Performance in the Age of Trustworthy Computing

Ben Zorn
PPRC
Microsoft Research
“Six months ago, I sent a call-to-action to Microsoft's 50,000 employees, outlining what I believe is the highest priority for the company and for our industry over the next decade: building a Trustworthy Computing environment for customers that is as reliable as the electricity that powers our homes and businesses today.”

Bill Gates
Executive Memo, 7/18/02

(emphasis mine)

Trustworthy = secure, reliable, available, private, etc.
TwC Research on the Rise

PLDI Papers by Type

Papers Published

Year

1999 2000 2001 2002 2003

Mem Mgmt
Optimization
Analysis
Security, Reliability+
Proebsting’s Law and other Doubts

- Moore’s Law states roughly that advances in hardware double computing power every 18 months
- “Compiler Advances Double Computing Power Every 18 Years” - Todd Proebsting, Microsoft Research
  - “Perhaps this means Programming Language Research should be concentrating on something other than optimizations. Perhaps programmer productivity is a more fruitful arena.”
    http://research.microsoft.com/~toddpro/papers/law.htm

- Other doubts about performance and optimization research
  - “Is Code Optimization Research Relevant?” Bill Pugh, U. Maryland
  - “Systems Software Research is Irrelevant” Rob Pike, Bell Labs
Exponential Growth is Hard to Beat…

Relative CPU Performance (log scale)

Data from *Computer Architecture: A Quantitative Analysis (3rd ed.*) by Hennessy and Patterson

Ben Zorn, PPRC
Performance is Dead, Long Live Performance!

- A revolution is happening, but…
- Performance is **not** a solved problem

Outline for rest of talk
- The Memory Wall and Efforts to Climb It
  - Memory latency
  - Optimizing layout to reduce disk I/O
- Challenges and Opportunities of Managed Code
- Concurrency (I wish I had time…)
Revisiting Moore’s Law…

CPU / Memory Performance Gap (log scale)

Increases:
Memory = 1.07x / year
CPU = 1.55x / year (since 1986)

Data from *Computer Architecture: A Quantitative Analysis (3rd ed.)* by Hennessy and Patterson
Caches Hide Many Cycles of Latency

Data from Dileep Bhandarkar, Intel Architect, PACT 2002 Keynote Address
“Parallelism in Mainstream Enterprise Platforms of the Future”
A Case Study – Optimizing Working Set

- Relative cost of I/O is enormous
  - 40,000,000+ cycles per page fault
  - Much user-perceived latency is disk-related

- Overview
  - PPRC and our approach
  - Improving code locality via reordering with profiles
  - Results
  - Process considerations

- Work of Hoi Vo’s Binary Technologies (BiT) group
What is PPRC?

- PPRC – Programmer Productivity Research Center
  - Amitabh Srivastava, Director
  - Focus on improving software development process
  - Areas: performance, correctness, compilation, tools
  - Approach
    - Build flexible infrastructure on which to layer tools, research
    - Build strong interactions with product teams by focused solutions
    - Used knowledge of important problems to drive infrastructure and further research
  - Successes
    - Vulcan – binary instrumentation
    - PREfix – static analysis for error detection
Code Does Matter

Desktop Application Working Sets

- Word XP
- Internet Explorer
- Excel XP

Application

4 KByte Pages

Other Data
Mapped
Teb
Stack
Heap
Code/SData
Other System
PTE
Improving Code Locality

- Basic idea – use profiles to direct code placement
  - Separate hot/cold functions, basic-blocks
  - Impact at page level, cache level
  - Static data can be placed with code where used

- Profile methodology
  - Separate instrumented build to gather profiles
  - Requires mechanisms to integrate profiles from different scenarios, weight them
  - Impact on build process cannot be ignored
Function Reordering

Function A
Function B
Function C
Function D
Function E
Function F

Page Boundary
Results

Foxpro 8.0 Working-Set

- Function Reordering
- Original Order
Function Separation

BB 1
BB 2
BB 3
BB 4
BB 5
BB 6

Live Code

Ben Zorn, PPRC
Results

Foxpro 8.0 Working-Set

- Function Reordering
- Func Ord + Separation

Page Faults vs. Time
Making It Work for Real

- Must be well integrated into build process
  - Different for every major group
  - Vulcan technology key to widespread adoption

- Time budget
  - “Compile -> Profile -> Opt” process rarely fits within time constraint
  - Profile rarely matches the same build

- Multiple platform support

- Serviceability
  - Debuggers work after code reordering
  - Patching
Managing Profile Data

- Organizing scenarios
  - Startup important in reducing delay
  - Phases associated with typical uses (print, spell check, etc.)

- Stale profile data
  - Collecting new profiles takes lots of time
  - Delaying the build cycle is unacceptable
  - Solution: profile propagation via binary matching
    - Most profile data remains similar between builds
Data Locality Research

- Data presents additional challenges
- Trishul Chilimbi – Daedalus Project
- Goal – identify opportunities to improve data locality and exploit

Contributions
- Hot data streams data abstraction (PLDI’01)
- Bursty-tracing measurement approach (FDDO’01)
- Prefetching hot data streams (PLDI’02)

Runtime Analysis and Design (RAD) group
  - [http://research.microsoft.com/rad](http://research.microsoft.com/rad)
  - [http://research.microsoft.com/~trishulc/Daedalus.htm](http://research.microsoft.com/~trishulc/Daedalus.htm)
So What about TwC?

- Question: What software technology is likely to have the most impact on computing in the next 5-10 years?
- My answer: Managed code
What is Managed Code?

- **Managed code** =
  - Code executed by the Common Language Runtime (CLR)
  - Provides metadata to allow the CLR to
    - Locate methods encoded in assembly modules
      - Dynamic loading with interface type checking
    - Store and retrieve security information
      - Implement a security model
    - Handle exceptions
    - Walk the program stack
    - Garbage collect the heap

* As defined by [.Net Glossary](http://example.com/netglossary)
Impact of Managed Code

- Performance implications
  - Pointers = abstraction (less direct control)
  - GC has global properties
  - Runtime metadata continually present, referenced
  - Large, feature-filled class frameworks
Shift in Platform

- Should most software be managed?
  - Historically, transitions from asm to C, C to C++, and now C++ to Java / C#
  - Transition to Java / C# in progress but stalled
    - Where does most Java code run? Why?
    - Just a matter of time or technology?

- Should most interfaces be managed?
  - Class libraries a start – what about OS APIs?

- These are not hypothetical questions
Managed Code on the Client

- Managed code research is mature...
  - Many Java implementation papers since 1995
  - SPECJVM benchmarks in widely used, cited
  - New GC research after 40+ years!

- However
  - Increasing client-side managed code
  - Client-side performance issues less understood
  - Opportunities for research + product impact
CLR Platform Research Opportunities

- C# / CLR / .Net available, used on clients
  - Caveat: in transition 1.0 -> 1.1 -> Whidbey (1.2)
- Sizeable applications written
  - HeadTrax (see next slides)
  - FxCop, clrprofiler (download from gotdotnet.com)
- Rich profiling API exists in CLR, Windows
  - Hook calls, returns, allocations
  - Easy integration with Windows perfmon APIs, tools
  - clrprofiler written in C#, sources available
The HeadTrax Experience Report

- **HeadTrax study** (Ovidiu Platon, July 2003)
  - Multi-tier internal MS app manages HR information
  - Client / server - focus on client experience
  - Client configuration: 128 Mb, 1 GHz CPU

- **Implementation**
  - Client written in C# with .Net Framework 1.1
  - Network interaction via web services and database APIs
  - Security important – strongly signed binaries, encryption

- **Preliminary numbers (startup)**
  - Cold start 23 seconds
  - Warm start 10 seconds

How they Improved Performance

- Changes performed
  - Made web service calls asynchronous
  - Cache data locally
  - Lazy instantiation of proxies
  - Show UI before populating
  - Results: cold 23 -> 10 secs, warm 10 -> 8 secs

- Changes proposed
  - Merge assemblies, DLLs
  - Merge threads
  - Use thread pool
What We can Learn from This

- 10 seconds is still a long time to wait
  - 1500 16+ Kb chunks read from disk at 6 ms / seek
- Logical and physical organization are at odds
  - E.g., 21 assemblies, 50 DLLs for 1 app
  - Databases figured this out long ago
  - Determining “correct” granularity is tough
    - What choices do systems provide? How easy to use?
    - Performance at odds with logical and physical isolation
- XML serialization uses reflection, C# compiler
- Eclipse faces many similar issues
- Pre-JIT is important (what is it?)
What is Pre-JIT (aka Ngen)?

- Pre-JIT is ahead-of-time compilation
  - Generates high-quality native code
  - Reduces runtime checking required across interfaces
  - Opportunities for placement of code and static data

- Ngen represents one choice in design space
  - Full runtime solutions not proven (esp. on client)
  - Best solution employs thoughtful integration of
    - Compiler, load time, runtime organization and optimization
  - Any solution requires care in widespread deployment
Longhorn on the Horizon

- MS Longhorn (OS after XP)
  - Details given in Oct 2003 (PDC conference)
  - Large components written in managed code
    - WinFS – transactional file system
    - Avalon – managed UI + shell
    - Web Services
  - Managed APIs
- Longhorn emphasis…
  - Increases availability of interesting managed apps
  - Increases potential impact of performance solutions
Managed Code Challenges

- New overheads
  - I/O, Memory, CPU beyond SPECJVM issues
- Complex mental model
  - Biggest performance improvements involve human intervention
  - Managed code abstraction creates new developer challenges
I/O Overhead

- Substantial overhead at startup and ongoing
  - Code, metadata, static data all important
    - Static nature enhances optimization opportunities
- Disk and OS interaction cannot be ignored
  - HeadTrax warm start times highly variable
  - How useful is I/O data without a disk model?
  - OS / PL communities should get together on this
    - Who is considering placement on the disk?
- Should startup be a 1\textsuperscript{st} class research focus?
  - Why isn’t it now?
Memory Overhead

- Memory footprint has broad implications
  - GC is only one aspect
  - Who is looking at / solving other problems?
- What’s the memory cost of runtime ops?
  - How much space does JIT compiler, metadata, GC tables, etc. take up?
  - What’s overall performance impact of footprint on client?
- How to balance small program units versus memory fragmentation?
  - Current pressure to merge units
- Tools needed to expose issues and optimize
CPU Overhead

- Significant sources of CPU overhead
  - GC – thankfully, lots of research here
    - CPU overhead not currently on critical path for client
  - Exceptions – not as exceptional as one might expect
  - Managed / unmanaged interface
  - Security model
  - Runtime checking
What a Developer has to Think About

- GC gotcha’s from Rico Mariani (April 2003)
  - Too many allocations
  - Too large allocations
  - Too many pointers (high connectivity)
    - Too many roots
  - Too many writes (esp. to older objects)
  - Too many almost long-lived objects
    - Reasoning about lifespans and promotions
  - Finalization

- What tool support does a dev need or have?
Thoughts about the Future…

- Performance space is getting trickier
  - Memory latency is bad, getting worse
    - Prediction, placement, compression only go so far
  - Chip design favors chip multiprocessors
    - Pentium 4 – 2 HW threads, Prescott 4? HW threads
    - Power 4 – 2 processor, Power 5 – 2 processors w/ 2 threads each
    - Intel “core hopping” to balance temperature hot spots!

- Design is and should be a research option
Where Could Managed Code Go?

- How suitable for defining large-grain abstractions?
  - CLR has assemblies, Java has MJ, what else?
- How suitable for defining OS?
  - Several Java attempts, any serious contenders?
  - Valuable exercise or waste of time?
- Existing support for concurrency
  - Threads just too hard to get right? Alternative?
- Better models for isolation and robustness?

How do we get there?
Summary

- TwC (reliability, security) an important focus
  - Systems can and will get better
- Performance challenges remain
  - Can always trade performance for other qualities
- Memory latency threatens Moore’s Law
  - I/O performance a major challenge, underinvestigated
- Increasing investment in managed code
  - Developer experience is still immature
  - Current research misses important challenges
Things to be aware of…

- Phoenix research compiler infrastructure
  - Intended to be the basis of commercial compiler + research vehicle
  - Infrastructure for analysis, optimization at multiple compilation stages

- Rotor (SSCLI) continues to be developed
  - Tracking Whidbey design changes
  - Increased awareness of performance requirements for research use
  - Second RFP funded
Additional Resources

- CLR Performance Info
  - Includes white papers, clrprofiler tool
- FxCop
  - [http://gotdotnet.com/team/fxcop/](http://gotdotnet.com/team/fxcop/)
- PPRC
  - [http://research.microsoft.com/pprc](http://research.microsoft.com/pprc)
- Phoenix
  - [http://research.microsoft.com/phoenix](http://research.microsoft.com/phoenix)
- Rotor
  - [http://sscli.net](http://sscli.net)
More things to be aware of…

- PPRC now has link to Windows Org.
  - Amitabh now Windows VP of Development

- PPRC Groups
  - Advanced Compiler Technology (ACT) – David Tarditi
  - Binary Technologies (BiT) – Hoi Vo
  - Runtime Analysis and Design (RAD) – Trishul Chilimbi
  - Reliability – G.S. Rana
  - Static Program Analysis (SPA) – Manuvir Das
  - Software Productivity Tools (SPT) – Sriram Rajamani
  - Testing, Measurement, and Verification (TMV) – Tom Ball

- Applications for interns, fulltime hires requested by Feb 15, 2004
Something to think about...

Chip Multiprocessors are real

Today:
IBM dual processor Power4
HP dual processor PA-8800

2004:
IBM / Sony “Cell” processor
(speculated to have 4-16 processors on a chip)

2010?
The potential for these chips is enormous!

Time is running out! Thank you...
SPEC2000: eon vs mcf

Instructions/sec = 1/(CPI*cycle time)

- eon = 4.8x speedup (57% /yr)
- mcf = 1.9x speedup (20% /yr)

Data gathered and reported by Trishul Chilimbi
Data gathered and reported by Trishul Chilimbi
FxCop – a Short Introduction

- Managed app available on the Web
  - Checks conformance rules for .Net assemblies (think “lint” for CLR)
  - Easy to make it do a lot of work
- Presents performance challenges
  - Startup, memory footprint, CPU overhead
- Keeps GC busy as well!
  - Lots of strings
- Easy to get, I’m happy to demo + tools