127 to cross-call lifetimes

Copyright © 2004 Intel Corporation. All Rights Reserved.

Stacked Registers across setjmp() calls



Copyright © 2004 Intel Corporation. All Rights Reserved.

Stacked registers across setjmp() calls: Summary

- Reduces spill/fill
- Reduces memory stack size
- Reduces code size
- Eliminate spills/fills of callee preserved int registers (r4-r7) at function entries and exits
- Costs : Increase in RSE traffic

Exploit Itanium® Architecture

Copyright © 2004 Intel Corporation. All Rights Reserved.

Agenda: Compiler Technology

- Code Scheduling and Instruction Selection
- Function Layout Optimizations
- Data Layout Optimizations
- RSE Traffic Reduction
- Setjmp() Overhead Reduction
- Linux* Software Conventions

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Linux* Software Conventions: Data

- Default behavior for global data
  - Addressed through linkage table
  - Position Independent
  - Allows for symbol preemption
  - All preemptable symbols are in .data
    - Only static data is in .sdata
  - 2 levels of indirection
    - add r3 = @ltoff(data),gp
    - ld8 r2 = [r3]
    - ld4 r8 = [r2]

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Linux* Software Conventions: Functions

- Default behavior for global functions
  - All external functions are preemptable
  - Need to save and restore gp around calls as we may leave the gp domain (load module)

Extra Cost Because of Data/Function Preemption

* Other names and brands may be claimed as property of others

Copyright © 2004 Intel Corporation. All Rights Reserved.

Linux* Preemption Model

- All references to global data and functions can be optimized unless marked as shared (equiv. dllimport)
  - Global data can be referenced
    - movl r3 = ltconst(data)
    - ld4 r8 = [r3]
- All calls to functions can avoid save/restore of gp unless marked as shared

Avoids Extra Cost for Data and Functions