



Multi-Core Coding for Games & Phenom™ Software Optimization

Justin Boggs
Sr. Developer Relations Engineer
Sunnyvale, CA, USA

July 2007

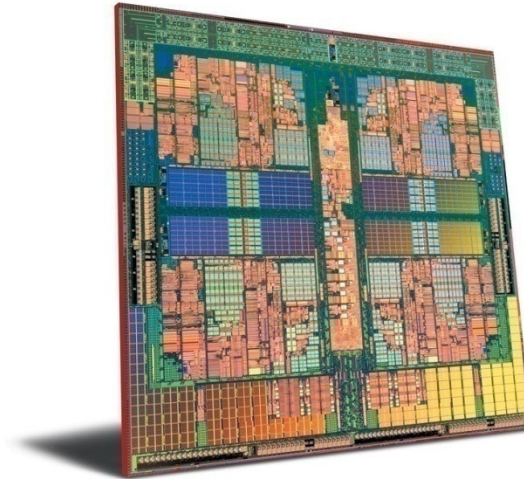
Agenda

- AMD Phenom™ Processor
- CPU Software Tools
- True Multi-core Microarchitecture
- Multi-threading Optimization
- Optimizing for 128-bit SSE
- New Instructions
- Next Steps



AMD Phenom™ Processor & Platform

AMD Phenom™ Processor



Phenom Performance and Feature Enhancements

- Up to four cores per processor
- Dedicated L2 cache, w/ shared L3 cache on many models
- HyperTransport™ 3.0 Technology
- 128-bit SSE Floating Point Unit (per CPU core)
- Support for DDR2-1066 memory
- Supports SSE4a multimedia instructions
- Split Power Plane

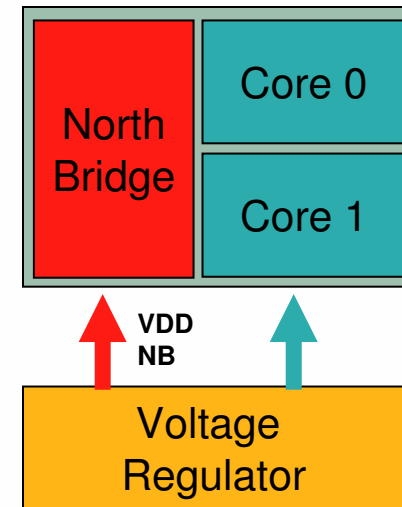
AMD Phenom™ Split Power Planes

Two power planes will be used for AMD Phenom processors with Socket AM2+

- One to power the processor core(s)
- One to power the internal northbridge

Improves power efficiency and increases memory performance

- While the processor core voltage changes up and down with p-state changes, the northbridge voltage and speed remain constant, sustaining the north-bridge's performance.



HyperTransport™ 3.0

The HyperTransport™ 3.0 bus can provide 2x or greater processor-to-I/O bandwidth vs. HyperTransport 1.0

The greater bandwidth supports PCIe Gen 2 interfaces and more

- Important for supporting Crossfire and SLI on PCIe Gen 2
- Important for future integrated graphics chipset performance
- Important for multiprocessor system performance

HyperTransport 3.0 is fully compatible with HyperTransport 1.0



CPU Software Tools

AMD CodeAnalyst™ profiler

- Profile your code to find hot spots
 - Timer-based sampling
 - Event-based sampling
 - Thread profiler
- Very easy to get started with AMD CodeAnalyst
 - Learn a lot about your application quickly
- Also see system code, drivers, other applications
- 32-bit and 64-bit, Microsoft® Windows ® and Linux®
- Download it from <http://developer.amd.com>

APL Overview

The AMD Performance Library (APL) is a collection of software routines designed to help developers write high performance multimedia software.

- **Simplifies** application development, debugging, and optimization
- Provides **future proof** interfaces to the processor; as processors evolve, so does APL, but the API stays the same
- Designed to work on all AMD **compatible** processors
- Currently focuses on image and signal processing functions
- Give us your **feedback** on what functions you want for **game development**
- Version 1.1 soon will add H.264 & JPEG functions and “Barcelona” optimizations

<http://developer.amd.com/apl.jsp>

Visual Studio 2005 on AMD64 - 1

- 32-bit code (x86)
 - Default: Uses x87 for FP
 - Recommendation: Try /arch:SSE2 when targeting Athlon 64 and newer processors, could improve performance
 - Recommendation: Use /LARGEADDRESSAWARE to take advantage of more VA space on an x64 OS, just beware of pointer assumptions
- 64-bit code (x64)
 - Uses SSE/SSE2, no more x87
 - Default: Optimized for “blend” target (/favor:blend) of “K8” (FP64) and P4 (FP128)

Visual Studio 2005 on AMD64 - 2

- New performance & code generation features
 - First Microsoft Compiler to Support AMD64/x64
 - SPEC CPU 2000 Performance Tuning
 - Added OpenMP threading support
- Compiler flag recommendations
 - /O2, /GL, /GS- (if safe), /fp:fast (if safe)
 - Try /O1 for min. code size, may be faster than /O2
 - Linker Optimizations: /ltcg, /OPT:REF
 - AMD Compiler Usage Guidelines Technical Document:
http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/32035.pdf

Visual Studio 2008 "Orcas" on AMD64

- 32-bit code (x86)
 - Default: Uses x87 for FP
 - Recommendation: Try /arch:SSE2 when targeting Athlon 64 and newer processors, could improve performance
 - Recommendation: Use /LARGEADDRESSAWARE to take advantage of more VA space on an x64 OS, just beware of pointer assumptions
- 64-bit code (x64)
 - Uses SSE/SSE2, no more x87
 - Default: **Optimized for "blend" target (/favor:blend) of AMD Phenom™ (FP128) and Core 2 (FP128)**
- New performance & code generation features
 - **Added Intrinsics for new "Barcelona" instructions (SSE4a)**
 - **SPEC CPU 2006 Performance Tuning**
 - **Additional FP128 Tuning**
- Compiler flag recommendations
 - /O2, /GL, /GS- (if safe), /fp:fast (if safe)
 - Try /O1 for min. code size, may be faster than /O2
 - Linker Optimizations: /ltcg, /OPT:REF
 - AMD Compiler Usage Guidelines Technical Document:
http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/32035.pdf

Visual Studio 2008 "Orcas" Beta

- Microsoft Pre-release Software Visual Studio Code Name "Orcas" - March Community Technology Preview (CTP)
 - <http://www.microsoft.com/downloads/details.aspx?familyid=CF76FCBA-07AF-47AC-8822-4AD346210670&displaylang=en>
- Microsoft Pre-release Software Visual Studio Code Name "Orcas" - March Community Technology Preview (CTP) (Team Foundation Server)
 - <http://www.microsoft.com/downloads/details.aspx?familyid=C17C9FB8-2A4A-426D-B08B-6AE614D16A0D&displaylang=en>

More information at AMD Dev Central

Real-world solutions with practical guidance.

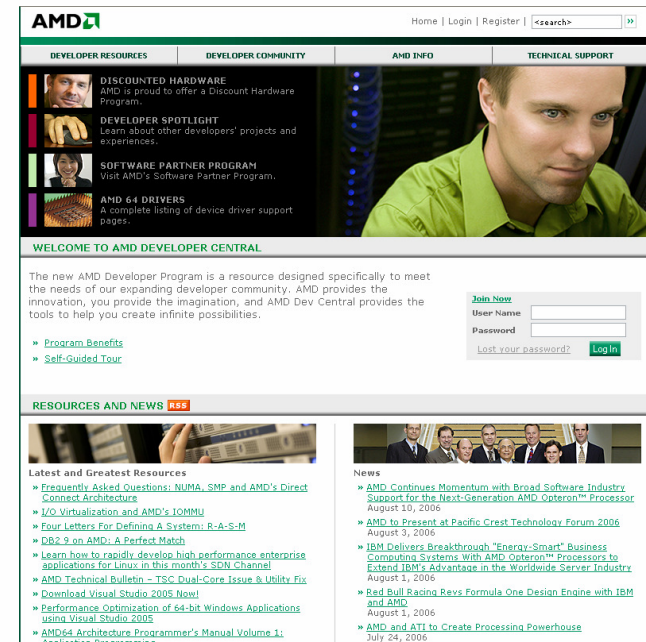
- Detailed technical articles
- Documentation, tutorials, and guides
- Case studies

Free tools and resources to get your job done better, faster.

- CodeAnalyst Performance Analyzer
- AMD Core Math Library
- Multi-core Technology Zone

Expertise from the leader in x86 and 64-bit computing.

- Secure access to latest AMD systems
- Sneak peek at emerging technologies
- Inside look at AMD's vision



Join today, it's free!

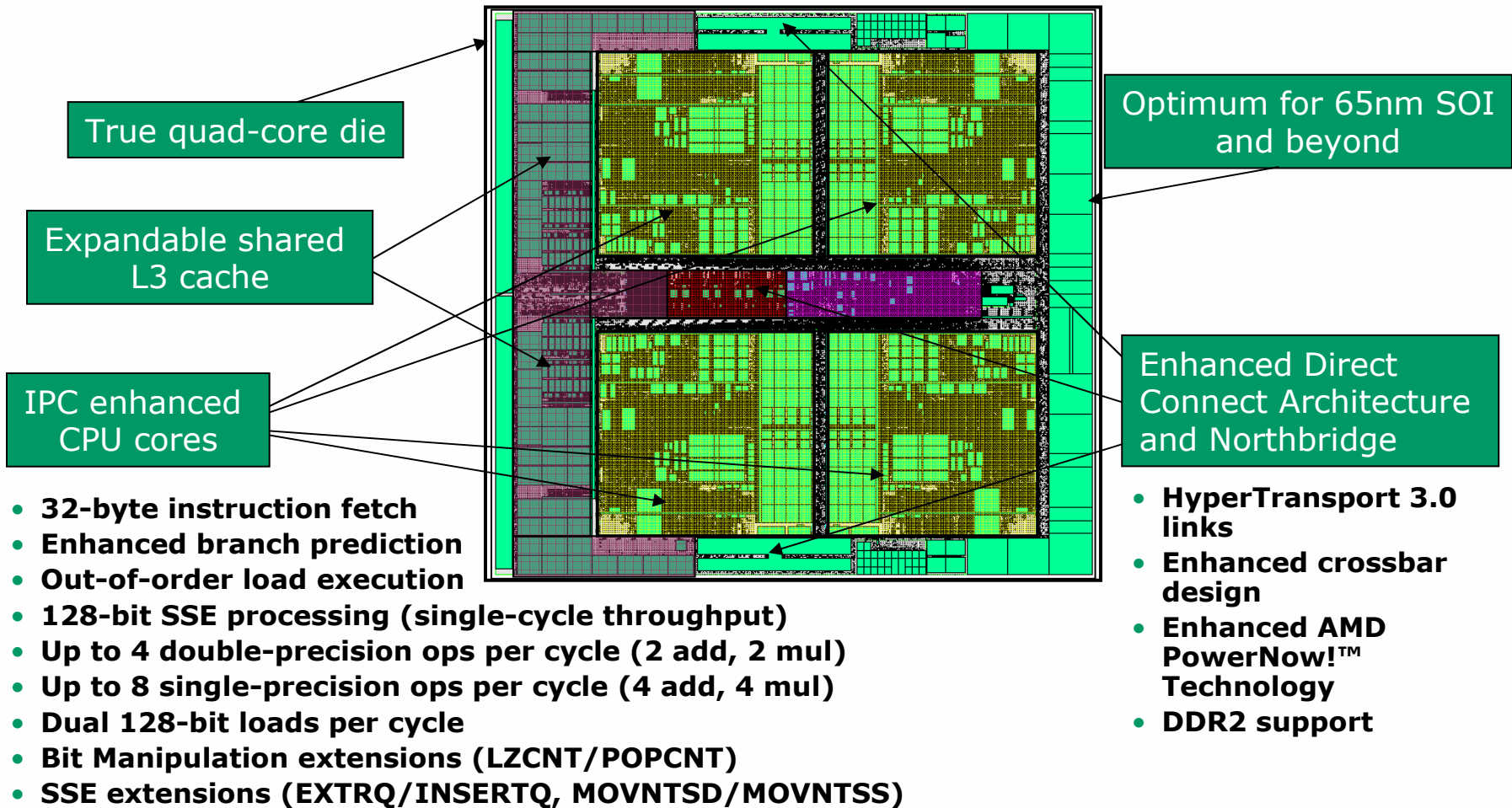
developer.amd.com



True Multi-core Microarchitecture

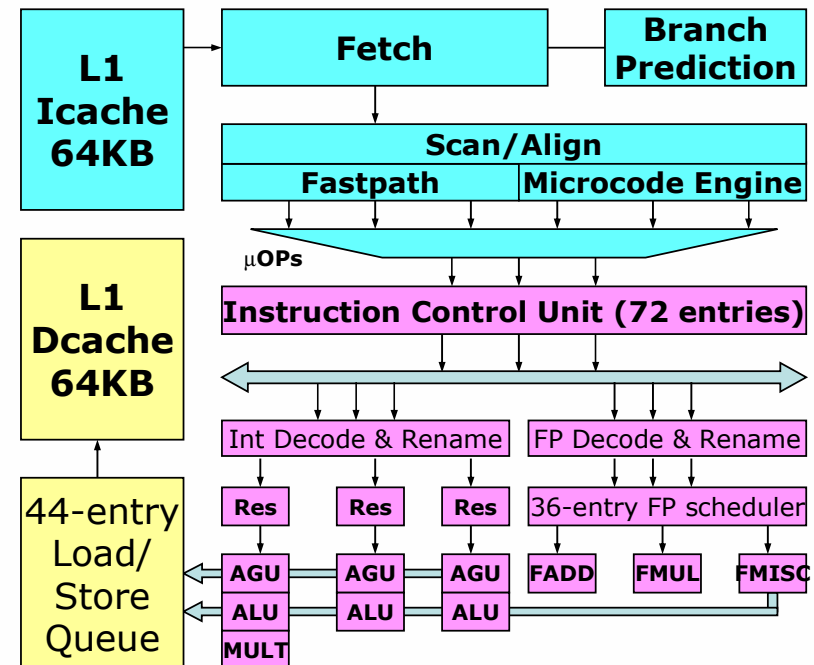
Quad-Core Phenom™ Processor

Continued Performance Leadership



CPU Core IPC Enhancements

- Advanced branch prediction
- 32B instruction fetch
- Sideband Stack Optimizer
- Out-of-order load execution
- TLB Optimizations
- Data-dependent divide latency
- More Fastpath instructions
 - CALL and RET-Imm instructions
 - Data movement between FP & INT
- Bit Manipulation extensions
 - LZCNT/POPCNT
- 128-bit SSE processing
- SSE extensions (SSE4a)
 - EXTRQ/INSERTQ,
 - MOVNTSD/MOVNTSS



Balanced, Highly Efficient Cache

Dedicated L1

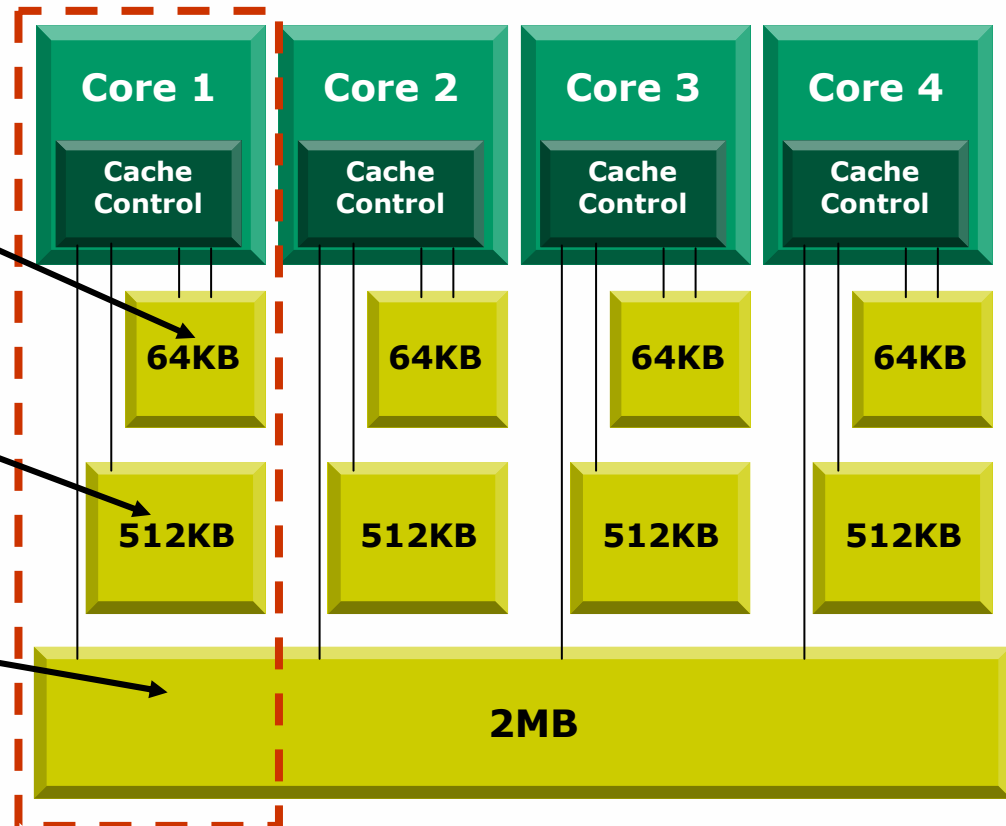
- Locality keeps most critical data in the L1 cache
- Lowest latency
- 2 loads per cycle

Dedicated L2

- Sized to accommodate the majority of working sets today
- Dedicated to eliminate conflicts common in shared caches

Shared L3 – NEW

- Victim-cache architecture maximizes efficiency of cache hierarchy
- Fills from L2 leave likely shared lines in the L3
- Sharing-aware replacement policy



Optimizing for cache

- Efficiently using the cache is important for performance
 - Each core has its own 512k L2 cache
 - L2 cache is “unified”, it stores both instruction and data
- Compiler code generation option
 - Evaluate “minimize code size” instead of “maximize speed”
- Data structures
 - Be careful to avoid wasteful padding in structs (check “sizeof”)
 - Sometimes a 2-byte offset value or array index can replace a 4/8-byte pointer
- Algorithms
 - Process data in blocks of several K-bytes, not megabytes
 - Read them using non-temporal software prefetch when appropriate*
 - Avoid writing large data sets to memory, then reloading them!
 - Keep operating within cache whenever possible*
 - Write final data blocks out to memory using Streaming Store
 - Avoid disturbing data and code that are in the L2 cache*
- Analysis
 - Use AMD CodeAnalyst to measure various cache miss events
 - Easily locate performance trouble spots in your code

RDTSC Instruction

Read Time Stamp Counter (RDSTC) changes in the AMD Phenom™ processor

- The instruction is now an invariant instruction, independent of what processor core it is executed on
- Counter will not be different depending on which core it is executed on, as could occur with "K8"
- RDTSC latency increases from ~12 cycles to ~60 cycles

AMD Recommendation:

- Use the OS timing functions whenever possible, such as the these APIs (for Microsoft Windows):

QueryPerformanceCounter ()

QueryPerformanceFrequency ()

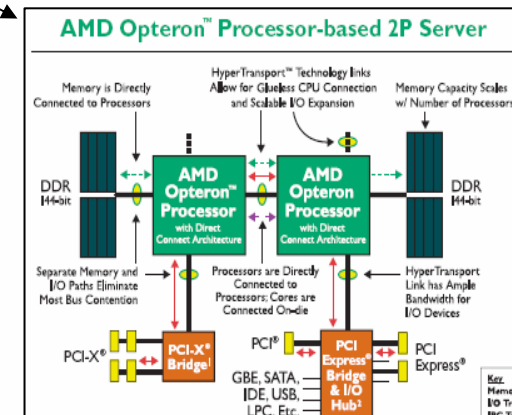
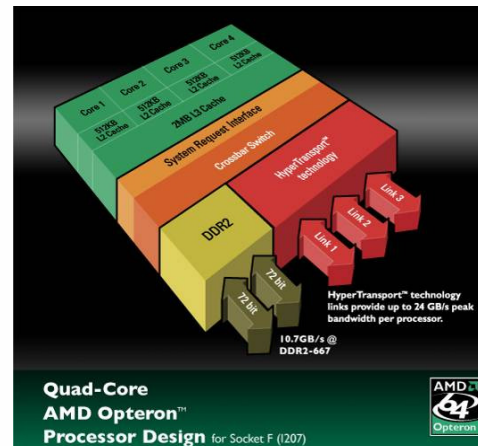
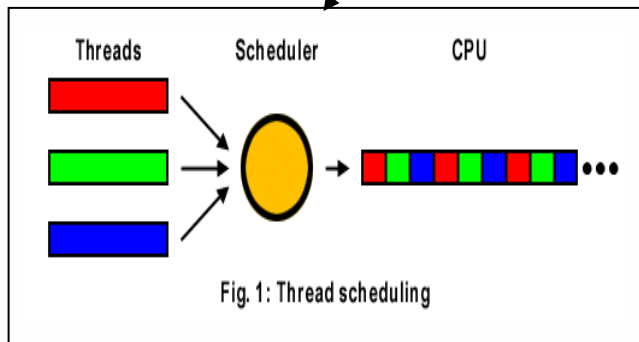
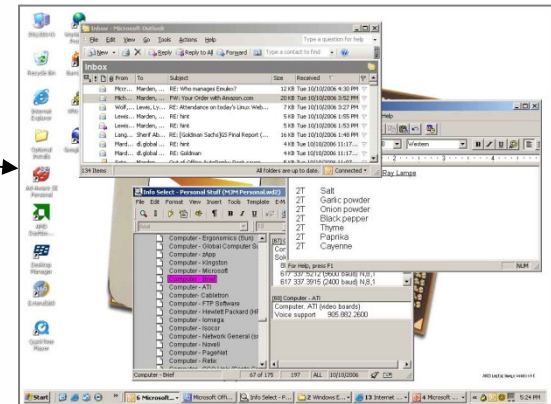
- Let the OS determine which counter it should use under the hood



Multi-threading Optimization

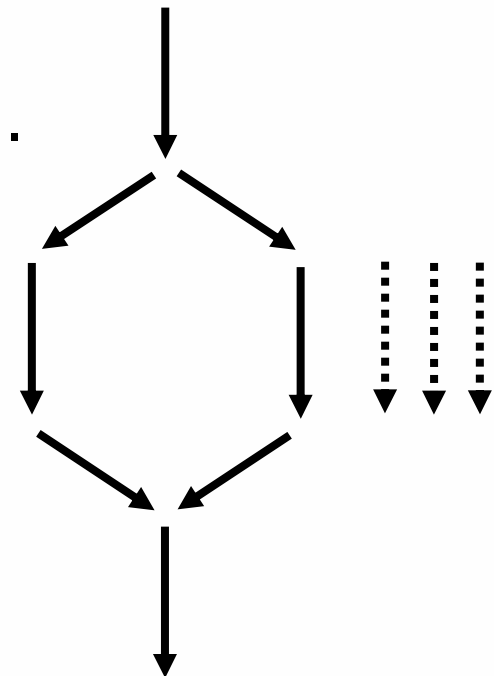
The World now thinks of computer power in Multiples

- Single Tasking
- Multiple Tasks
- Multiple Processors
- Multiple Cores
- Multiple Threads



Multi-Threading Optimization

- Dual-core or greater will be everywhere
 - Utilize those cores!
- Functional threading will run out of steam...
 - Solution: Data Parallel Threads
 - Use OpenMP, thread pools, or your own threads using OS API
 - Generally at least one thread per core
- Keep locks to a minimum
 - Take locks as late as possible, release as early as possible
- Beware of false sharing
 - Keep variables updated by different threads on separate cache lines



OpenMP: Example

```
#pragma omp parallel for
for (int i = 0; i < size; i++)
{
    x[i] = (y[i] + y[i+1]) / 2;
}
```

- Computes average of two values in one array, storing into another array
- Join at end of region is implicit – all threads complete
- OpenMP library handles scheduling over available cores in a static fashion (roughly iterations / cores)

The Cache and Multi-Threading

- AMD Phenom™ shared L3 cache can help threads share data
 - If you are using a producer/consumer thread model...
 - Keep producer and consumer threads on the same chip*
 - Set affinity to the same socket (currently = same NUMA node)*
 - Unlike quad-core processors from other vendors, AMD's cache is shared across all cores
- Be careful to avoid "thrashing" the L2 cache
 - Don't have multiple threads modifying the same variables
 - And don't have one thread modifying it, while others read it*
 - Don't split a cache line between two threads
 - Allocate thread-specific memory with 64-byte granularity*
 - AMD CodeAnalyst™ can easily find these type of trouble spots
 - Excessive cache miss events*
- Shared cache considerations
 - Heavy cache usage by a thread on one core could negatively impact other threads

Detecting the # of Cores Can Be Tricky

- Use the great “CoreDetection” sample code in the latest DirectX SDK
- Anticipate the adoption of virtual environments
 - Physical computer may be partitioned: you don’t get the whole thing
 - Trust the OS to tell you about available resources
 - *Really try to avoid using low-level instructions on the hardware*
- For a recent OS (Vista and later) use the new function `GetLogicalProcessorInformation()`
 - Gives you more complete info about NUMA, cache, etc.
 - Lets you distinguish true cores from SMT threads (pseudo-core)

Multi-Threading Resources

- AMD Developer Central
<http://developer.amd.com/>
- “Designing & Optimizing for N Cores” presentation
http://developer.amd.com/assets/AMD_Designing_for_N_cores.pdf
- MSDN
<http://msdn.microsoft.com/>
[http://msdn2.microsoft.com/en-us/library/172d2hhw\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/172d2hhw(VS.80).aspx)



Optimizing for AMD Phenom™ 128-bit SSE

128-bit SSE improves performance

- Doubles VECTOR SSE performance vs. previous CPUs
- Example : ADDPD (Add Packed Double)
 - AMD eighth generation processor e.g. current Athlon 64
 - Decodes into two ADD micro-ops*
 - Each micro-op takes a pass through the 64-bit adder*
 - AMD Phenom™ w/ 128-bit SSE
 - Decodes into a single ADD micro-op*
 - Takes one pass through the 128-bit adder*
- Example performance gains
 - Matrix multiply performance improves by ~85%
 - Other math intensive apps show 10%-50% gains
- **Vectorized SSE is very fast with 128-bit SSE**
 - Much faster than x87 & scalar SSE
 - Double-precision is 2x faster than x87
 - Single-precision is 4x faster than x87

Comprehensive Upgrades for 128-bit SSE

Current Generation versus Next Generation



Parameter	Current Processor	AMD Phenom™
SSE Exec Width	64	128 + SSE MOVs
Instruction Fetch Bandwidth	16 bytes/cycle	32 bytes/cycle + Unaligned Ld-Ops
Data Cache Bandwidth	2 x 64bit loads/cycle	2 x 128bit loads/cycle
L2/NB Bandwidth	64 bits/cycle	128 bits/cycle
FP Scheduler Depth	36 Dedicated x 64-bit ops	36 Dedicated x 128-bit ops

- Can perform SSE MOVs in the FP “store” pipe
 - Execute two generic SSE ops + SSE MOV each cycle (+ two 128-bit SSE loads)
- SSE Unaligned Load-Execute mode
 - Remove alignment requirements for SSE load-op instructions
 - Eliminate awkward pairs of separate load and compute instructions
 - *To improve instruction packing and decoding efficiency*

128-bit SSE Caveats

- Only *vectorized* SSE is accelerated
 - No improvement for scalar code

- 128-bit SSE can't separately address 64-bit halves of XMM reg
 - MOVLPD/MOVHPD internally need to merge before writing 128-bit register
 - Merge creates an extra dependency on output register:
 - Example: MOVLPD XMM0, [...]*
 - 128-bit SSE part will read XMM0 ; today's parts just overwrite XMM0L
 - Impacts these instructions:
 - movhpd, movhps***
 - movlpd, movlps***
 - movsd*** (*reg version only*)
 - cvtpi2ps, cvtsi2sd, cvtss2sd*
 - sqrtsd*
 - movhlps, movlhps*

- 128-bit SSE adds an additional register read pipe stage
 - Only impacts floating point pipeline
 - Adds a cycle to FP load latency

128-bit SSE Loads and Stores

- Data cache bandwidth
 - Two ports (banked)
 - Each port supports either a 128-bit load or a 64-bit store
 - Better than AMD eighth generation (current) processor, where each port supports either a 64-bit load or a 64-bit store*
 - **Software may want to use SSE for copy loops to take advantage of increased data cache load bandwidth**
- Loads are decoded as single 128-bit micro-op
 - Includes MOVUPD/MOVUPS/MOVDQU (misaligned loads)
 - No penalty for these if aligned*
- Stores are decoded into two 64-bit micro-ops
 - But data delivered from floating point to DC in a single 128-bit chunk
- 128-bit Store-To-Load Forwarding
 - Can forward from 128-bit stores to 128-bit loads
 - Not from MOVHPD/MOVLPD stores to 128-bit load

The best way to load an SSE register (when memory alignment is in doubt)

- Latency and required Data Cache ports

128-bit Ld	"K8" revE	"K8" revF/revG	Phenom™ w/ 128-bit SSE
MOVLDP+ MOVHPD	A8:3-cycle, 2-ports	A8:3-cycle, 2-ports	A8:5-cycle, 2-ports + input dep for merge
MOVU*	A8:5-cycle, 2-ports 2-line UCODE	A8:5-cycle, 2-ports 1-line UCODE	A16:3-cycle, 1-port A8:4-cycle, 2-ports

- Key: A16: aligned on 16-byte boundary A8: aligned on 8-byte boundary

→ On Phenom w/ 128-bit SSE, MOVU* is preferred over MOVLDP/MOVHPD pairs

- A16: MOVUs take only 1 DC port
- A8: MOVUs same latency/BW as MOVL+MOVH
 - Only one op required

Optimizing your own code for 128-bit SSE



- Vectorize your own numeric code for best performance
 - Up to 4x performance gain vs. scalar code
- “Blended” optimizations with no performance impact on FP64
 - Be aware of input dependency on MOVLPD/MOVHPD and MOVSD
 - Avoid choosing a recently-written register:*

```
MULPD   XMM7, ...,
MOVLPD  XMM7, [ ] ;; in 128-bit SSE, has dependency on MULPD.
```
 - Or perhaps replace MOVLPD-mem with MOVSD-mem*
 - Schedule MOVLPD/MOVHPD pairs as taking 2 extra cycles
- For best performance on 128-bit SSE:
 - Replace MOVLPD / MOVHPD pairs with MOVUPD or MOVDDUP
 - Replace MOVLPD-mem with MOVSD-mem (upper 64 bits are zeroed)
 - Replace MOVSD-reg with MOVAPD
 - Take advantage of misaligned load-op mode (special code path required)
- Use SHUFPS/SHUFPD instead of unpccklps/hps/lpd/hpd

Optimized libraries for 128-bit SSE

- Use vectorized SSE libraries!
 - Why reinvent the wheel?
- Use AMD Performance Library, APL
 - Optimized SSE code, vectorized and multi-threaded
 - Functions include array math and image processing
 - <http://developer.amd.com/apl.jsp>
- Other optimized libraries
 - **RAD Game Tools**: Bink, Miles, Granny, and Pixo use SSE extensively
 - And many more!



AMD Phenom™ New Instructions

New SSE4a Multimedia Instructions

- For all new instructions, make sure to test for support by using CPUID!
- MOVNTSD
 - Stores DP XMM register into 64-bit memory location, treating it as non-temporal to minimize cache pollution
 - Useful when data is unlikely to be used again soon
- MOVNTSS
 - Same as MOVNTSD, but for SP 32-bit value
- EXTRQ
 - Extracts bits (at a given offset/length) from lower 64 bits of XMM register and stores into the least significant bits of the destination XMM register
- INSERTQ
 - Inserts least significant bits from lower 64 bits of XMM register into lower 64 bits of destination XMM register (providing offset and length)

New SSE Misaligned Access Support

- Prior generation processors (i.e. K8) raise an exception if an SSE instruction tries to access a 128-bit memory location that is not 16-byte aligned
- Phenom support misaligned access without a fault, with a slight performance penalty
 - Data should still be aligned for best performance, when possible
- Compilers and asm code can avoid testing for alignment when it is not known up front

New General Purpose Instructions

- LZCNT
 - Leading Zeros Count
 - Counts the number of leading zeros in a register or memory location. Starts at the most significant bit and works downward until a 1 is seen.
- POPCNT
 - Bit Population Count
 - Counts the number of set bits in a register or memory location
 - Useful for counting bits in a mask



Next Steps

Next Steps

- Quad-core systems & processors are here in 2007... the time to multi-thread is now
 - Use Data Parallel Threading as your threading model for scalable performance for years to come
- Download and evaluate APL v1.0 today, v1.1 this summer
- Vectorize your math intensive code by using optimized libraries (ex: APL) or by writing optimized SSE code
- Test, develop, and optimize on AMD to ensure top performance and solid compatibility

Disclaimer & Attribution

DISCLAIMER

The information presented in this document is for information purposes only. AMD has used commercially reasonable efforts to confirm that the procedures and other information set forth in this document are accurate and reliable. Despite such efforts, the information contained herein may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including, but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN OR FOR THE PERFORMANCE OR OPERATION OF ANY PERSON, INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, DAMAGE TO OR DESTRUCTION OF PROPERTY, OR LOSS OF PROGRAMS OR OTHER DATA, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

©2007. Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD Athlon, AMD Sempron, AMD PowerNow!, AMD CodeAnalyst, AMD Phenom, CrossFire, and combinations thereof, are trademarks of Advanced Micro Devices, Inc. HyperTransport™ is a licensed trademark of the HyperTransport Technology Consortium. Microsoft and Windows are registered trademarks and Windows Vista is a trademark of Microsoft Corporation in the U.S. and/or other jurisdictions. Linux is a registered trademark of Linus Torvalds. Other names are for informational purposes only and may be trademarks of their respective owners.