# The 2 PetaFLOP, 3 Petabyte, 9 TB/s, 90 kW Cabinet: A System Architecture for Exascale and Big Data

Bruce Jacob, *Senior Member, IEEE*
Electrical & Computer Engineering, University of Maryland, College Park MD, USA — www.ece.umd.edu/∼blj

**Abstract**—We present a system architecture that uses high-efficiency processors as opposed to high-performance processors, NAND flash as byte-addressable main memory, and high-speed DRAM as a cache front-end for the flash. The main memory system is interconnected and presents a unified global address space to the client microprocessors. A single cabinet contains 2550 nodes, networked in a highly redundant modified Moore graph that yields a bisection bandwidth of 9.1 TB/s and a worst-case latency of four hops from any node to any other. At a per-cabinet level, the system supports a minimum of 2.6 petabytes of main memory, dissipates 90 kW, and achieves 2.2 PetaFLOPS. The system architecture provides several features desirable in today's large-scale systems, including a global shared physical address space (and optional support for a global shared *virtual* space as well), the ability to partition the physical space unequally among clients as in a unified cache architecture (e.g., so as to support multiple VMs in a datacenter), pairwise system-wide sequential consistency on user-specified address sets, built-in checkpointing via journaled non-volatile main memory, memory cost-per-bit approaching that of NAND flash, and memory performance approaching that of pure DRAM.

## 1 Motivation

Current high-performance systems fill rooms and dissipate on the order of 10 megawatts. To reach exascale, high-performance systems will have to increase performance by 100× without dissipating any more power—i.e., more performance, more capacity, and more bandwidth must all be delivered at no cost. A system design is proposed, targeting computation efficiency and addressing current limitations:

- Large systems dissipate significant power, often in the megawatt range for petascale computing capabilities, with the most efficient high-performance machines running in the range of 1–5 GFLOPS per Watt [10][11].
- Large systems either have limited per-socket main memory capacity [2][3], or they provide high capacity at extremely high price points (factors of 10–100x the cost/bit of consumer main-memory systems) [9].
- The per-node power is high: for example, the POWER8 chip alone dissipates 350W, and the per-node memory systems often dissipate power on par with that of the processing components [7]. Note that 1TB of DRAM dissipates roughly 100W, just in refresh.
- The network file systems can represent a significant bottleneck, particularly in those systems that use checkpointing to extend their application runtimes.
- The programming models typically do not allow easy sharing of data across the machine, for instance by allowing shared pointers system-wide.
- Systems are not easily partitioned, such that different threads (e.g., different VMs) can be assigned different amounts of memory, beyond what is on a single node.

Our proposed system uses low-power microprocessors [6] instead of high-performance processors, for high-density system packaging and better energy efficiency. It uses NAND flash as a load/store-addressable main memory to reduce both memory power and memory cost-per-bit; also, larger per-socket capacities reduce the need for network data movement. The system uses DRAM as a last-level cache to approach DRAM performance [5]. It journals the flash main memory system, to obviate the need for a separate file system. It networks the memory system to produce a flat, global memory space. It uses node-to-node signaling based on Moore graphs [1] to achieve low latency between remote nodes. As a result, the system supports 2.6 PB of main memory, achieves 2.2 PetaFLOPS, and dissipates under 90 kilowatts per cabinet. The resulting 20+ GFLOPS per Watt would put the design in the top tier of HPC efficiency (for comparison, the top computer in the most recent Green500 ranking achieves 5.3 GFLOPS per Watt [10]). Moreover, given enough I/O ports, it could scale to 200 PFLOPS, 250 PB of main memory, and a bisection bandwidth of 3,000 TB/s, at 10 MW.

*Manuscript submitted: 12-Mar-2015. Manuscript accepted: 18-Jun-2015. Final manuscript received: 22-Jun-2015.*

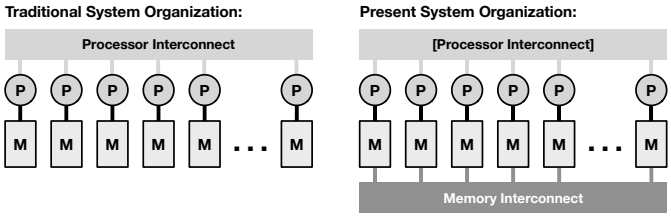


Fig. 1. Networking the memory system

## 2 System Architecture

As shown in Fig. 1, typical systems network together their processing components; this complicates addressing across nodes, as pointers are not easily shared. Instead, by networking the memory, it becomes trivial to export a system-wide global physical (*or virtual*) address space, facilitating applications such as large graph algorithms. In general, the processor interconnect could still remain, for example to provide a facility for explicit message-passing or to handle coherence traffic for any caches.

### 2.1 The Move to Flash for Main Memory

Once, SRAM was the storage technology of choice for all main memory systems [13][12]. However, when DRAM hit volume production in the 1970s and 80s, it supplanted SRAM as a main memory technology because it was cheaper and denser. Though DRAM was slower than SRAM, it was argued that an appropriately designed memory hierarchy, built of DRAM as main memory and SRAM as a cache, would approach the performance of SRAM, at the price-per-bit of DRAM [8]. Today,

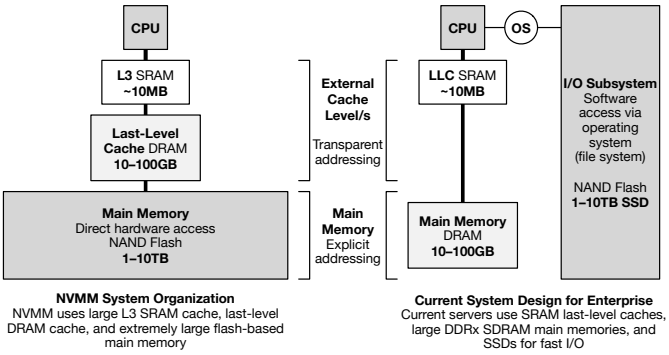


Fig. 2. The memory components at each node

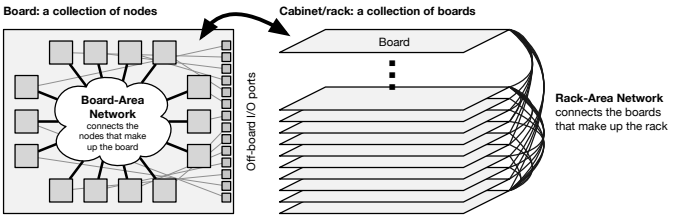


Fig. 3. The board-level and rack-level networks

we could speed computers by up to an order of magnitude if we built main memory out of multiple gigabytes of SRAM instead of DRAM—but this option is not even considered, because to build that system would be prohibitively expensive.

Today we face a similar inflection point. For reasons both technical and economic, we can no longer afford to build ever-larger main memory systems out of DRAM. Flash memory, on the other hand, is significantly cheaper, denser, and lower power than DRAM and therefore should take its place. It is now time for DRAM to go the way that SRAM went: move out of the way for a cheaper, slower, denser storage technology, and become a cache instead. While it is true that flash is significantly slower than DRAM, one can afford to build much *larger* main memories out of flash than out of DRAM, and we have shown that an appropriately designed memory hierarchy, built of flash as main memory and DRAM as a cache, will approach the performance of DRAM, at the price-per-bit of flash [5].

Our memory organization is shown in Fig. 2: NAND flash is used as the main memory technology, and DRAM is a cache for flash. Among other things, this allows per-node capacities starting at 1TB, without an increase in power dissipation, and no significant loss of performance [5]. The performance difference is roughly a factor of two compared to a 1TB DRAM system (which would be prohibitively expensive to build).

### 2.2 Network Topology

The cabinet network is partitioned into a two-part hierarchy, shown in Fig. 3. The system is a collection of boards, each a collection of nodes. The network topology is based on Moore graphs, from early computing systems [1]; Fig. 4 illustrates, using the Petersen graph as an easily-visualized example. These high-radix networks maximize the total nodes, given a max latency (node-to-node hops) and a fixed number of I/O ports per node. For example, the Petersen graph has ten nodes, each with three nearest neighbors, and a maximum latency of two hops between any two nodes. These tend to compare favorably with meshes and even high-radix networks such as butterfly and dragonfly topologies. For instance, a 2-hop dragonfly or flattened butterfly network (e.g., rook's graph) with $2n$ ports

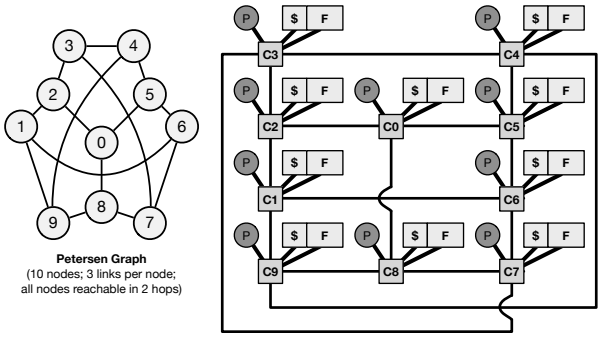


Fig. 4. The Petersen graph as a network with near-planar layout

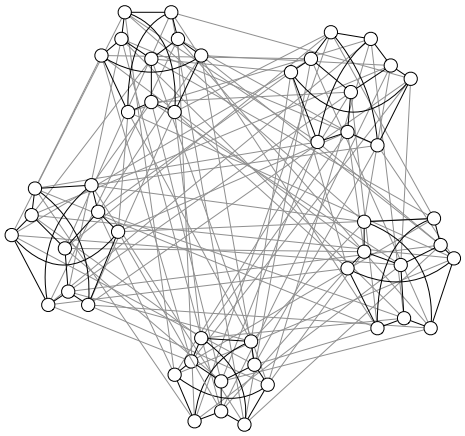

Fig. 5. Hoffman-Singleton graph as five disjoint Petersen graphs

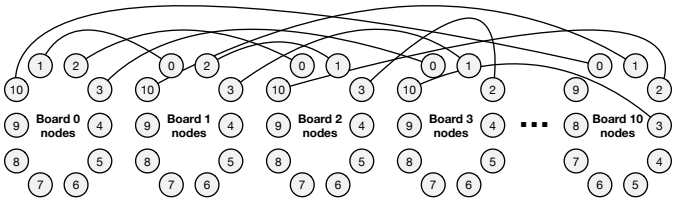


Fig. 6. Naming convention for rack-level network, simplified

per node supports a maximum of $(n+1)^2$ nodes ($n$+1 fully-connected nodes in each dimension), whereas a 2-hop Moore network using $2n$ ports has an upper bound of $4n^2+1$ nodes.

Each rack contains 51 boards, each of which has 50 nodes. The nodes on each board are connected in a Hoffman-Singleton topology (illustrated in Fig. 5), a collection of fifty nodes, each of which has seven ports, with a maximum latency across the board of two hops. As the figure shows, the Hoffman-Singleton graph contains five disjoint copies of the Petersen graph, and clustering nodes this way reduces the inter-cluster wiring.

To support static routing, and to use the same layout for each board (no need to custom design each board), we connect every pair of boards through additional ports. In general, one can construct, for any $n$-node board-area network, a rack-area network of $n^2+n$ nodes. For instance, the 10-node Petersen graph yields a 110-node rack-area network comprised of eleven boards. Fig. 6 shows the resulting rack-area network, based on the Petersen graph. The nodes in Board 0 are labeled 1..10 (no node 0); the nodes in Board 1 are labeled 0,2..10 (no node 1); the nodes in Board 2 are labeled 0..1,3..10 (no node 2); etc.; ... the nodes in Board 10 are labeled 0..9 (no node 10).

For all boards and nodes, the controller at Board X, Node Y

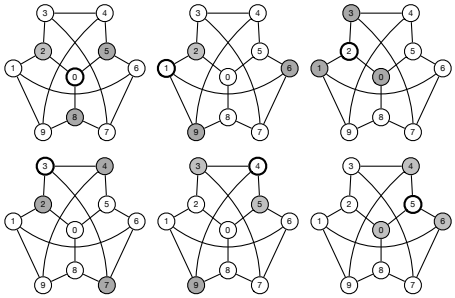

Fig. 7. Each node in the Petersen graph defines a unique 3-node subset lying at a distance of 2 from each other and a distance of 1 from the rest

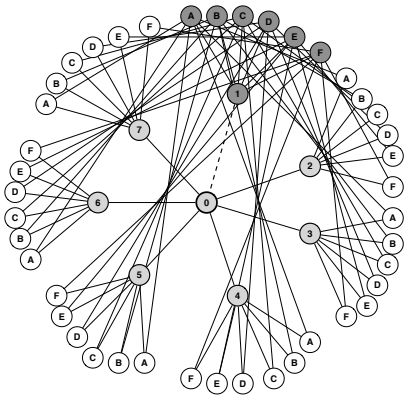

Fig. 8. Link fail: only the linked nodes & nearest neighbors are affected

is logically paired with the controller at Board Y, Node X: each controller has $n$ neighbors, and we connect those neighbors to each other (the $n$ neighbors of Node Y on Board X connect to the $n$ neighbors of Node X on Board Y). In a two-hop network, each node defines an $n$-node subset of nodes that are two hops from each other, and any other node in the graph is at most a distance of 1 from a node in the subset. Fig. 7 illustrates this in a 10-node Petersen graph.

In our network of 51 boards of 50 nodes each, every unique node has seven unique neighbors, and so there are seven links from Board 0 to Board 1, seven links from Board 1 to Board 2, and, generally, seven links from Board $i$ to Board $j$. This doubles the I/O ports on each node, from seven to fourteen ports. It also increases the maximum hop count from two to to four, but it extends the network to 2550 nodes. A reference to a node within the same board will take at most two hops. A reference to a remote board will first go to a gateway node for that remote board, and, by definition, a gateway node is at most a distance of one hop from every node on the board. Thus, the maximum number of hops is four: at most one local, one board-to-board, and at most two across the remote board.

The bisection bandwidth is relatively high. Each node has 14 inter-node links; 7 connecting to nodes on the same board; 7 to other boards. Each link is bidirectional at a conservative 2GB/s, for low power link I/O. Two 1024-byte messages can be sent in opposite directions on a link every microsecond. 35,700 simultaneous messages can be in-flight at any given moment, half of those between nodes on different boards. Cutting through the middle of the network, at the narrowest part, yields $26 \times 25 \times 7$ links $\times$ 2GB/s, for a total of 9.1 TB/s.

### 2.3 Network Redundancy

In general, the redundancy in networks based on Moore graphs is similar to that of meshes or butterfly/dragonfly networks. When a link goes down, all nodes in the system are still reachable; the latency simply increases for a subset of the nodes. Fig. 8 shows the Hoffman-Singleton graph (relevant links only, for clarity), with the link between nodes 0 and 1 cut. Node 1 and its remaining nearest neighbors are shaded in dark grey; node 0 and its remaining nearest neighbors are shaded in medium grey. As the figure shows, only communications involving node 0 or node 1 are affected: the latency between node 0 and node 1 is now 4 and can take any route, and the latency between these nodes and the neighbors of the other is now 3 and can take any route (we use random routing in the case of link failure).

### 2.4 Global Memory System

The per-node capacity is extremely large, to reduce inter-node data movement as much as possible. At each node, a highly banked, multi-channel DRAM system serves as a cache for a highly banked, multi-channel flash system, and the DRAM cache uses a large block size that matches the flash page size. Due to the large latency disparity, is important to prefetch constantly, and we have found that even simple $n$-block lookahead works well, as do the aggressive SSD-prefetching heuristics in the Linux file system [5]. Each flash channel can operate at up to 800 MB/s [4], and using multiple channels enables prefetching on otherwise idle channels while handling demand-page data (which occupies a channel for a long time: $\sim$10,000 cycles).

To simplify the development of extremely large inter-node address spaces, we have networked the memory system, such that a global 64-bit address space can be exported and shared across multiple clients without need for pointer swizzling or any other translation. Among other things, this supports applications with extremely large data sets, such as irregular graph applications and in-memory databases. It also supports the easy partitioning of the entire space into different sections to support different applications, such as VMs, that require their own dedicated resources, though some of those resources may be remote. As described earlier, the latency between nodes is at most four network hops, and the topology and its bisection bandwidth support thousands of simultaneous transfers.

While the system offers a global address space, it does not, however, offer coherent caches; if cache coherence is desired, it must be provided by the user application. The memory system provides a primitive that guarantees pairwise sequential consistency between data objects, across all 2.6PB of main memory, by co-locating objects thus paired on the same home node, and therefore access to the object pairs is serialized by definition [5]. However, as should be obvious, this guarantee does not hold if the user code caches the data objects.

### 2.5 Node-Level Power and Processing Capabilities

Computation at each node is handled by a low-power, high-performance multicore processor. Our initial design targets the 64-bit Bostan chip from Kalray, which combines 256 VLIW cores on one chip. These chips run Linux, support MPI and OpenMP, and are programmed in C, C++, and Fortran. Each chip delivers max 845 GFLOPS and dissipates 24W at 800 MHz [6]. Each node has 64GB of DDR3 SDRAM and 1TB of NAND flash, each of which dissipates 4W under load. The I/O links dissipate an additional 3W, giving 35 Watts per node, and a cabinet-level total under 90 kW. The peak rate is 0.845 TFLOPS per node, for 0.845 $\times$ 2550 = 2155 TFLOPS at the rack level. This yields a power-efficiency rating of over 20 GFLOPS per Watt, comparing favorably with typical high-performance systems, which are typically well below 10 GFLOPS per Watt.

For larger systems, though we could use a more traditional multi-cabinet fabric like InfiniBand, we would prefer to

continue scaling out the Moore-graph network topology. This would require a chip design with significant I/O requirements (several dozen I/O ports for chip-to-chip communication), which no current commercial CPU offers. We are leaning toward the option of designing a custom NIC for this purpose. For example, were one able to build a 50-port low-power NIC, one could connect 100 racks at 2,500 nodes per rack, with a system-wide four-hop maximum latency (all 250,000 nodes in the system would lie at a max distance of 4 from each other), and a bisection bandwidth of 3,000 TB/s.

### 2.6 Journaled Main Memory

Flash memory lends itself to a journaled organization, if one retains the most recently overwritten values instead of immediately garbage-collecting them. When new data is written to an existing page, NAND-type flash requires the new data to be written to a new flash page. Typically, the new flash page is taken from a free list maintained by the controller, and the old page is placed on a dead list of pages to garbage collect, and the mapping for that page is changed to point to the new location.

Instead of deleting or overwriting the old mapping information, we keep it. Using flash makes main memory "permanent" in the same sense as a traditional file system, and retaining the previously written values makes the virtual address space recoverable as well. This effectively unifies the virtual memory system with the file system, thereby simplifying the operating system's design and providing built-in checkpointing. Numerous applications in the high-performance space rely upon checkpointing to make forward progress in the face of frequent device failures (in large-scale systems with millions of semiconductor devices, one-per-million failures become commonplace); journaling the main memory system means that, instead of dumping the entire address space's contents to a remote I/O device, a checkpoint is merely a synchronized cache flush.

Our design uses a direct-indexed mapping table kept in flash, cached in dedicated DRAM while the system is operating. Each entry of the table contains the following:

| | |
|---|---|
| 34 bits | Flash Page Mapping |
| 30 bits | Previous Mapping Index |
| 32 bits | Sub-Page Valid Bits & Remapping Indicator |
| 24 bits | Time Written |
| 8 bits | Page-Level Status & Permissions |
| 16 Bytes | Total Size |

The Flash Page Mapping locates the virtual page within the physical flash-memory system. A virtual page must reside in a single flash block, but it need not reside in contiguous pages within that block. The Previous Index points to the mapping for the previously written page data. Time Written keeps track of the data's age, for use in garbage collection. The Sub-Page Valid Bits and Remapping Indicator allow the data for a 64KB page to be mapped across multiple page versions written at different times and also allow for pages within the flash block to wear out. The Virtual Page Number directly indexes the table; the indexed entry contains the mapping for the most recently written data. As pages are overwritten, old mapping info is moved to free locations in the table, maintaining a linked list, and the indexed entry is the head of the list. Fig. 9 illustrates.

When new mapping information is inserted into the table, it goes to the indexed entry, and the previous entry is copied to an unused slot. Note the pointer value in the old entry is still valid after it is copied. The indexed entry is updated to point to the previous entry. The Previous Mapping Index is 30 bits, for a maximum 1B table entries, meaning that it can hold three previous versions for every virtual page in the system.

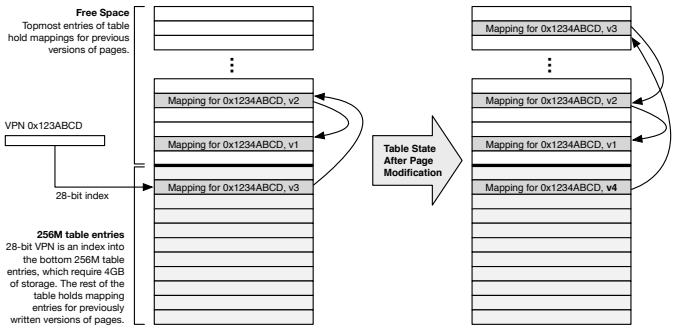


Fig. 9. A page table for journaled flash main memory

The Virtual Page Size is 64KB, and pages are written to flash at the granularity dictated by the flash device (for example, 8KB).

Note that the page size is simply a mapping construct: data is moved into the CPU at a load/store granularity, and data is moved between the DRAM cache and the flash backing store at a flash page granularity. The 64KB size is not a data-movement granularity; it is simply a mapping granularity that allows the page table to be reasonably small and yet direct-mapped.

## 3 Bottom Line

A new system architecture is presented that offers enormous data footprint, significant compute performance, and yet relatively low power dissipation. It offers a global physical/virtual address space, journaled non-volatile main memory, high cross-section bandwidth, low network latency, low per-bit memory costs (approaching that of NAND flash) and high memory performance (approaching that of DRAM). At the cabinet level, the system provides 2.6 PB of main memory, achieves 2.2 petaflops, and dissipates under 90 kilowatts. We have built a proof of concept to test both inter- and intra-board communications; we are currently developing our first full 50-node board.

We believe that this is an exciting building block for exascale computing and big data applications, as it performs well and can scale well. For instance, with a 50-port NIC at 10 W, one could connect 100 racks at roughly 2,500 nodes per rack, with a two-hop intra-rack latency, a four-hop inter-rack latency, and a bisection bandwidth of 3,000 TB/s. This would support 250 PB of main memory, exceed 200 PFLOPS, and dissipate 10 MW.

## References

[1] J. C. Bermond *et al.*, "Graphs and interconnection networks: diameter and vulnerability," *Surveys in Combinatorics*, pp. 1–30, 1983.
[2] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer-on-board memory systems," in *Proc. ISCA*, Jun. 2012, pp. 392–403.
[3] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, "Fully-buffered DIMM memory architectures," in *Proc. HPCA*, 2007, pp. 109–120.
[4] Intel, Micron, Phison, SanDisk, Hynix, Sony, and Spansion, *Open NAND Flash Interface Specification, Rev. 4.0*, May 2014.
[5] B. Jacob *et al.*, "A journaled, NAND-flash main-memory system," *Technical Report UMD-SCA-2010-12-01*, Dec. 2010.
[6] D. Kanter and L. Gwennap, "Kalray clusters calculate quickly," *Microprocessor Report*, Feb. 2015.
[7] P. Kogge et al., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. DARPA, Sep. 2008.
[8] J. Mashey, "Why no more SRAM main memories?" *The yarchive.net database, http://yarchive.net/comp/sram_main_mem.html*, 1999.
[9] J. Stokes, "MetaRAM quadruples DDR2 DIMM capacities, launches 8GB DIMMs," *ArsTechnica*, Feb. 2008.
[10] The Green500 List, *www.green500.org*, 2015.
[11] The Top500 List, *www.top500.org*, 2015.
[12] J. E. Thornton, *Design of a Computer—The Control Data 6600*. Scott Foresman & Co., 1970.
[13] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. of R. & D.*, vol. 11, no. 1, pp. 25–33, 1967.