Scaling Linux for HPC and Financial Applications

Linux User Summit 2009

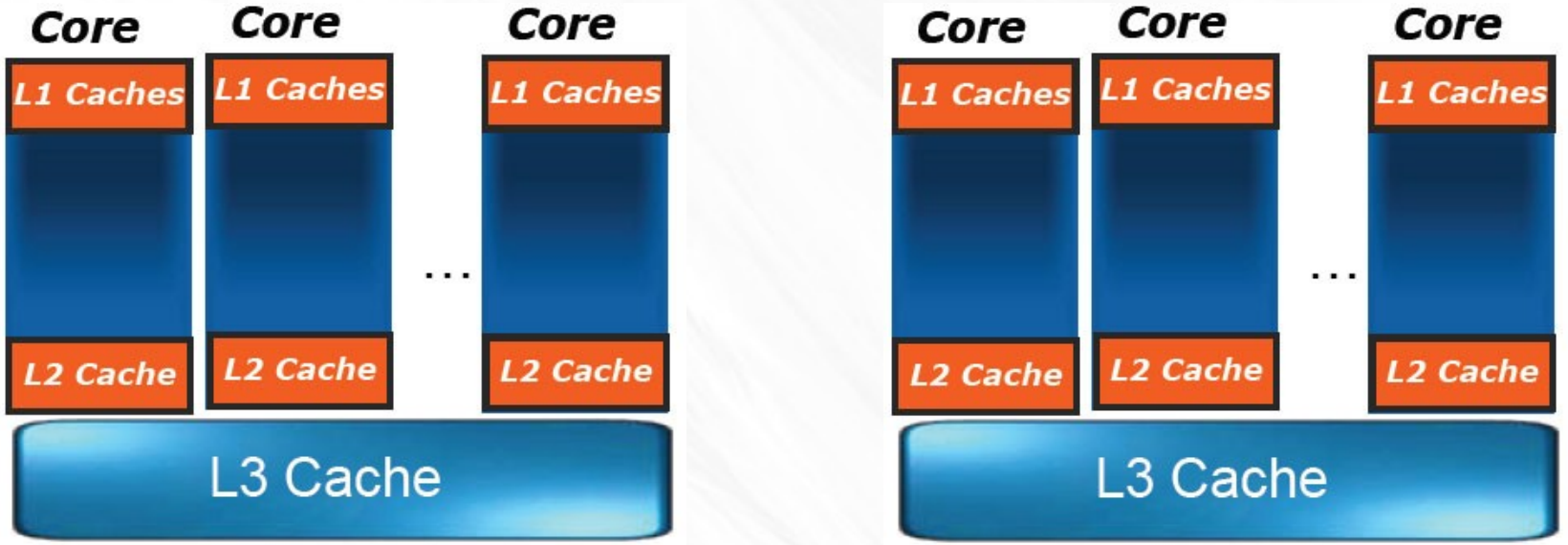
Christoph Lameter

Modern processors

- Phase where clock frequency increased
- Phase where the concurrency increases
- Performance can only be reached with multi-threaded applications.
- Memory bottlenecks (FSB) leads to distributed memory systems even on a single motherboard.
- NUMA effects begin to become a factor

2009 Nehalem Systems

- Standard server: Dual Quad Nehalem 2.93 Ghz
- 4 cores per socket
- Hyper threaded
- 16 hardware threads total

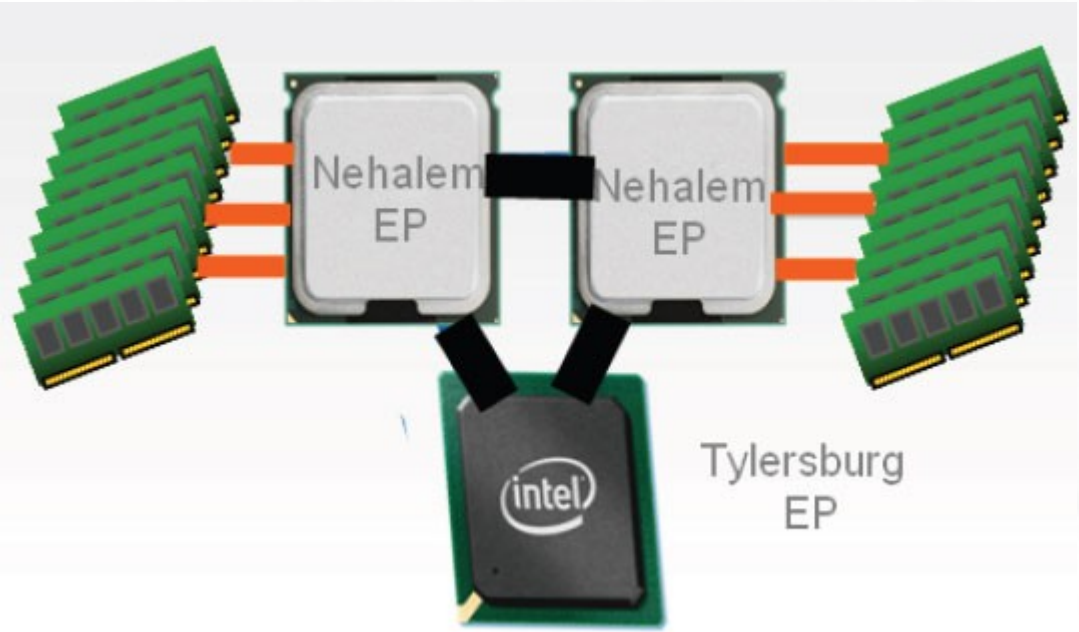


Implications

- Performance is dependent on presence of data in cpu cache. Access to system memory is extremely slow.
- Full sharing of all caches only between threads hyperthreaded on the same core.
- Threads on the same socket share the same L3 cache. If they execute the same code then cachelines must be refetched for L1 and L2 caches.
- Cache warming effects.
- Performance of a code segment varies depending on where it is placed in the system.

NUMA Memory Architecture

- Memory is local or must be reached via the other socket.
- Two NUMA nodes (today more to come)
- OS must manage locality of memory assigned to processes.



NUMA allocation strategies

- Kernel has a NUMA subsystem to control memory allocations.
- Default is to allocate memory local to the processor on which a thread is *currently* running.
- Not adequate for data that is shared between lots of processes. *Interleaving* is then necessary to avoid overloading a processor.
- Devices (such as network interfaces and storage) may be local to some memory and work faster if threads are running “near” the device.

2010: Nehalem EX

- See EX announcement: 8 cores per socket, hyperthreaded.,
- Quad Socket systems: 64 threads, 4 NUMA nodes.
- Octo Socket: 128 threads, 8 NUMA nodes.
- By 2015 Intel expects 512 threads in an average system.
- Memory sizes in the Terabyte range become possible.

Supercomputers on the Desktop

- NUMA and high processor counts were typical for Supercomputer in 199x and 200x.
- Likely becoming a standard feature next year for high end machines desktops and servers alike.
- Special technology (Nvidia GPUs) can give you thousands of “threads” today.
- Performance of computers is increasing significantly.

How to get maximum performance

- Hardware caching needs to be exploited to maximum effect.
- The OS can only provide heuristics.
- If well implemented then the OS can only assume that past behavior will continue in the future.
- OSes in generally are not doing a good job with scheduling in the complex environment of todays processors.
- Scheduler is optimized for maximum job throughput of the overall system. It is not designed to make a specific thread run with maximum performance nor to allow a process to react in time.
- Manual tuning is necessary for minimal latency, highest performance or predictable response times.

Realtime

- Supportive features by the OS to limit latencies by decreasing overall effectiveness of processing.
- Realtime features requires OS expertise and application changes to be used in a meaningful way.
- The OS can generate very long latencies for any memory allocation, I/O action or memory reclaim even if “Realtime” features are available and enabled.
- Predictable response times are only possible if expertise exists that allows one to assess what potential latencies can be created through which system call.
- Realtime features have to be disabled to have a system with minimal latencies and optimal performance.
- Realtime is in Linux is a fuzzy term that refers to a variety of techniques implemented at various levels. There is no consistent meaning.

Data Locality

- Spatial
  - NUMA nodes
  - Virtual to physical mappings TLB.
  - DRAM “pages”
- Temporal
  - L1, L2, L3 cache

Optimal Single Thread

- Simple because no caches are shared.
- Warm up needed.
- Memory locality is an issue
- No scheduler interruptions.
- Run process with sufficiently high priority so that another process is not run on the same processor.
  - **renice -5 <process>**
  - **chrt -f 90 <process>**
- Pin the process to a processor so that the process cannot be moved by the scheduler.
  - **taskset –cpu-list <processor> <process>**
- Force allocation from local memory.
  - **numactl –membind <localnode> <process>**

Dual Threaded Process

- Process should be bound to a hyperthreaded pair of threads if the individual threads do not exhaust core resources.
- If core resources are a problem then different cores are needed. This will require additional cacheline fetches from main memory (or L3) because the L1 and L2 caches are separate.
- The commands from the last slide can be reused. Just the taskset command needs to be changed:
  - **taskset –cpulist <processor1>, <processor2>**

Multi threaded

- It is advantageous to restrict a process to a single socket and its memory as long as possible.
- Single node means that all memory will be local as long as the memory is forced to come from the local node and the processes are restricted to the cores of one socket.
- If more threads are needed than available on a socket then memory also may have to be reassigned to still be local to a process.
- Memory then has to be categorized into data accessed by a single thread or a thread group local to a socket and memory that is shared between threads running on multiple nodes.