High Performance Computing: The move from general purpose processors to custom hardware and the implications for Linux

Open Source Summit North America 2017
Christopher Lameter, Ph.D.
Jump Trading LLC
(September 12, 2017)
Unix - The simple OS

- CPU emergence in the 70s designed for **general processing** suitable for any algorithm.
- Performance increases through faster execution of instructions.
- Unix is designed for such a processor.

"UNIX is simple. It just takes a genius to understand its simplicity"

-Dennis Ritchie

(Creator of Steve Jobs, Linus Torvalds, Bill Gates)
The Golden Age of POSIX APIs (1970 - 2010)

- Everything is a file in a Hierarchical structure of files
- A simple set of operations (read, write, seek ... )
- POSIX code can run on any OS (Windows, Unix, Linux ....)
- POSIX code is competitive performance wise.
- Optimizations on OS with limited effects on user space applications.
Early emergence of improvements requiring a code redesign (1990-)

- **SMP Multiprocessing**: Code needs to be restructured to optimally use multiple execution cores on the same processes address space. The Hardware now determines the code layout. Synchronization becomes necessary and thus nasty races that could not happen before.

- **NUMA**: Memory has different costs to access depending on locality. Data layouts need to be restructured to take advantage of memory placement.

- **Processor and TLB caches**: Accessing data in the same cache or recently used data is more effective. Access patterns need to be optimized to exploit processor caches. Temporal Data locality and Spatial locality become important.
Performance vs. “Portability” Tradeoffs

**Golden Age Code**
- Tasks determined by the algorithm
- Do not care where memory comes from
- Variable use and placement determined by ease of reading the code

**Performance optimized**
- Tasks created to use existing hardware execution units.
- Memory layout determined by available NUMA nodes.
- Variables placed in cache lines for optimal cache footprint and to avoid cache line bouncing
- Memory footprint of functions designed to fit in instruction or data cache for optimal performance.

*High Performance requires coding to the hardware characteristic of the target system. The OSes role as a way to abstract from the hardware is compromised.*
Later offload and bypass technologies for devices

- Stateful offloads for NICs (2004, rejected by IP stack then Iwarp etc supported by RDMA subsystem)
- Stateless offloads for NICs (2004-2010)
- GPUs (2001-2008)
- Many Core coprocessors (Xeon Phi etc)
- RDMA (2009)
- Direct Device access from user space (DPDK, RDMA API use for messaging)
- FPGA
New Performance features for Processors

- Vector Processing (AVX, AVX512)
  - Compilers can optimize code to use vector processing but loops must be written in a certain way to be recognized. New compilers.
- Non Volatile RAM support (2017/2018)
  - Support as a fast disk works
  - Support as “RAM” still has issues that have not been worked out.
- Custom processors (especially on ARM platform) with special logic to offload key processing (f.e upcoming Cavium ThunderX2)
- ARM embedded solutions with a long history in small devices
Simple APIs are still there but ......

- Move away from Golden Age computing is forced by the hardware that one wants to exploit for full performance. Initially this was only done for HPC computing but now it becomes commonplace.
- There are always new types of offload techniques that optimize more special cases.
- As time progresses Golden Age code is losing the game becomes an academic exercise for beginners.
- We have no choice but to endorse the rise of the specialized hardware.
- FPGAs, custom processors and custom ASICs will go to further extremes.
- Performance requires catering to the hardware. Simple Unix APIs will never give you the ultimate performance and are constrained by the paradigm.
- We try to hold on to the various existing APIs (more or less POSIX) in the kernel. Constant war back and forth between the interests of keeping APIs general or support custom APIs to exploit the last bit of performance.
- The ultimate radical solution is to use low level access from user space.
One needs General Purpose Processing to support most of these techniques

- Issue of administration, control and monitoring. It is unrealistic to implement these on complex hardware devices (FPGA, GPU, many core devices).
- That is where the existing performance challenged General Purpose paradigm succeeds.
- Easy to write code for that purpose without the efforts to support high performant code
- Survival of the Golden Age approach in a limited context.
- Result is that in practice machines will have both components. A control instance that is slow but easy to program and a highly specialized performance sensitive component.
- Critical path processing, data flowing through the unit, performance sensitive processing will be done by the offloaded logic.
FPGA technology

- The dreams of custom configurable hardware has come true!
- Extremely complex
- Difficult to define a standard API for operations since it is so highly configurable
- Highest processing power
- Development takes a long time.
- Often highest performant solution would be possible with an FPGA or custom ASIC.
- Tradeoffs become evident with the challenge and cost to master complexity vs a more standard solution that does not have full performance but will do the job.
Non Volatile Memory

- Another example of a bypass technology to go directly to storage without Intermediate OS action to manage storage.
- Use filesystem to then directly manage non volatile memory mapped to the processor address space.
- Reads and writes happen directly on the medium.
- OS administrates and manages but has to get out of the way.
- New file systems are being written and Non Volatile memory can be treated as a disk but then we will not have full performance.
- Direct memory mapped file contents require extensions to the processor and there are additional user space considerations (as expected) mostly related to data coherency and ability to survive crashes.
Algorithmic Floating point offload

- Specialized computation engines focusing on scaling floating point operations
- Weak integer, control path support
- Often combined with vector operations
- High Bandwidth memory
- GPUs (Nvidia)
- Many Core (Xeon Phi, EZchip, PEZY etc)
-
NPUs (Neural Processing Units for Machine Learning etc ....)

- Offload of special logic for machine learning.
- First only for applying learned behavior (language composition, ....)
- Then also for actual training of neural nets.
- Google calls this TPU and developed a special ASIC for it
- Processors are coming out with NPU support integrated in the processor (newest Huawei Cell Phone processor)
- Specialized form of matrix processing.
QPU (Quantum Processing Units)

- Another future field
- Qubits
- Highly experimental and disputed
- QuBits (Quantum bits) neither 0 nor 1.
- There is a vendor (DWave) selling quantum computers
  (It is disputed if this is actually a true quantum computer…)
- Real quantum computer being prototyped in various research labs. We are at around 50 qubits.
- Specialized processing is forwarded to the QPU from the main processor.
- It is likely that this may end up as another processor extension to offload special processing that benefits from quantum computing.
-
Operating System Implications

- The Operating System is a management layer and not a data processing layer
- Since it is not performance critical the OS can add complexity and more overhead (Freedom to add more bloat to cover more features)
- OS unable to compete with offload technologies. Let’s give up and endorse the bloat.
- Thus we can add more security overhead, containers, isolation techniques, sophisticated memory management.
- Whenever data processing hits the OS there will be a dramatic slowdown. Thus the demand for putting stuff into hardware offload will be increasing as bloat increases.
- The golden age for FPGAs is upon us.
- OS is going to be more and more commodified.
- There is a trend to have the Universal OSs that is striving to do everything (but slowly ...)

15
What to aim for (and what is currently going on in many areas)

- Create new abstraction layers and new APIs that allow abstraction for classes of accelerator
  - Graphics Display
  - Floating point: Vectorization
  - GPU floating point
  - FPGAs: Create generic management tools to program, start and shutdown. Diagnostic tools.
  - Custom ASIC
  - Storage engines
  - Remote Storage access (NVMEoF)
  - Network offload (direct send and receive from user space)
  - Remote memory operations (RDMA)
  - Non Volatile Memory
RDMA hierarchical ioctl scheme (New API by Matan Barak)

- Arises from the need on the one hand to support common functionality on many devices but also specialized functionality on some devices
- Extension of RDMA verbs and a chance from the write() based technique
- Hierarchical object attributes and methods
- Management of meta data in kernel that are required for operations on devices
- Methods that create special operations for devices
- First pieces merged into Linux v4.14.
- Here is another way of abstraction interaction with devices that have a direct data path bypassing the kernel but also allows the kernel to maintain administrative control.
Predictions

- The reign of Redhat, Centos and Ubuntu will be supreme despite continuing to suffer from bloat.
- Other Linux distros will slowly but surely lose their niche.
- OS developers will be generalists and the OS will be a big heap of bloat that only the select few can understand. The establishment will rule until their retirement.
- If you want to be working in Performance Engineering then learn FPGA and hardware skills.
- FPGA and hardware engineers will make lots of $$$$ in the next years and the specialization will give you job security.
- Custom hardware with general purpose controller logic running Linux is or will be the standard model for many use cases requiring high performance.
- Multiple instances of Linux are likely to have to interoperate in any device.
Suggestions and Comments?
Contact me at cl@linux.com