TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Overview over the x86 Processor Architecture

Daniel Molka Daniel.Molka@tu-dresden.de

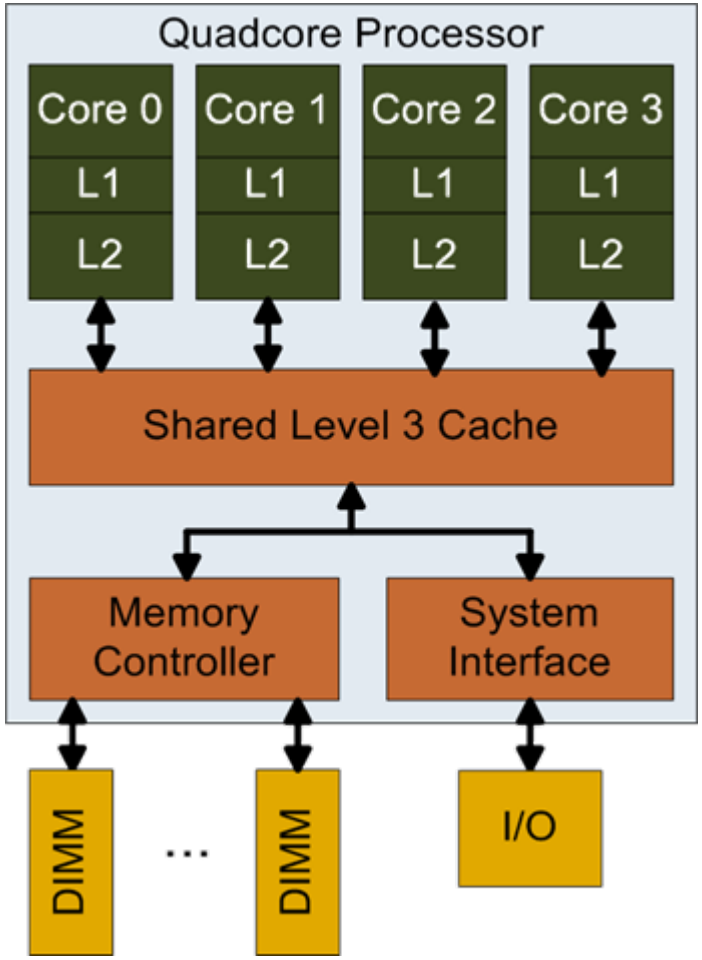
Ulf Markwardt ulf.markwardt@tu-dresden.de

ZIH
Center for Information Services & High Performance Computing

Outline

- Instruction set
  - Instruction format
  - ISA extensions
- Pipelining
- Out-of-order execution
- Memory access
  - Address translation
  - TLB
  - Measurements on current multicore processors, complex cache hierarchy
  - Remote memory access

Multicore Processors



- Multilevel Caches
  - L1/L2 per core
    - Accesses to other cores caches can have high overhead
  - Shared L3
    - Competitively shared
    - Interference due to replacements caused by other cores and bandwidth limitations of concurrent accesses
- Shared memory controller and system interface
  - Cores compete for bandwidth
  - One core can not always utilize available memory bandwidth

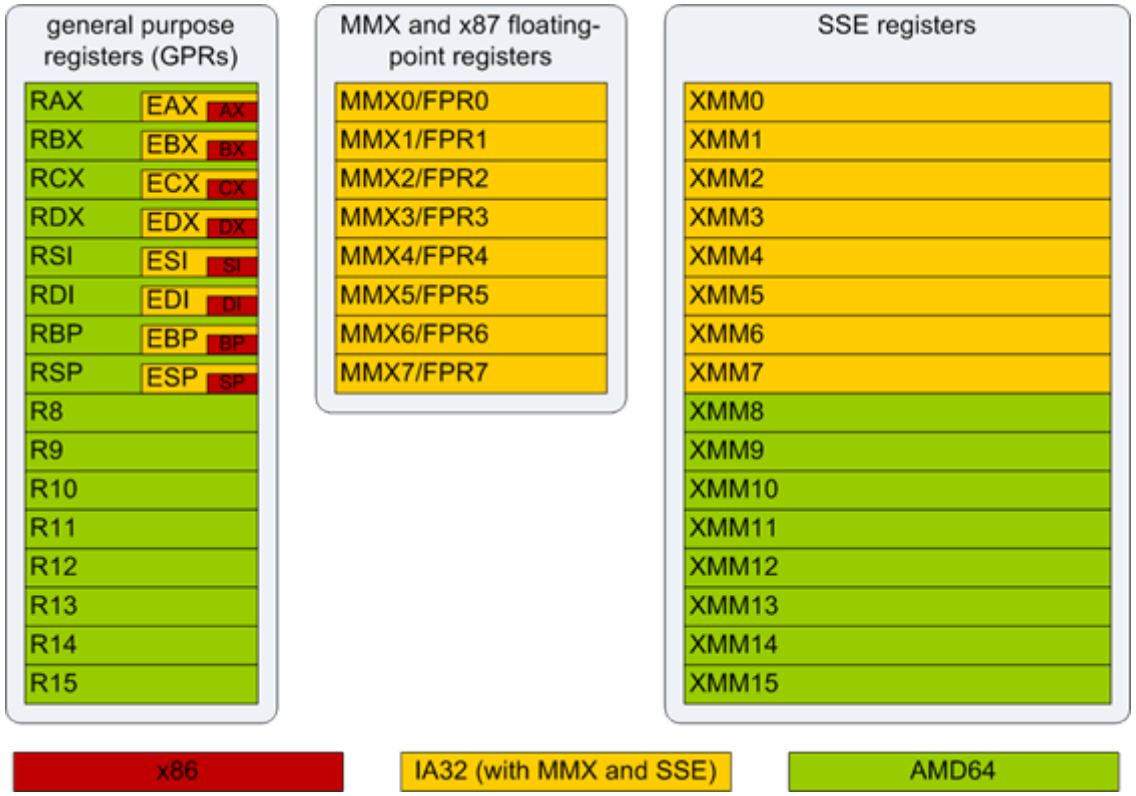
Instruction set

- x86 instructions
  - Integer instructions
    - originally 16 Bit, later extended to 32 (IA-32) and 64 Bit (AMD64)
- FPU
  - Floating point instructions
  - 32, 64 and 80 Bit
- SIMD extensions
  - Single instruction multiple data
    - Reduces code size if applicable
  - MMX: 64 Bit Register, 8x 8 Bit/ 4x 16 Bit/ 2x 32 Bit integer ops/cycle
  - SSE: 128 Bit Register
    - 4x 32 Bit/ 2x 64 Bit floating point ops/cycle
    - 16x 8 Bit/ 8x 16 Bit/4x 32 Bit/ 2x 64 Bit integer ops/cycle
  - Coming soon: AVX: 256 Bit Register

Instruction format

- CISC (Complex Instruction Set Computing)
  - Many instruction formats
  - Variable instruction length
  - Multiple addressing modes
- RISC (Reduced Instruction Set Computing) execution
  - Complex x86 instructions are decoded into RISC-like instructions, so called microops
  - Load/Store architecture
    - Special instructions for data movement between registers and memory
    - Execution units (ALU,FPU) only work on registers

Architectural Registers



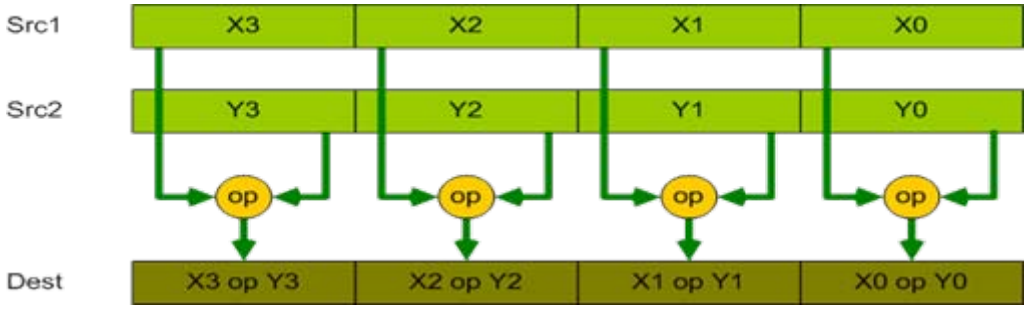
- Original x86 had very few registers
  - Frequent memory accesses requiered
- extensions added more and wider registers

Streaming SIMD Extensions on Intel and AMD processors

- Intel
  - SSE (*1999 Pentium III)
    - Single precision floating point instructions on XMM registers
    - Additional integer instructions on MMX registers
  - SSE2
    - Double precision floating point instructions on XMM registers
    - Integer instructions on XMM registers
  - SSE3: additional floating point instructions
  - SSSE3 (Supplemental SSE3): additional integer instructions
  - SSE4.1: scalar product, blending, min, max
  - SSE4.2: string operations, CRC32
- AMD
  - 3D-Now! (*1998 AMD K6-2)
  - SSE, SSE2, SSE3 compatible with Intel definition
  - SSE4A: insert and extract instructions, not compatible with Intel's SSE4

Using SIMD instructions

- Basic Operation:



- Data Types:

Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
Word		Word		Word		Word		Word		Word		Word		Word	
Doubleword				Doubleword				Doubleword				Doubleword			
Quadword								Quadword							
Double Quadword															

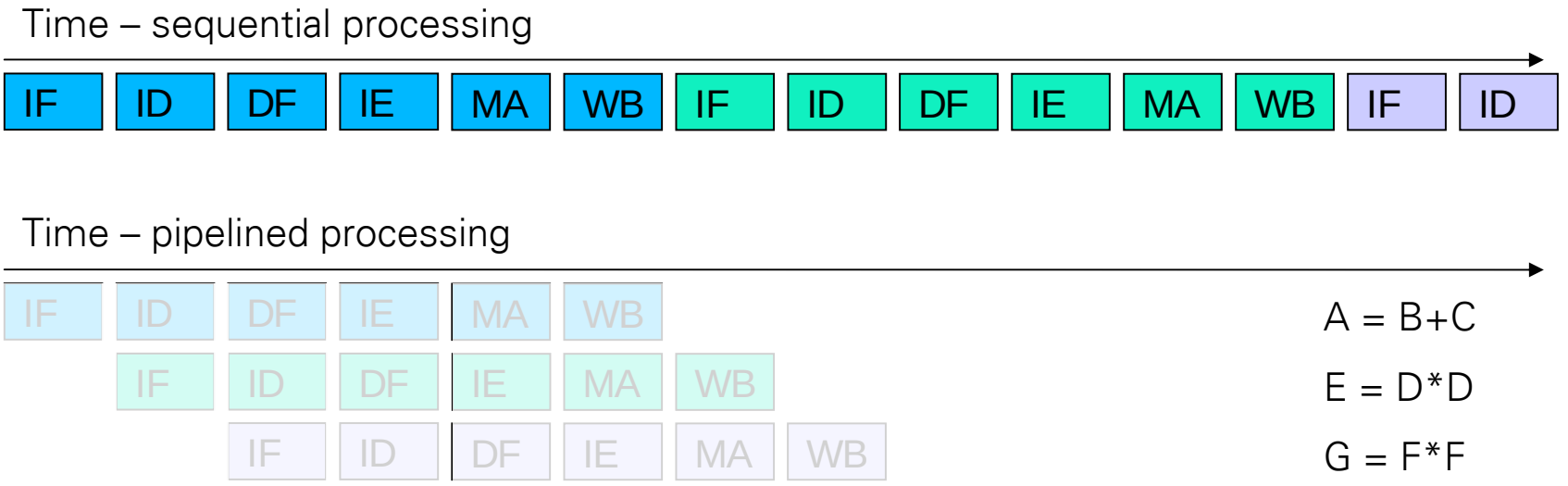
  - Aligned instructions are faster, but require double quadwords to be 128 Bit aligned
  - Standard function malloc() allocates unaligned memory
  - Compilers fall back to unaligned instructions if alignment is unknown

Using SIMD instructions

- Available via intrinsics, assembly or auto-generated by the compiler
- icc
  - -msse2 (default), -msse3, -mssse3, -msse4.1
    - -msse2, -msse3: generated code compatible with AMD processors
  - -x and -ax
    - Optimized code for Intel processors
    - -x requires specified extension, -ax includes fallback implementation according to -m/-x
      - E.g. -msse2 -axSSE4.2,SSE4.1,SSSE3,SSE3: runs on all processors with SSE2

        -xSSE4.2: only runs on Intel Core i7 and Nehalem based Xeons
- gcc
  - -msse, -msse2, -msse3, -mssse3
    - -msse, -msse2 enabled by default on 64 Bit systems

Basic pipelining



- Instruction fetch: retrieves the instruction from the cache
- Instruction decode: decodes the instruction and looks for operands (register or immediate values
- Execure: performs the instruction (ADD, SUB,…)
- Memory access: accesses the memory, and writes data or retrieves data from it
- Write back (retire): records the calculated value in a register

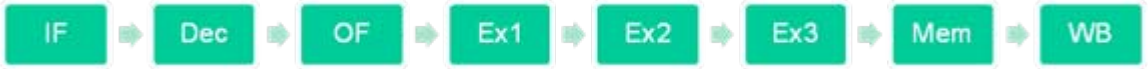
Pipelining

- Each pipeline-stage can work on a different instruction
- Superscalar pipelines can process multiple instructions in each stage
- Ideally n instruction finish each cycle in an n-way superscalar architecture

2-way superscalar pipeline

Longer Pipelines

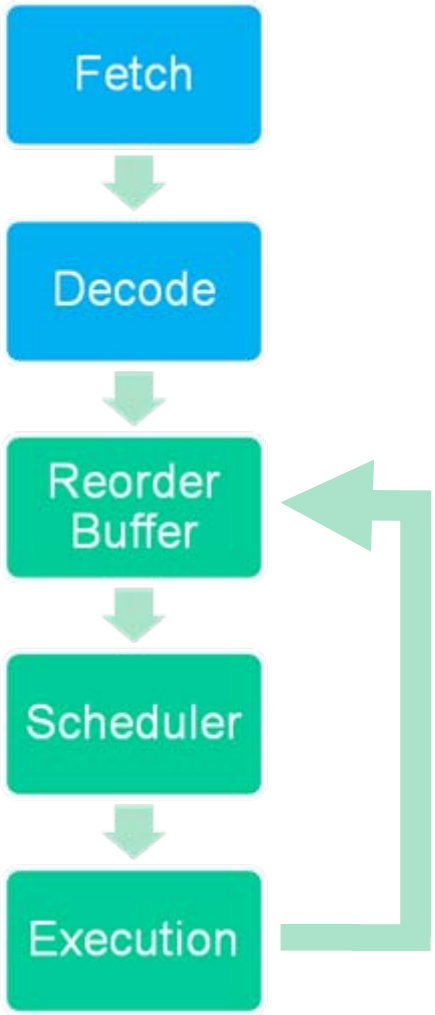
- Phases can be split further
  - Decode
    - Pre-Decode (e.g. determine instruction length if not constant [x86: 1 Byte – 18 Byte])
    - Decode: Analyse Opcode
  - Execution



    - E.g. Floating Point Addition
      - Align exponents
      - Add significants
      - Normalize result
    - E.g. Floating Point Multiplication
      - Add exponents
      - Multiply significants
      - Normalize result
- Hyper-pipeline / super-pipeline on modern micro-processors with appr. 20 stages

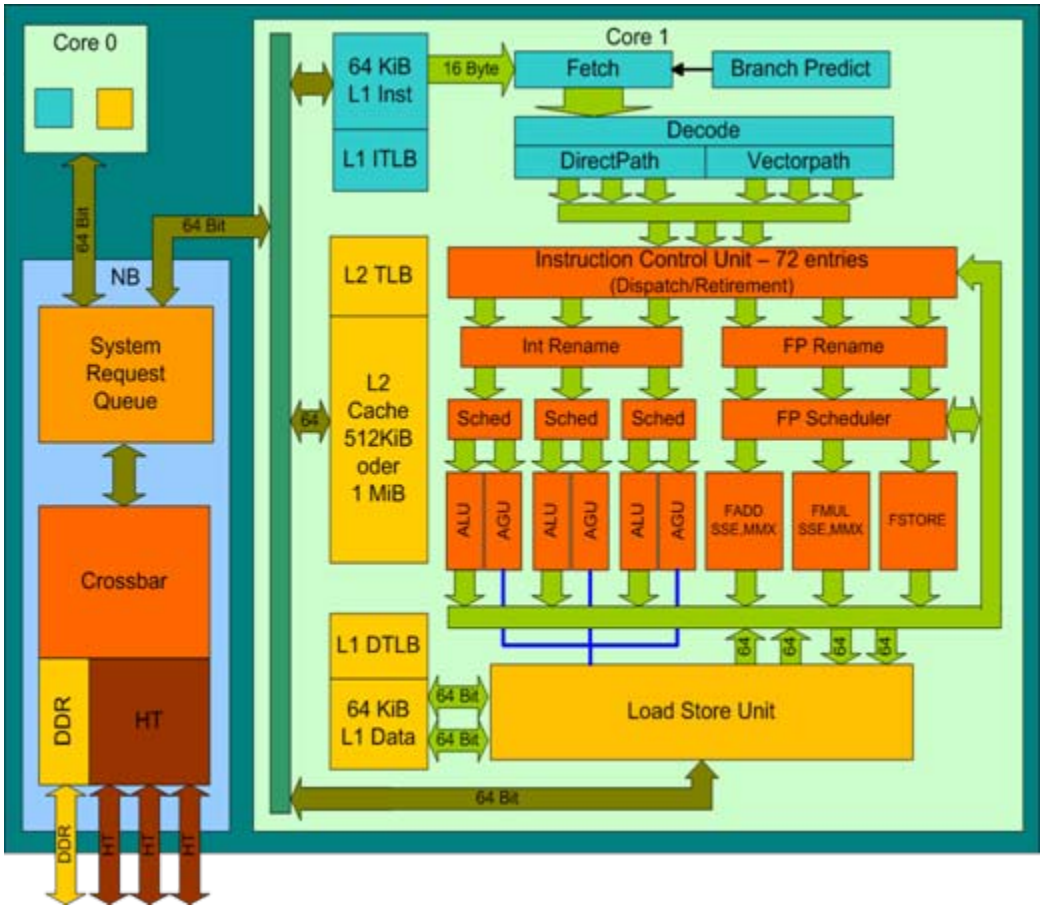
Out-of-order execution

- Reordering of instructions
  - Microops do not have to be executed in program order
    - Slow operations (e.g. division) do not stall execution
    - Scheduler can sent subsequent instructions to execution units if they do not have unsatisfied dependencies
  - In-order completion
    - Execution units work on internal registers
    - Reorder Buffer commits (write back/retirement) results to architectural registers in program order



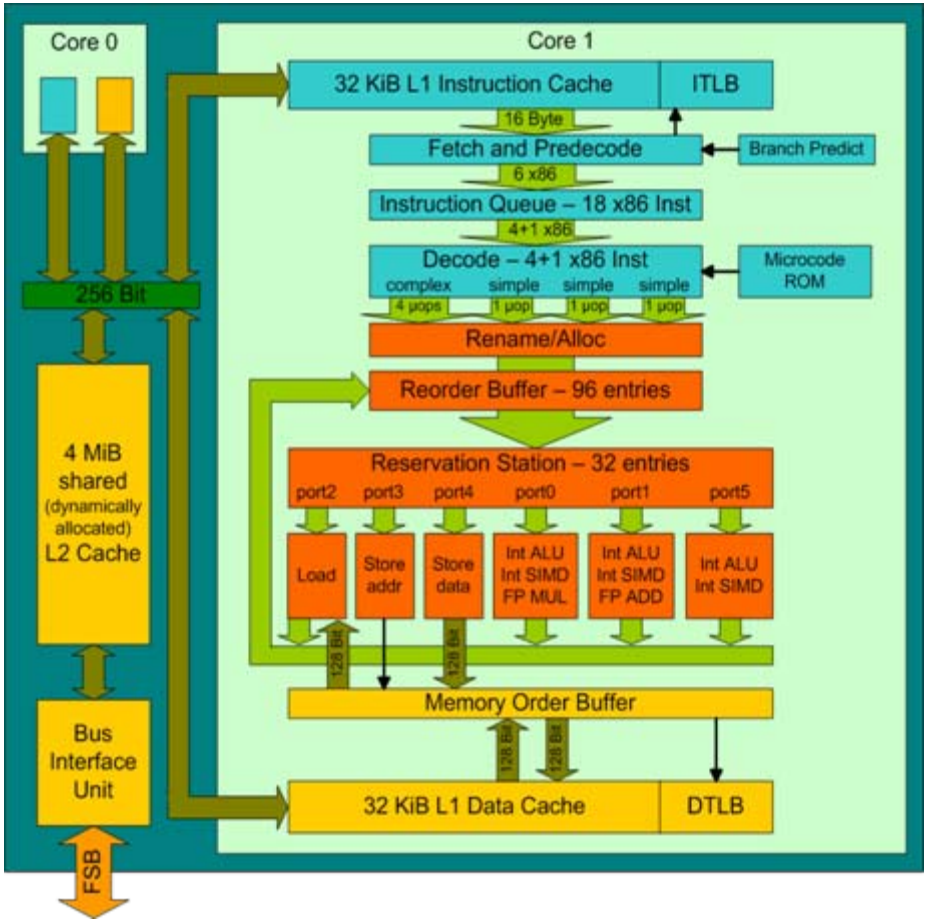
AMD dual-core Opteron (K8 microarchitecture)

- Decodes up to 3 instructions per cycle
- 128 Bit SSE instructions are spl into two 64 Bit ops
- 72 Makroops "in-flight"
- 3 Integer Units
- 2 (+store) FP pipelines
- two 64 Bit Load/Store operations per cycle
- L1/L2 per core
- System interface shared between both cores



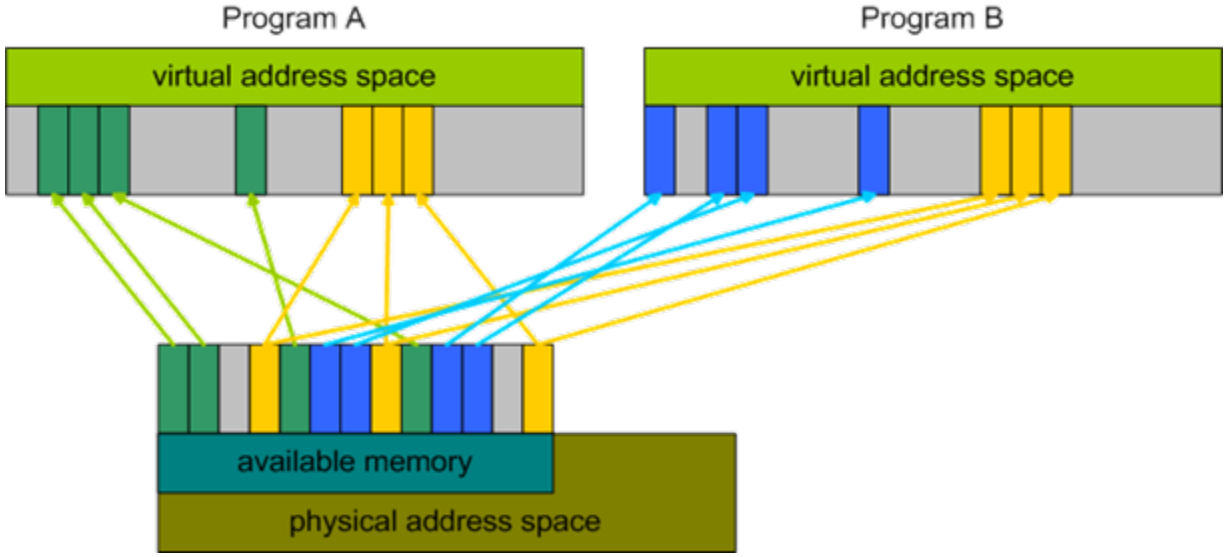
Intel dual-core Xeon (Core™ microarchitecture)

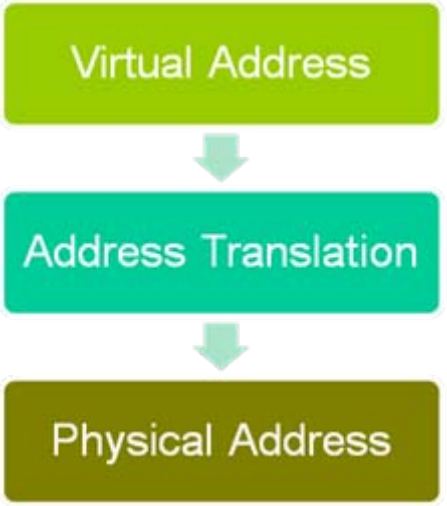
- Decodes up to 4 instructions per cycle (5 with Makroop-Fusion)
- 128 Bit wide SIMD units
- 96 Microops “in-flight”
- 3 Integer Units
- 2 FP pipelines
- One 128 Bit load and one 128 Bit store per cycle
- L1 per core, shared L2
- System interface shared between both cores



Virtual Memory

- Each application has its own unique address space
  - Isolation between processes
  - No restrictions which address regions can be used
  - Shared memory possible if required
  - Multiple page sizes: 4 KiB, 2 MiB, 1 GiB (AMD 64)



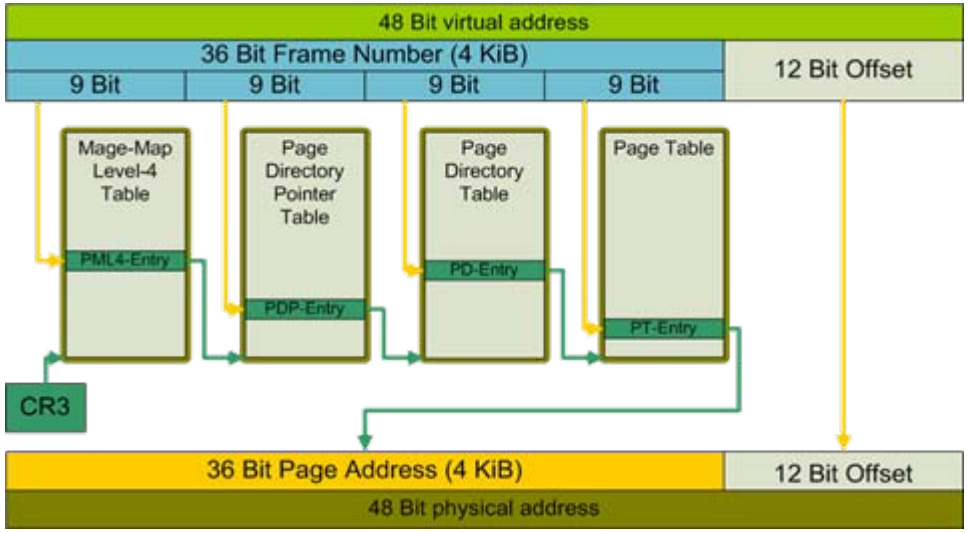


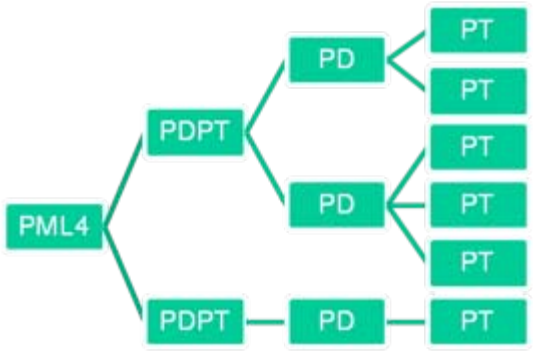
Fast Translation: Translation Lookaside Buffer (TLB)

- Stores translations from virtual page addresses (frame numbers) into physical page addresses
- Very low latency required to provide physical addresses for accessing caches
  - Multilevel TLBs are common, fast L1 TLB, larger L2 TLB
  - Limited number of entries
    - Usually hardly enough 4 KiB entries to cover today's cache sizes
      - E.g. Core i7: 8 MiB L3 cache, 512 4 KiB entries (covers 2 MiB)
      - Usage of 2 MiB pages (hugetlbfs) can reduce TLB misses significantly (e.g. Core i7: 32 entries cover 64 MiB)
- Transparent for applications
  - Not programmable, entries are generated on demand
  - Operating system can invalidate TLBs (e.g on context switches or after freeing memory)
  - TLB miss results in complex and slow address translation

Slow Translation: Page Table Walk (after TLB miss)

- Multiple stages of translation tables
  - Individual base address (from CR3 register) per process points to first table
  - Parts of virtual address used as index into tables
  - First 3 tables contain pointers to other tables, 4th table stores required physical address

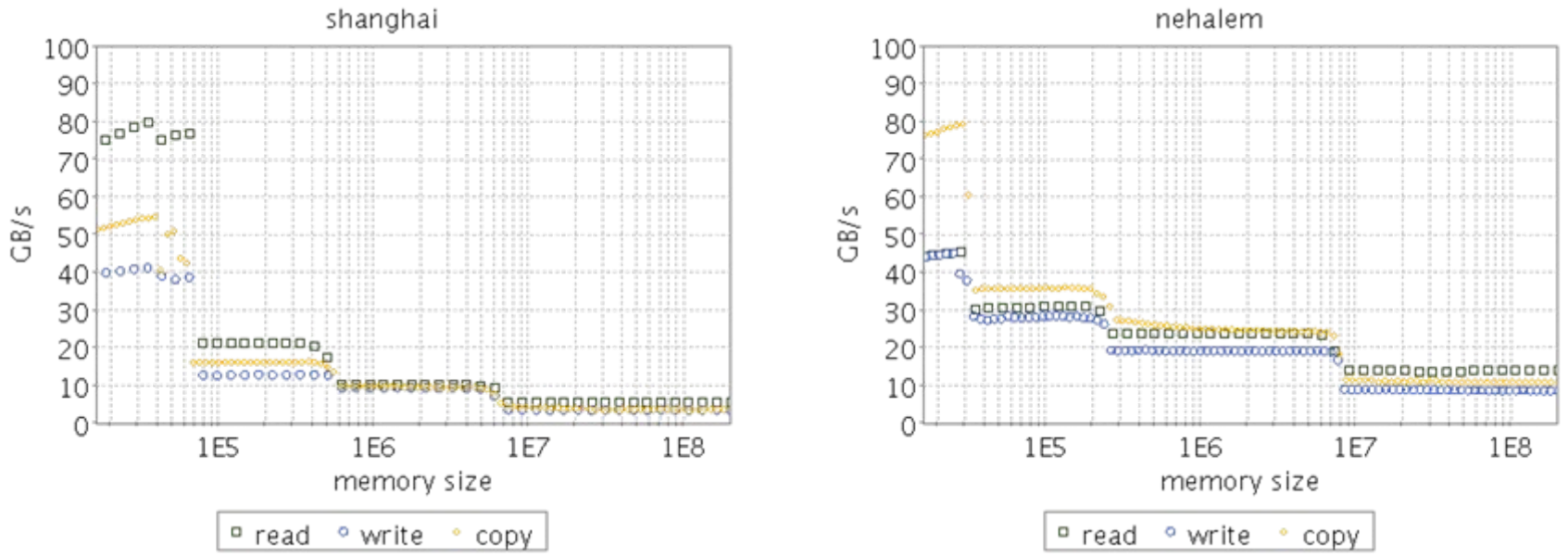




– only required tables are allocated
– requires less space than one big table

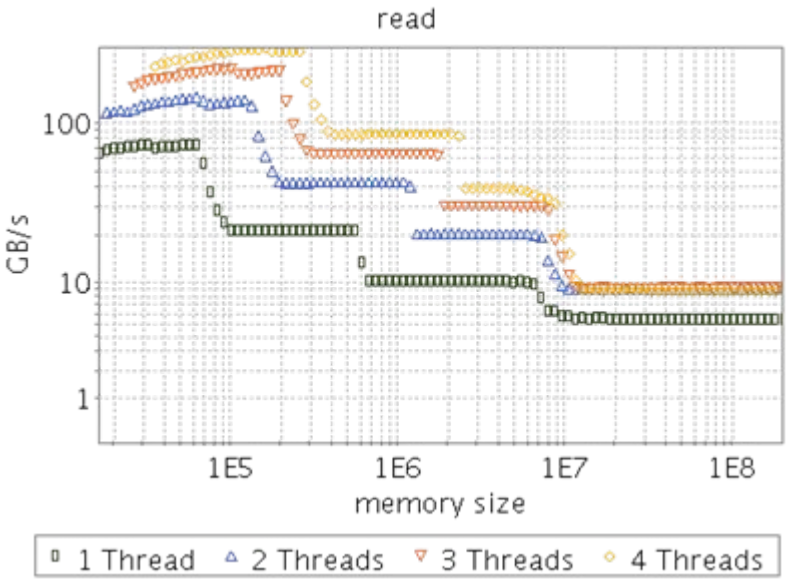
- Page Tables written by operating system, translation in hardware
  - 4 additional memory accesses to resolve physical address
  - After Translation result is stored in TLB for future use
  - Larger pages supported as well (2 MiB pages, 21 Bit offset, 3 translation stages)

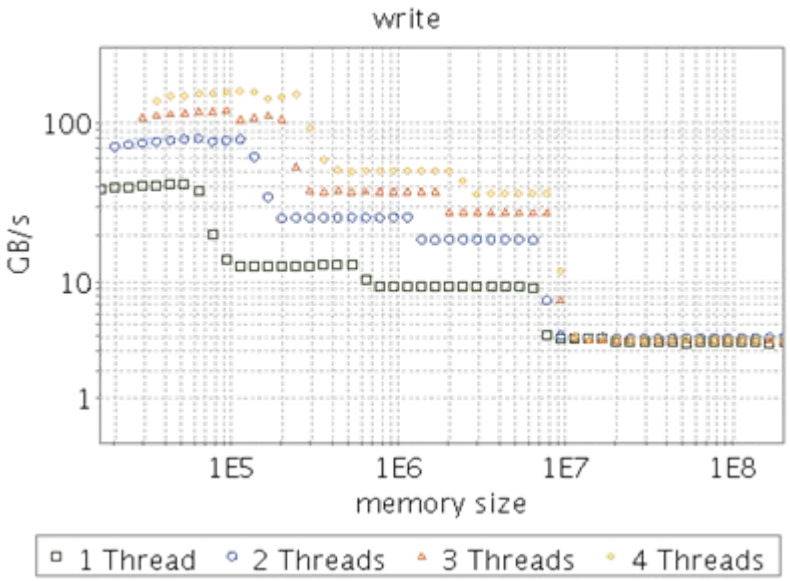
Example: Copy Bandwidth



- Opteron 2384 (Shanghai)
  - Dual ported L1 cache; 2x 128 Bit load/cycle or 2x 64 Bit store/cycle
  - 64 KiB L1-I, 64 KiB L1-D, 512KiB L2, 6MiB L3
- Xeon X5570 (Nehalem)
  - Dual ported L1 cache; 1x 128 Bit load/cycle and 1x 128 Bit store/cycle
  - 32 KiB L1-I, 32 KiB L1-D, 256 KiB L2, 8 MiB L3

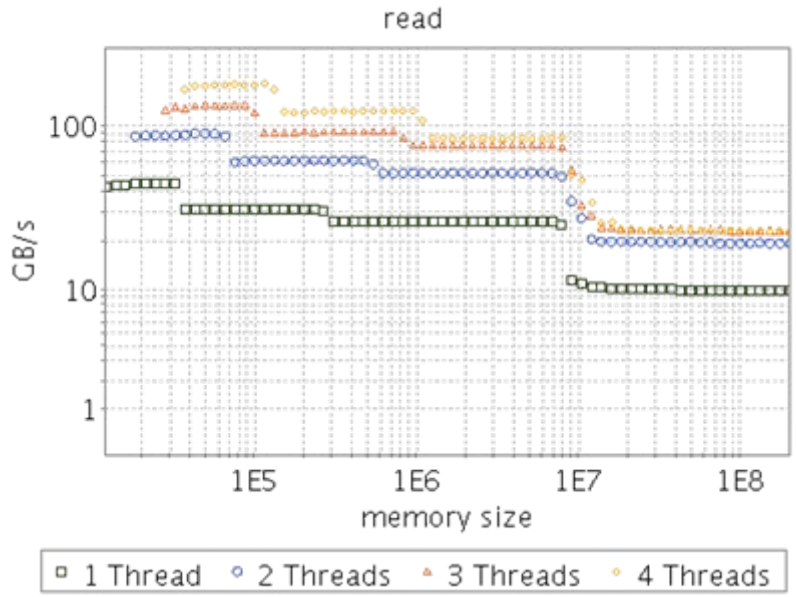
Example: Opteron 2384 multithreaded memory bandwidth

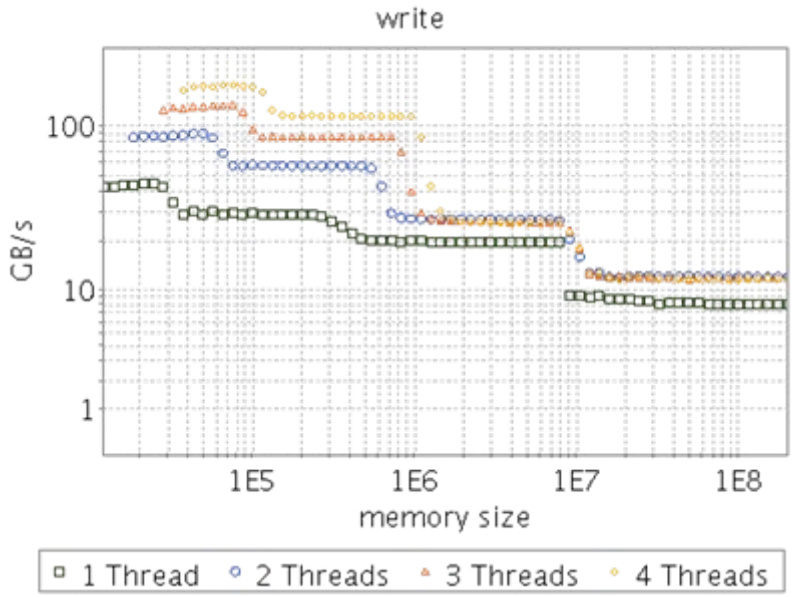




- All cache levels scale well for reading and writing
  - Cores do not compete for cache bandwidth (but for L3 size)
- 2 Threads required to fully utilize memory bandwidth

Example: Xeon X5570 multithreaded memory bandwidth

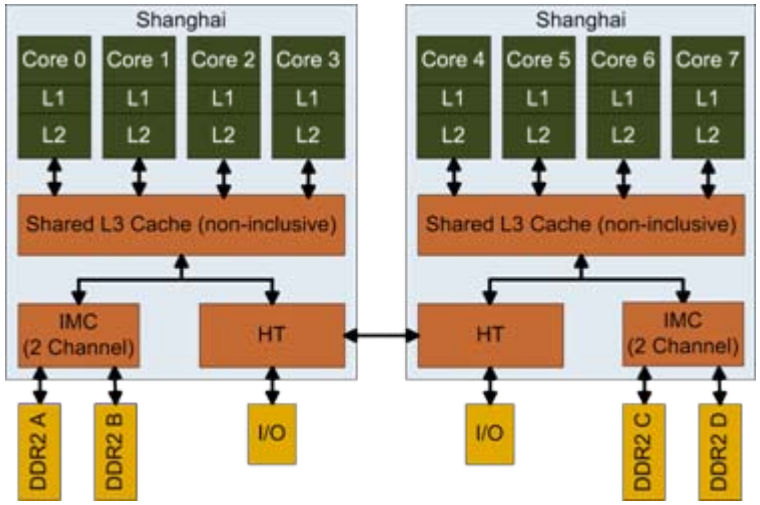




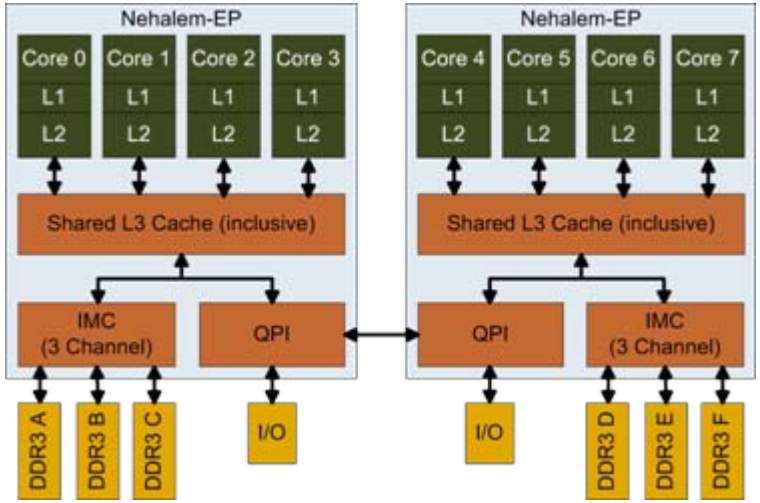
- L1/L2 scale well
- L3 read bandwidth scales well up to 3 Threads, L3 write bandwidth does not scale well
  - Cores compete for L3 size and bandwidth
- 3 Threads required to fully utilize memory bandwidth

Current 2 socket systems

AMD Opteron 2384

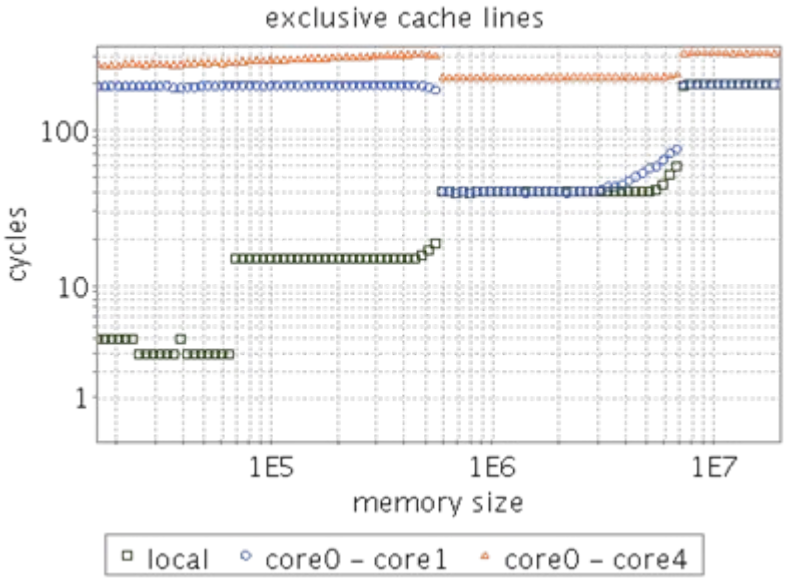


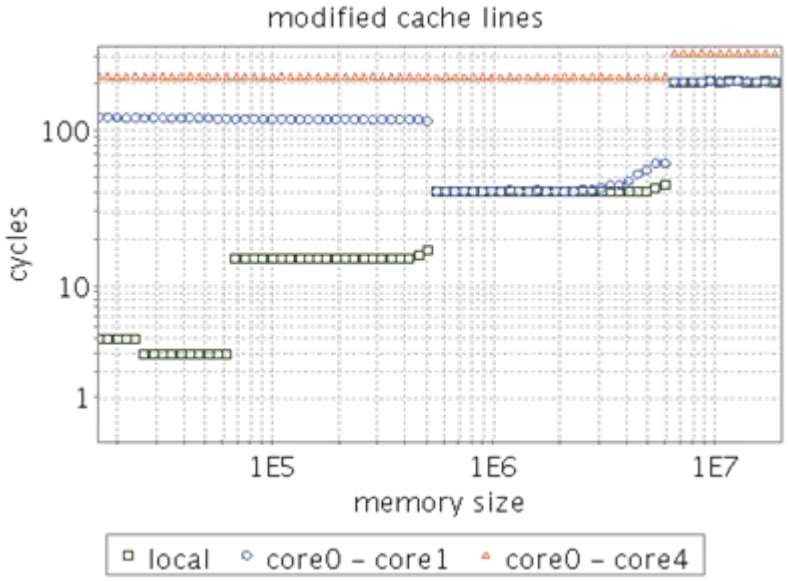
Intel Xeon X5570



- Comparable system architecture
- NUMA architecture
  - Each processor has its own memory controller
    - 2x DDR2-667: 10,6 GB/s je Socket
    - 3x DDR3-1333: 32 GB/s je Socket
  - Point-to-Point connections between processors
  - Low latency to local memory
  - Higher latency to remote RAM
  - Limited interconnect bandwidth
    - HT 1.1: 8 GB/s
    - HT 3.0: 17,6 GB/s
    - QPI: 25,6 GB/s

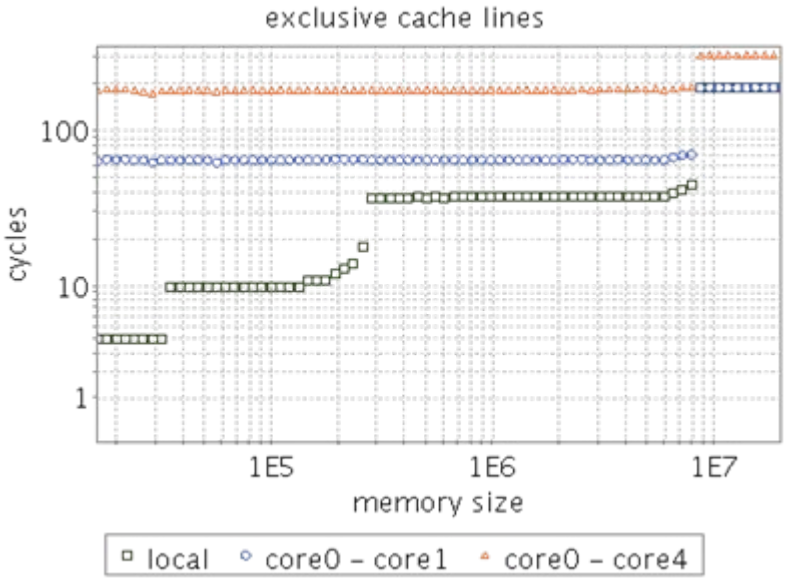
Example: Opteron 2384 memory latency

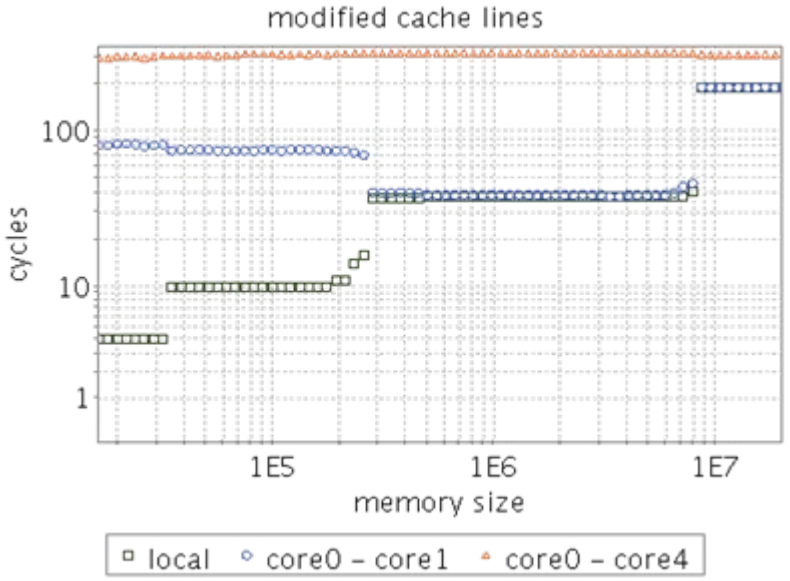




- Faster access to modified cache lines in other cores then to exclusive cache lines
- L3 cache much faster than accesses to other cores local caches
- Remote caches slower than local memory
- Remote memory access adds about 40 ns

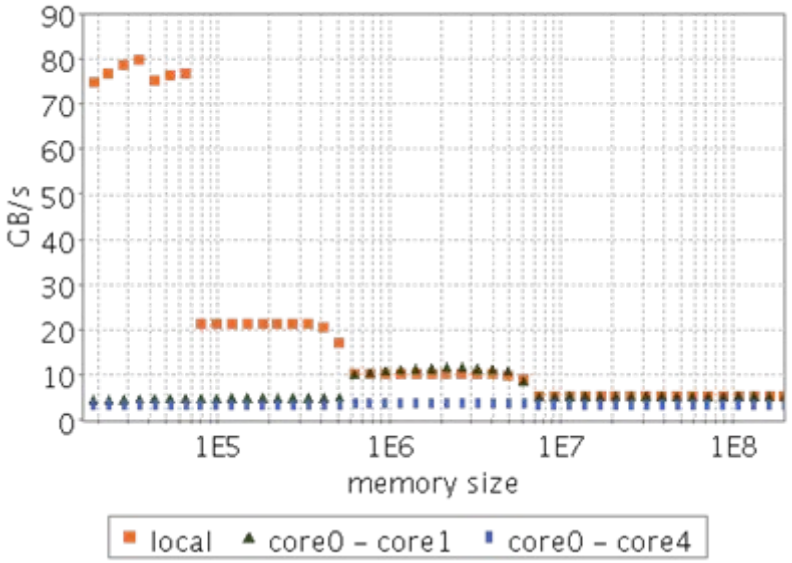
Example: Xeon X5570 memory latency

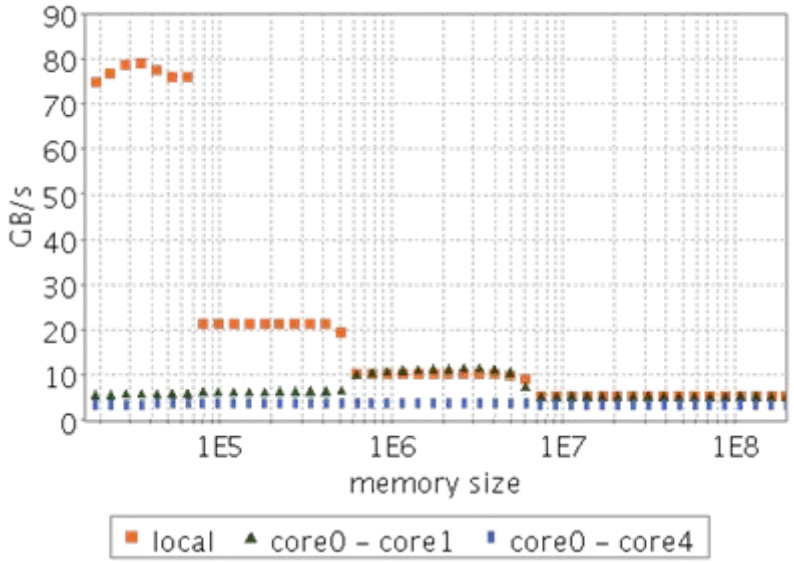




- Inclusive L3 cache handles requests for unmodified data in other cores
- Fast access to modified data in other cores on the same die
- Accesses to modified cache lines of the second processor causes write back to main memory
- Remote memory access adds about 40 ns

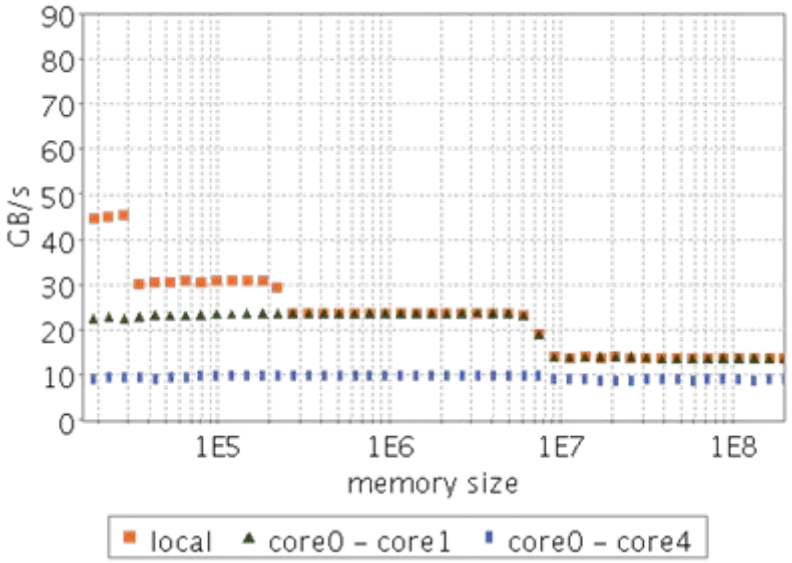
Example: Opteron 2384 memory bandwidth

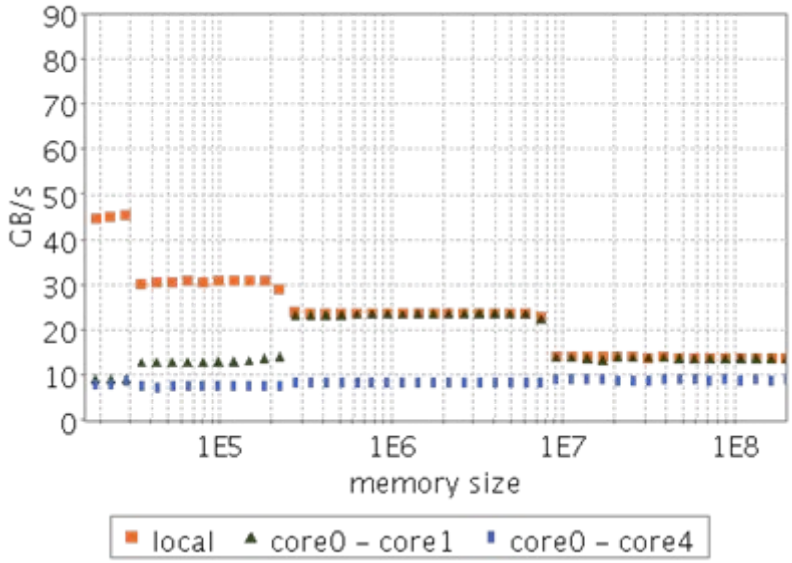




- Bandwidth to other cores on the same die as low as main memory bandwidth
- Accesses to the other processor limited by HyperTransport

Example: Xeon X5570 memory bandwidth





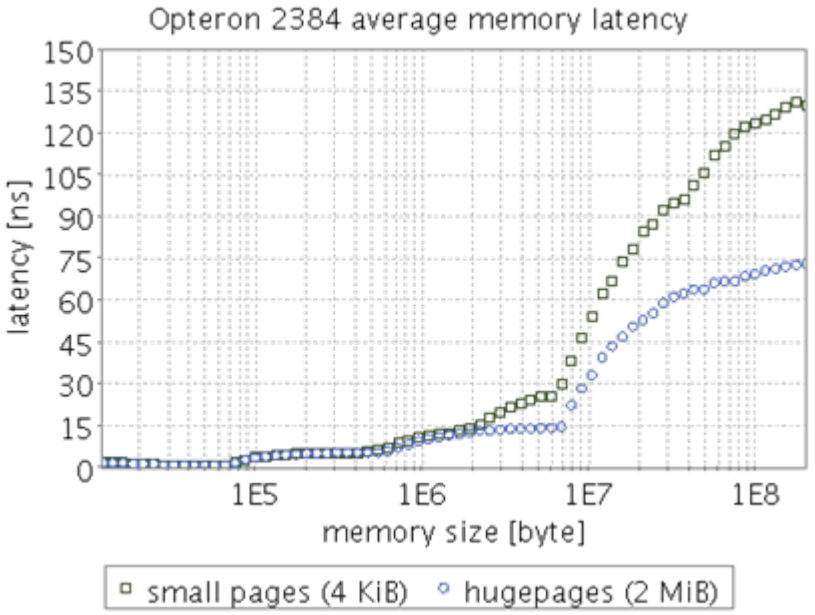
- High bandwidth for on-die accesses to unmodified data
- Low bandwidth for cache-to-cache transfer of modified data
- Accesses to the other processor limited by Quickpath Interconnect

Optimizations – SIMD

- icc automatically vectorizes loops if –O3 is specified
  - Replaces 4(single) or 2(double) loop iterations with one using SIMD
    - Significantly improves performance
    - Use simple loop constructs (e.g. for(i=0;i<n;i++){…})
      - no for(;;){…}
      - Check compiler output if performance critical loops were vectorized
  - #pragma vector aligned
    - Hint for the compiler to use aligned load and store instructions
  - #pragma vector non temporal
    - Hint for the compiler that written data is not accessed again
    - Writes data directly into main memory, avoids cache pollution

Optimizations – Hugetlbfs

- Needs kernel support (enabled by default in most current distributions)
  - /proc/meminfo shows available number of hugepages
- Has to be mounted as virtual file system
  - echo #pages > /proc/sys/vm/nr_hugepages
  - mount –t hugetlbfs nodev /mnt/huge
  - chown root:users /mnt/huge
  - chmod 777 /mnt/huge



- Create file in hugetlbfs and use mmap() to allocate memory
  - fd = open(filename,O_CREATE|O_RDWR,0666);
  - buf = mmap(0,size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
- Can significantly reduce TLB misses

Further Information

- Intel 64 and IA-32 Architectures Optimization Reference Manual
  - http://www.intel.com/products/processor/manuals/
- Software Optimization Guide for AMD Family 10h Processors
  - http://support.amd.com/de/Pages/techdocs.aspx
- AMD Software Optimization Videos
  - http://developer.amd.com/documentation/videos/pages/SoftwareOptimizationVideoSeries.aspx