x86 and x64 Performance Considerations when using Microsoft Visual Studio 2008

Robin Maffeo
AMD Alliance Manager for Microsoft Developer Division
Session Prerequisites

- Working knowledge of Managed code and the CLR
- Familiarity with C++ intrinsics
Session Objectives and Agenda

• Overview of AMD’s new native quad core AMD Opteron™ processor (“Barcelona”)

• Discuss implications and best practices for managed applications, including the Garbage Collector, code generation, threads, and locking.

• Explain Visual Studio 2008 native C++ code generation enhancements and intrinsics support for AMD “Barcelona” processors.
AMD Native Quad-core “Barcelona” Processor
AMD “Barcelona” improvements

- Native Quad-core die with shared L3 cache
- Micro-architectural enhancements for improved Instructions Per Cycle (IPC)
  - 32 byte instruction fetch
  - Enhanced branch predictor
  - Sideband Stack Optimizer
  - Load reordering
  - Larger TLBs
- Independent Dynamic Core Technology
- Dual Dynamic Power Management™ (aka split plane)
Improvements (continued)

• More efficient memory controllers
  • Deeper buffers for higher bandwidth
  • DRAM prefetcher
  • Independent 64-bit controllers
• Wide Floating-Point Accelerator (SSE128)
  • Single-cycle scalar operations
• SSE4a and Advanced Bit Manipulation instructions
• AMD-V™ enhancements
  • Faster world switch times
  • Rapid Virtualization indexing (aka Nested Page Tables)
• P-State Invariant RDTSC
Cache Enhancements

- L3 cache is up to 2MB in size; larger next year
- L3 is dynamically shared between cores
  - Intelligent LRU algorithm
- Each core maintains independent L1 and L2 caches
- As with prior AMD processors, caches higher than L1 are exclusive (victim) caches
MANAGED CODE
AMD and Microsoft

• Microsoft has worked closely with AMD on the CLR since Everett (CLR v1.1), with AMD providing hardware and engineering support

• AMD has developers dedicated to Microsoft compilers and runtimes
AMD and the CLR

• The x86 64-bit CLR started its life on AMD64 simulators, then actual Opteron hardware

• Performance and stress testing was all done on AMD hardware
  • Performance tuning
  • Correctness issues
  • NX Bit compatibility (Windows Data Execution Protection)
CLR Code Generation

- CLR contains hand crafted 64-bit assembly
  - JIT helpers (allocations, write barriers, etc)
- x64 code generation (JIT64/NGEN)
- Both generated code and Helpers built and measured on AMD hardware
- CLR code generation improvements always under consideration to enable improved managed application performance on AMD processors
Garbage Collection

• GC can be a performance critical aspect of managed applications

• GC Overview
  • CLR GC is a stop-the-world, generational, compacting, mark/sweep collector
  • Assumptions for a generational collector:
    • Young objects typically die young
    • Older objects typically die old
    • Collecting a portion of the heap is advantageous
CLR GC

- Three generations: 0, 1, and 2
  - Plus a separate large object heap
  - Temporally related objects have locality
- Collection occurs once Gen0 size has exceeded its budget
- Dead objects are marked
  - Objects without references “swept”, leaving holes
- First allocation after a GC will occur in an empty Gen0
CLR GC

- Compacting decision is made
  - Based on fragmentation
  - Fill holes, patch references
- Remaining objects in Gen0 promoted to Gen1
- Initial Gen0 size (budget) is based on processor cache size
- Generational budgets tuned dynamically at run time
CLR GC

- Older objects are promoted to Gen2
  - Gen2 size will increase when promotion occurs
  - Cost to collect relatively high
  - But collection frequency much less than Gen0 or Gen1
- Old objects in Gen2 could point to new objects
  - GC has to track objects in Gen2 via write barrier
  - Adds some overhead
GC Flavors

• Three GC flavors
  • Workstation with concurrent GC enabled
  • Workstation with concurrent GC disabled
  • Server
GC Flavors (Workstation)

- **Workstation GC**
  - Tuned for lower latency, at expense of throughput and slight working set increase

- **Concurrent GC**
  - Best for interactive UI
  - Reduces Gen2 pause times during collection
  - Small working set increase
  - Throughput slightly lower
GC Flavors (Server)

- Server GC
  - Tuned for throughput, at the expense of latency
  - No concurrent option – user threads are suspended
  - Best for non-interactive applications requiring allocator scaling
  - Affinitized heap per CPU
  - Enhanced for NUMA architectures
  - Lock free allocator
GC Flavor Selection

- Default is Workstation with concurrent GC enabled

- Using application .config file:

```xml
<configuration>
  <runtime>
    <gcConcurrent enabled="false"/>
  </runtime>
</configuration>
```
GC Flavor Selection

- To select Server GC:
  
  ```xml
  <configuration>
    <runtime>
      <gcServer enabled="true"/>
    </runtime>
  </runtime>
  </configuration>
  ``

- Using hosting API:
  
  - `pwszBuildFlavor` parameter of `CorBindToRuntimeEx()`

- Asking for Server GC on a single core machine gives you Workstation GC (concurrent disabled)
Object Allocation Best Practices

- Be prudent in your allocations
- Avoid medium lifetime objects
  - Kill them off early...
  - ...Or let them live long (and actually use them)
- Eagerly set objects to null to make unreachable
- Watch out for too many finalizable objects
  - May contribute to medium lifetime problems
Performance Counters

• .NET Performance counters are your friend
• Notable counters:
  • Gen0, 1, and 2 sizes
  • Gen0, 1, and 2 collection counts
  • % Time in GC
  • Allocated bytes/sec
  • Pinned objects
  • Promoted memory
  • Locks and Threads (contention)
Other considerations

• Be aware of allocations beneath you from Framework or 3rd party
  • Don’t assume APIs are inexpensive

• Also be aware of other side effects

• Example: XmlSerializer in v1.1
  • Invoked C# compiler underneath you at runtime!
  • Solution: Pre-generate serialization assembly with SGEN in v2.0
CLR Profiler

- Analyzes allocation behavior
- Helps diagnose object lifetime issues
  - Allocation and call graphs
  - Allocated types
  - GC survivors

Demo
CLR Threads

- Currently 1:1 mapping between logical CLR threads and physical OS threads
- Create new threads with System.Threading.Thread class
- Suspension, sleep as with Win32 threads
- Can wait on events created with AutoResetEvent, etc.
- Can wait for thread completion using Thread.Join()
- Synchronization using C# lock keyword or Monitor.Enter and Monitor.Exit
CLR ThreadPool

- Alternative to explicit thread creation
- ThreadPool manages thread creation/destruction and execution of queued items based on system utilization

API:
- QueueUserWorkItem(WaitCallback, state)

- Improved throughput in Visual Studio 2008
Lock Optimizations

• Multiple threads eventually require locks for synchronization
• Old CLR versions (v1) used a sync block to handle Monitor.Enter
  • Expensive...
• Newer CLR versions use thinner, lighter locks
  • Object header stores lock state
  • Very fast with no contention
  • Under moderate contention, promoted to sync block
• Old adage still applies: enter locks late, leave early
• Don’t lock gratuitously and only lock when necessary
• Think of ways to exploit data parallelism
VISUAL C++ 2008
Visual Studio 2008 “Barcelona” Improvements

- Improved code quality
  - Better utilization of branch predictor under certain conditions
  - Support for single-cycle SSE128
  - Improved performance for floating point conversions
    - int to float
    - int to double
    - float to double
  - Improved function inlining heuristics
Compiler Flag Recommendations

- For x64 code running on “Barcelona”:
  - Use /favor:blend (the default in Visual Studio 2008)
  - Enables optimal code generation for SSE128 support
  - When targeting older processors, consider /favor:AMD64

- Use /GL
  - Whole program optimization
  - Can inline across modules, reduce loads and stores

- Use /MP
  - Speeds compilation by compiling files in parallel on multi-core machines
Compiler Flag Suggestions

- /fp:fast
  - Depending on precision needs
- /O1
  - Consider compiling for size over speed depending on scenario
- /GS-
  - Improves performance slightly
  - Carefully consider security implications!
- /arch:SSE2 when targeting x86
  - Generates SSE2 instructions
Linker Flag Suggestions

- /LTCG (implied by /GL)
  - Linker can optimizes entire program

- /OPT:icf,ref
  - Ref tells linker to discard unreferenced modules
  - Explicit icf removes redundant (duplicate) functions

- For all flag changes – measure, measure, measure!
  - Consider using AMD’s CodeAnalyst™ Profiler
  - http://developer.amd.com/cawin.jsp
“Barcelona” SSE4a instructions

- **INSERTQ / EXTRQ**
  - Insert / extract bits from XMM registers

- **MOVNTSS / MOVNTSD**
  - Scalar non-temporal stores from XMM registers
  - Useful for data that won’t be referenced soon
  - Streaming store no longer needs to pack data into SSE register
“Barcelona” Advanced Bit Manipulation Instructions

- **LZCNT**
  - Counts the leading zeros in a register or memory location

- **POPCNT**
  - Counts the 1 (set) bits in a register or memory location
  - Useful for counting bits in a mask
Visual Studio 2008 Intrinsics for “Barcelona”

_mm_extract_si64, _mm_extracti_si64
  • Extract bits

_mm_insert_si64, _mm_inserti_si64
  • Insert bits

_mm_stream_sd
  • Writes 64bit data to memory location without caching

_mm_stream_ss
  • Writes 32bit data to memory location without caching
Visual Studio 2008 Intrinsics for “Barcelona”

__lzcnt16, __lzcnt, __lzcnt64
  • Count leading zeros

__popcnt16, __popcnt, __popcnt64
  • Counts number of one bits

• Use __cpuid intrinsic to check for SSE4a, POPCNT, and LZCNT support
In Summary...

• Numerous improvements in AMD’s new “Barcelona” processor family for new and existing code
• Managed code runs great on current and future AMD processors
• Visual Studio 2008 C++ compiler enhancements for “Barcelona”
• Use scalar SSE128 where possible
• Come visit AMD in the exhibition floor, stand A1
Resources

• AMD Developer Central
  http://developer.amd.com

• AMD CodeAnalyst™ Performance Analyzer
  http://developer.amd.com/cawin.jsp

• Microsoft CLR Profiler
Resources

- Technical Communities, Webcasts, Blogs, Chats & User Groups
  [Link]
- Microsoft Learning and Certification
  [Link]
- Microsoft Developer Network (MSDN) & TechNet
  [Link] [Link]
- Trial Software and Virtual Labs
  [Link]
- New, as a pilot for 2007, the Breakout sessions will be available post event, in the TechEd Video Library, via the My Event page of the website
Complete your evaluation on the My Event pages of the website at the CommNet or the Feedback Terminals to win!

All attendees who submit a session feedback form within 12 hours after the session ends will have the chance to win the very latest HTC 'Touch' smartphone complete with Windows Mobile® 6 Professional.
Trademark Attribution

AMD, the AMD Arrow logo, AMD Opteron, AMD-V, Dual Dynamic Power Management and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Windows is a registered trademark of Microsoft Corporation. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.