A Multicore Communication Architecture for Distributed Petascale Computing

Wenji Wu*, Phil Demar*, Matt Crawford*, Xian-he Sun**
* Fermilab, P.O. Box 500, Batavia, IL, 60510; ** Illinois Institute of Technology, Chicago, IL, 60616
CHEP09, 21-27 March 2009, Prague, Czech Republic

Abstract. Distributed petascale computing involves analysis of massive data sets in a large-scale cluster computing environment. Its major concern is to efficiently and rapidly move the data sets to the computation and send results back to users or storage. However, the needed efficiency of data movement has hardly been achieved in practice. Present cluster operating systems usually are general-purpose operating systems, typically Linux or some other UNIX variant. UNIX was developed more than three decades ago, when computing systems were all single core. Computation intensive applications and timesharing were the major concerns. Though the UNIX OS family has evolved through the years, Unix network services are not well prepared for distributed petascale computing. The proliferation of multi-core architectures has added a new dimension of parallelism in computer systems. In this paper, we describe a Multi-core Communication Architecture (MCA) for the distributed petascale computing environment. Our goal is to design OS mechanisms that optimize network I/O operations for multi-core systems. In our proposed architecture, MCA vertically partitions CPU cores on a multi-core system, allocating cores for either computation or communication, respectively. Cores dedicated to communication perform TCP Onloading. MCA will dynamically adjust core partitioning, based on detected system loads. CPU cores could be dynamically reassigned between communication and computation. Combined with Receive-Side Scaling and flow pinning technologies, MCA would perform flow scheduling to ensure interrupt- and connection-level affinity for TCP/IP processing.

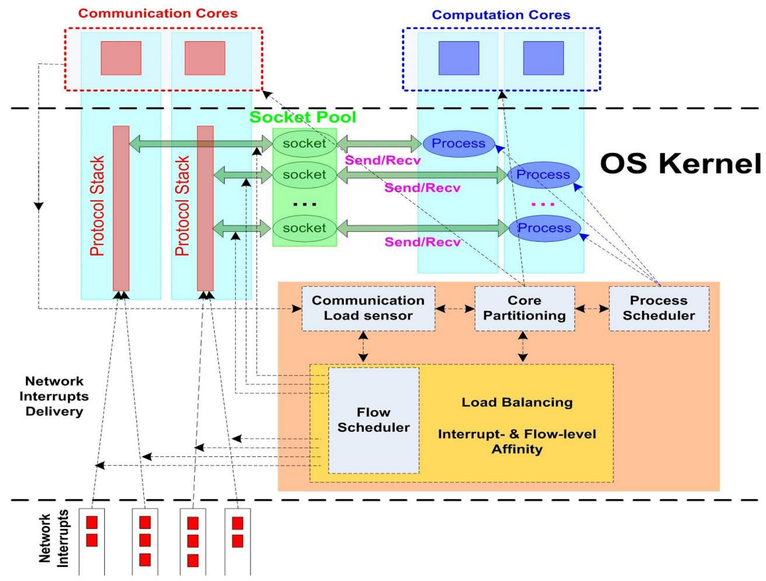
1. Background

- Demands for distributed petascale computing
  - Technical computing
  - National Security
  - E-Commerce
- For distributed petascale computing, industry and scientific institutions have created data centers comprising hundreds or thousands of commodity servers to develop massively-scaled highly distributed cluster computing platforms
  - large-scale task parallelism or data parallelism
  - The datasets involved commonly reach petabytes or tens of petabytes per year, and are growing; data access is mainly through network I/O
- The major concern of distributed petascale computing is to efficiently and rapidly move data sets to computation and send results back to users or storage. However, the needed efficiency of data movement is hardly achieved in practice:
  - The notorious Memory- and I/O wall problems
  - Inefficiency inherent in the middleware and distributed system software in the end systems
    - Present OS network I/O services are not well prepared for distributed petascale computing.
- Multicore architectures have become the pathway to higher performance computing. Multicore has added a new dimension of parallelism and requires a re-thinking of accepted engineering practice for single core system.

2. The communication problems in General purpose OS

- General purpose operating systems focus on different strategies to “fairly” share limited resources among tasks. With this principle, an OS is essentially a library of I/O and process management functions. I/O services are passive, and invoked only when requested by tasks. This model’s inability to support network I/O intensive distributed petascale applications is manifested in the following ways
  - Communication intrude on computation.
  - Communication protocol processing is not clearly separated from computation.
  - Communication protocol processing is inefficient in multi-core systems.

3. The Design of Multicore Communication Architecture (MCA)



MCA provides an integrated solution to perform efficient network I/O operations to reduce data access delay. Logically, it has three components: the host protocol stack, network application interfaces, and operating system supports.

4. MCA Logic View

- *Communication Load Sensor.* It monitors, collects and accounts various system loads on communication cores.
- *Core Partitioning.* It partitions CPU cores into two sets for communication and computation respectively. The dynamic core partitioning mechanism decides how many and which cores are assigned for communication; Communication cores will perform TCP Onloading.
- *Process Scheduler*. It coordinates with *Core Partitioning* and schedules threads only upon communication cores.
- *Protocol Stack.* On each communication core, Protocol Stack is invoked to perform TCP Onloading by interrupt threads and is executed in the interrupt context. Processes interact with Protocol Stack via the conventional socket interfaces. We will re-implement the socket system calls and adapt them to MCA mechanisms.
- *Flow Scheduler*. It adaptively directs and distributes incoming and outgoing traffic flows across communication cores.

5. Initial Results

- We have implemented the packet receiving mechanism for TCP Onloading and a simple core partitioning mechanism. Figure 1 shows the experimental results of running iperf data transmission. In the figure, red bar represents the TCP Onloading with communication separated from computation; blue bar represents the original Linux communication architecture. The figure shows that iperf is performing better for TCP Onloading with communication separated from computation. The performance improvement is up to 11%. The improvement is more apparent at higher throughput when network I/O is more intensive.

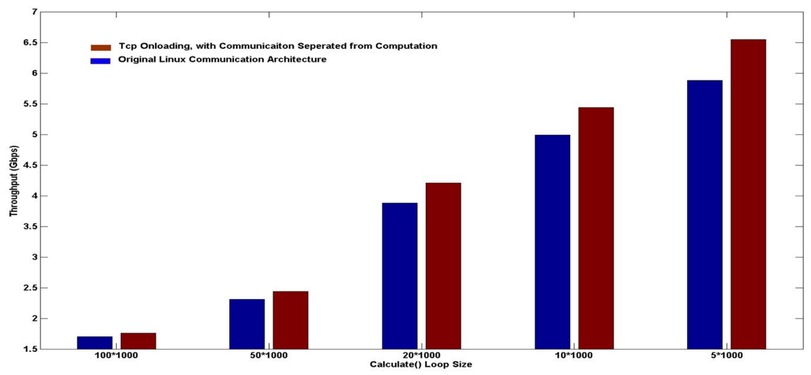


Figure 1 Throughput Comparison

Computing Division
Fermi National Accelerator Laboratory