Dynamic Profile-Guided Optimizations (DPGO) in Managed Runtimes

Greg Eastman
Shirish Aundhe
Robert Knight

Software and Solutions Group
Enterprise Platform Group
Intel Corporation

March 21, 2004

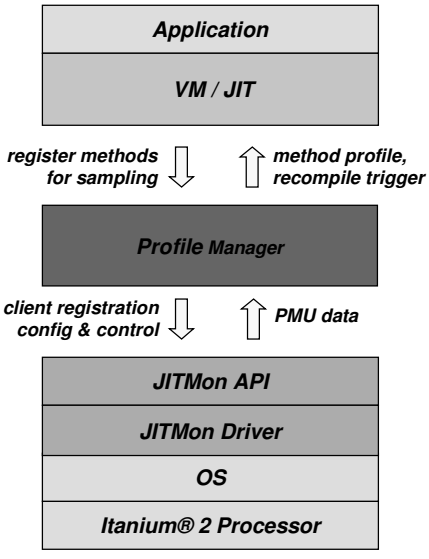
Overview

- Objectives
- Architecture
- JITMon
- Profile Manager
- Profile Details
- Optimizations
- Results

Objectives

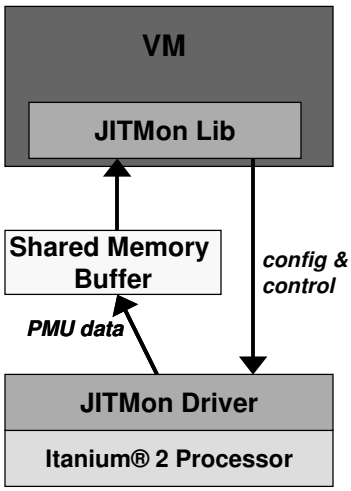
- Enable DPGO using processor Performance Monitoring Unit (PMU) on Intel Itanium® 2 Processor
  - Develop DPGO SW infrastructure
  - Demonstrate DPGO-based performance gains in a commercial JVM/JIT
  - Diffuse DPGO technology

DPGO Architecture



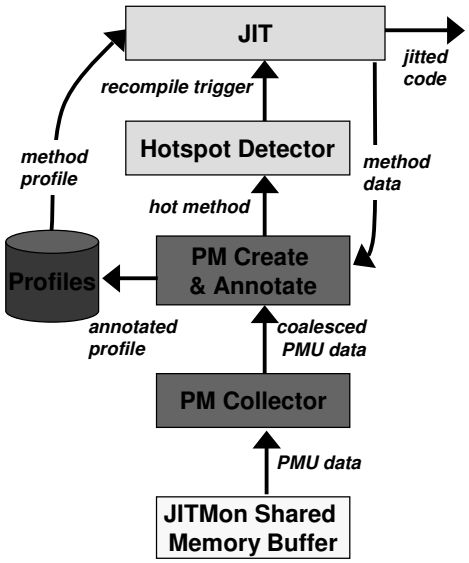
- Profile Manager
  - APIs for profile creation, annotation, use
  - IR-to-address mapping
- JITMon Sampling API & Driver
  - Collects processor performance data
  - Configures & controls data collection
  - Delivers PMU data to VM client

JITMon Details



- Platform/OS Independent
- Data delivered to client at runtime
- Selectable data
- Collection configuration can be modified at runtime
- Filtering by PID, TID, & virtual address range
- Minimal overhead
  - ~0.1% for sample collection

Profile Manager Details



- Profile Manager (PM) Collector registers as JITMon client
- JIT calls PM API to create & map method profiles
- JIT registers method for sampling with Collector
- Collector reads PMU data, coalesces by method
- Collector sends data to Profiler when method sampling threshold reached
- Profiler annotates profiles with sampling data, triggers recompile
- PM API provides profiles to JIT during compilation

DPGO Profiles

- Call profile
  - Profile of call sites
  - Annotated with branch trace buffer (BTrB) data
  - Used for profile-guided inlining & guarded devirtualization
- Edge profile
  - Profile of branch IPs/targets
  - Annotated with BTrB data
  - Collects edge counts, mispredicts; computes branch probabilities, basic block counts
  - Used for block reordering, method splitting, scheduling and other hot path optimizations
- Load miss profile
  - Profile of load instructions
  - Annotated with DEAR data (IP, latency)
  - Used to guide data prefetching

Compilation Model

- 1st compile – fast
  - No optimization
  - Call profile created
- 2nd compile
  - Hot methods
  - Optimized
  - Call profile used
  - *Edge and load profile created*
- *3rd compile*
  - *Edge and load profile used*

Profile Example – Call Profile

Code:

```
AClass.foo(S)V (0x000006fbf7244000 – 0x000006fbf74a5ff0)
…
0x000006fbf74a5ea0    br.call rp = b6;;           //indirect call
…
0x000006fbf74a5ee0    br.call rp = $+0x592e0;;    //ip-rel call
…
0x000006fbf74a5f40    br.ret rp ;;                //return
…
0x000006fbf74a5f60    br.call rp = $+0x28b360;;   //ip-rel call
```

Call Profile – Created and Mapped

IR_handle	Instr Address
48	0x000006fbf74a5ea0
129	0x000006fbf74a5ee0
182	0x000006fbf74a5f40
210	0x000006fbf74a5f60

indirect call: might have multiple targets

return: we profile ret instr so that we can express call frequency based on # times method called (approximated by # times method returned)

Call Profile – Annotated with BTrB Data

indirect call: has multiple targets, has a dominant target & is hot call site

IR_handle	Instr Address	Call Target	Count
48	0x000006fbf74a5ea0	0x000006fbf73b0480	198
		0x000006fbf73b1380	11
		0x000006fbf73b2300	4
129	0x000006fbf74a5ee0	0x000006fbf7521bc0	332
182	0x000006fbf74a5f40	0x000006fbf7518240	58
		0x000006fbf73f3ba0	23
210	0x000006fbf74a5f60	0x000006fbf75a7140	9

hot call site

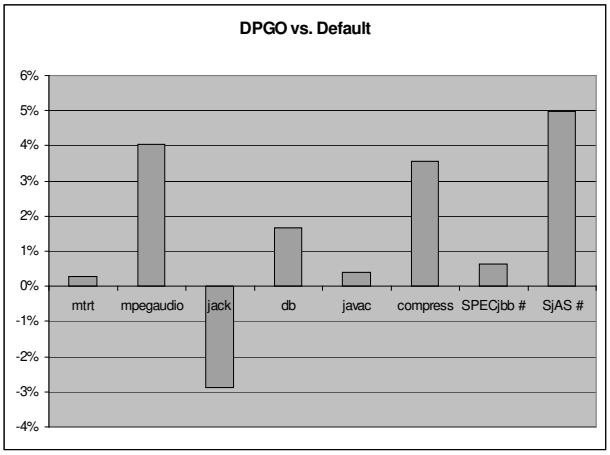
Use of Call Profile in Optimization

- Uses call frequency to help prioritize inline candidates
  - frequency(call site) == call-site count / ret edge count
- Call frequency is scaled at inlined call site
  - frequency(call site) * frequency(inlined call site)
- When an indirect call is hot and has a dominant target, generate a guarded devirtualization
  - "Domination threshold" is configurable, but should be high (e.g. >= 75%)

INTERNAL Results: Default vs. DPGO

Platform: 4x1.5 GHz Intel Itanium® 2-based server, Windows 2003 Server

DPGO version uses profiles for inlining and guarded de-virtualization



* All the results are from internal test runs only.

#: SjAS2002:SPECjAppServer2002. SPECjbb: SPECjbb2000

Benefits of PMU profiling

- **Low overhead**
  - **Cheaper than software profiling**
- **Accurate**
- **Enables profiles & optimizations that cannot be achieved with SW-based methods**
  - **Branch path w/mispredicts**
  - **Icache & ITLB miss**
  - **Dcache & DTLB miss**
- **Provides better quantitative measures of "hotness"**
  - **Instantaneous hotness ratings**
  - **Use to focus optimization & scale compilation effort**

Conclusions

- PMU-based profiling on Intel Itanium® 2 processor is a viable model for managed runtimes
- The approach shows potential for significant performance benefit
- More work ahead to make the infrastructure widely available on platforms of interest