DOI: 10.14529/jsfi170204

The Simultaneous Transmit And Receive (STAR) Message Protocol

Earle Jennings[1]

The STAR protocol is proposed, which solves three inherent problems with MPI, a well known security problem caused by data memory access faults, and the following four exascale communication problems. Exascale systems must efficiently save the state of their Data Processor Chips (DPCs) every fraction of a second to support rollback when an error is encountered. Exascale systems will have the possibility of frequent optical communications failures. Exascale systems must resiliently respond to these optical communication failures. An exascale computer needs to be fed enough data fast enough, and its data center must keep up. Optical implementations are developed compatible with 100 Gbit/sec Ethernet to solve these problems. Automatic fault resilience mechanisms are discussed, which improve HPC quality of service, and meet the exascale reliability and resilience challenges. The bandwidth problem for exascale computers interfacing with data centers is solved. The STAR protocol enables data centers and supercomputers, to be invulnerable to memory fault injection of viruses and rootkits.

Keywords: MPI, optical communications, exascale, HPC, security, router, memory wall, fault resilience.

Introduction: Problems to be Solved

MPI is a standard for message passing commonly found in parallel processing systems, in particular HPC systems [8], [17], [7],and [15]. While MPI is versatile and readily available, it has inherent problems. Figure 1 shows sending a message in MPI. The sending of the message locks up the buffer until the message has completed being sent. All instruction processing has to refrain from accessing the buffer until the buffer is unlocked. Often, the core must rely upon an interrupt structure to know when the buffer is again ready for access, implement a wait loop, or suspend a thread. This not only slows things down, but costs energy and increases complexity of the hardware and software operating environment.

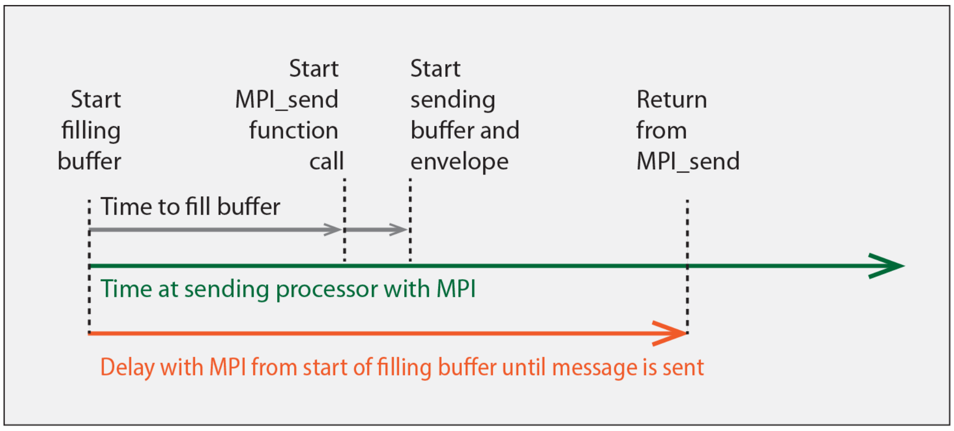
Start filling buffer
Start MPI_send function call
Start sending buffer and envelope
Return from MPI_send
Time to fill buffer
Time at sending processor with MPI
Delay with MPI from start of filling buffer until message is sent

Figure 1. Sending a message in MPI

Figure 2 shows receiving the MPI message not only locks up a buffer until the message is received, but also for the time required to process, or move, its contents elsewhere. Making this function efficient now requires that the operating environment know when this buffer mechanism

[1]QSigma, Inc., Sunnyvale, USA

can be accessed for a new operation. The operating envirnoment may use one or more of the following: status registers to trigger an interrupt in the core; implement a second wait loop; or suspend the thread. From an application program perspective, these operations appear to be primitives, but they come with a large amount of overhead, in terms of instruction processing to initiate these operations, wait for their completion and then release these resources to be used for a new operation. All of this takes energy, hardware, and computing time.

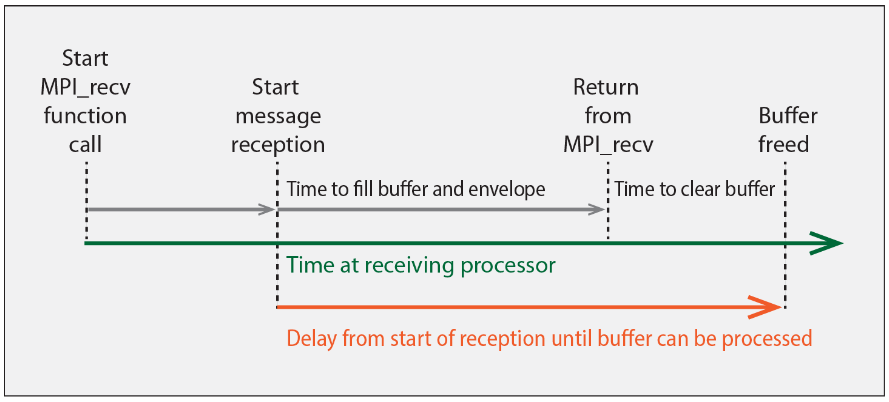
Start MPI_recv function call
Start message reception
Return from MPI_recv
Buffer freed
Time to fill buffer and envelope
Time to clear buffer
Time at receiving processor
Delay from start of reception until buffer can be processed

Figure 2. Receiving a message in MPI

With MPI, messages can be any length, and a short message can be stalled by a long message at a router transfer point, as shown in Figure 3. It should be noted that Intel has announced a component solution to this one problem, but it does not solve all three of these MPI related problems, much less the security, and the exascale related problems we consider next.

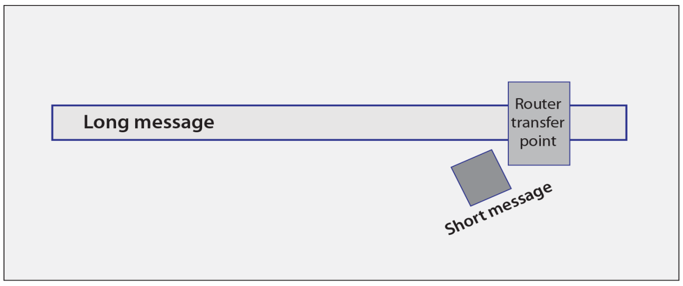

Figure 3. A short message stalled at a router transfer point by a long message

There is common security problem in processors derived from the PDP-11 and/or the IBM 360 [1]. A buffer is assumed to have a starting address, a length, and from this, a computed ending address. Suppose a computer starts writing at the starting address, but writes a string that is longer than the length of the buffer. In other situations, the writing goes toward the start of the buffer and writes data into addresses before the buffer's starting address. In either case, addresses outside of the buffer are altered. For example, these addresses can hold program instructions, pointers to code, such as the return address for a program during execution. These data buffer access faults can, and do, inject viruses or rootkits into the instruction memory of these processors [19]. Data fault injection turns seemingly innocent data into an injected virus, which can then access anything in the memory domain of the injected device. While this issue has been around for decades, it persists as a major security problem [13]. One of the earliest examples of this ocurred in 1988, when the Morris worm, infected DEC VAX and SUN computers running BSD Unix across the Internet [4]. This article shows how the STAR communications protocol supports rendering this situation impossible for a data center or a supercomputer.

Exascale brings with it four new problems.

1. Consider a Data Processor Chip (DPC) containing some increment of 20 Mbytes of internal state. An exascale computing system [3] will probably have at least several hundred thousand DPC's, each containing at least 500 cores. Further assume that for any specific fault, the system needs to be able to rollback back within a second, with an overhead of no more than a few percent, say 2%. Consider what is required when the DPC state is captured in a snapshot every 10 ms. Within that 10 ms, 2% of the time the DPC is saving the state, with each DPC operating on a local 1 ns clock. (All mention of a clock only refers to a local clock.) This implies that each 20 Mbytes must be saved in 2% of 10M clock cycles, or 200K clocks. Therefore, for each 20 Mbytes of internal state in the DPC, 100 bytes must be saved in a nanosecond. This is much more than the 100 Gbits/second optical fibers can support. While rolling back by a second seems natural, the question is how much of the system needs to roll back. By recording snapshots every 10 milliseconds, in many situations, the amount of roll back can be limited to a much smaller fraction of the system, saving energy and computing capacity.
2. Another problem is the failure rate for large scale optical networks. These failures can be defined as any message that is received, and through analysis of its Error Detection/Correction (EDC) envelope, is found to have detectable, but uncorrectable, errors. Failure rate is the number of failures per unit time. The overall communication failure rate of the exascale system [3] depends upon the failure rate for one optical fiber and its transceivers, and grows directly (but not necessarily linearly) based upon each of the following: The number of data channels per bundle interfacing to each of the chips; the number of optical fibers in each channel; the number of DPCs; the number of routers needed to avoid deadlocking these chips; and the complexity of the routers.
3. Analyses [3] of the maintenance logs of current US supercomputers indicate that the Mean Time Between Failure (MTBF) for exascale systems may be no more than minutes. When the optical communications fail, the system fails. With exascale systems, there is no time to rely upon an operating system, or a distant snapshot of the communications. This led the author to develop this multiple degree of freedom, fault resilience strategy, to be implemented in the local communications hardware.
4. There is a another, essentially mirror problem to the first exascale problem. How does one feed data into an exascale computer fast enough to keep it busy? This problem will be answered later at three system component levels, first at the DPC, second at the mostly binary topological network interfaces of the system components, and third at the cabinet to data center interface level.

1. Introducing the STAR Protocol

It is required for STAR message protocol compliance that any compliant device transmits and receives a STAR message on every local clock cycle, except when responding to an uncorrectable error upon reception. This requirement removes the three problems with MPI. Sending a message is completed in one local clock. Receiving a message is completed and buffer released in one local clock. Each STAR message clears each local pipe stage in the routers in one local clock, so that no message stalls another for an unknown amount of time. The response to such errors can involve automatic channel component replacement in, at most, a microsecond.

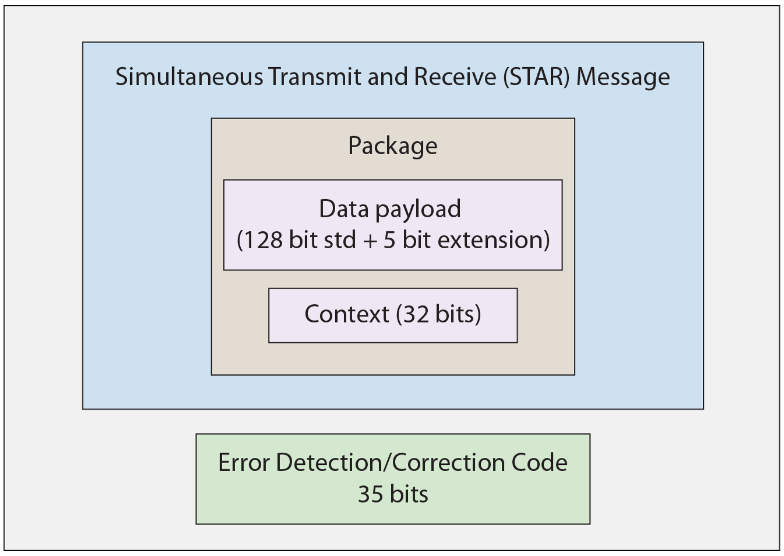
Simultaneous Transmit and Receive (STAR) Message
Package
Data payload
(128 bit std + 5 bit extension)
Context (32 bits)
Error Detection/Correction Code
35 bits

Figure 4. STAR message transport layer

Figure 4 shows the application layer of the STAR message, which includes a package containing a data payload and a context. The payload includes a 5 bit extension code and 128 bits of standard payload. The context is 32 bits, which is interpreted at every STAR message core to determine the package disposition and transfer. The context and its interpretation, is under complete control of the program.

The transport layer of each STAR message, also includes a 35 bit Error Detection/Correction (EDC) code. These 35 bits are arranged in clusters of 7 bits. Each 7 bit cluster relates to a separate group of 33 bits of the package. There are five groups covering the package. These 7 EDC bits allow the system, at each receiver, to calculate a 1 bit error correction of the received group of 33 bits. These same 7 EDC bits enable calculation of a 2 bit error detection flag for these same 33 bits of the received package. All of these EDC calculations are required to be performed within one, or more, locally clocked (nanosecond) pipe stages. The transport layer supports detecting and correcting up to five single bit errors, one in each of the groups of the application layer. The transport layer also supports detecting any, or all, of the five groups having uncorrectable 2 bit errors. The local clock is about 1 GHz, and is readily supported by standard CMOS, so that 200 Gbits/second can be simultaneously sent, and received, on each STAR channel.

1.1. Exascale State Resilience and Optimized Matrix Communication

The first example of the payload format, sets the top Extension bit (Ext bit 4) to 0, and the remaining payload is interpreted as two double precision floating point numbers, each augmented by two bits of the extension field, used as guard bits. This format can represent a complex number, as well as be useful in bulk downloading and uploading of the state of cores in the DPC. Note that each of these messages is transferring 16 bytes in one clock cycle, so that 8 of these channels delivers 128 bytes per nanosecond in and out. Recall that each DPC has some multiple of 20 Mbytes of internal state, which required 100 bytes per nanosecond to store, and restore, as discussed above with a reasonable overhead for application programs. This approach makes DPC state resilience feasible. The STAR bandwidth delivery, both as input and output, is a necessity for keeping an exascale system fed with data, which is further discussed in the following section on the optical implementation below.

The second format example configures Ext bit 4 as 1, and two of the other 4 bits as 0. This can be interpreted as a double precision number and its two guard bits with an index list of 64 bits. This format has two uses. In dense matrix manipulation, such as matrix inversion by Gaussian elimination with partial pivoting, or block LU decomposition [6], the format is used to communicate a pivot, or potential pivot candidate. In sparse matrix manipulation [10], [16], 4 bits of the index list can act as a field identifying sixteen large objects, such as sparse matrix A and several vectors associated with the matrix. This format configuration insures efficiency for both dense, and sparse, matrix operations. By configuring the context of a message, the second format can be interpreted as having more than one floating point number. For example, the 66 bits can be interpreted as dual single precision (each with a guard bit), or quad FP16 bit numbers. Further configurations using the context, can result in these 66 bits being interpreted as numeric values in a form of posit numbers [9]. With posit numbers, more opportunities are potentially available, for instance, 3 posit numbers, each of 22 bits in length, or 5 posits, each of 13 bits, or 6 posits, each of 11 bits. The net result of these opportunities is that as an application progresses in run time, it can use fewer parameters per payload, for greater precision.

2. STAR Bundle

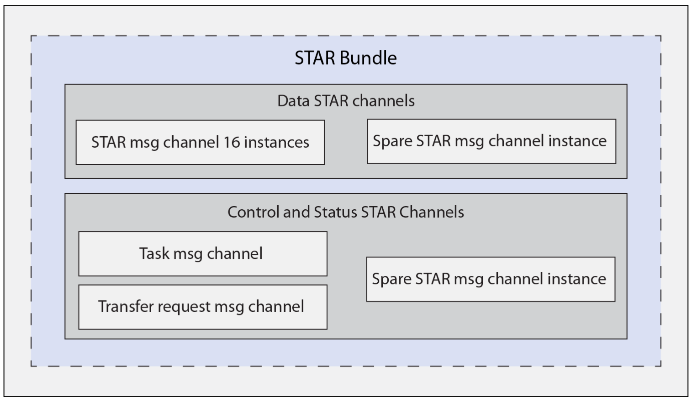
STAR Bundle
Data STAR channels
STAR msg channel 16 instances
Spare STAR msg channel instance
Control and Status STAR Channels
Task msg channel
Transfer request msg channel
Spare STAR msg channel instance

Figure 5. Example of a STAR bundle

Figure 5 shows a STAR bundle including data channels as well as control and status channels. There are 16 data channels, with a spare channel instance used for fault resilient recovery from unrecoverable message faults. The control and status channels include one task and one transfer request message channel, and a spare channel for fault resilient recovery. This example has several positive consequences. 16 data channels guarantee that the state of the DPC can be saved within a reasonable time overhead for exascale systems. Note that in the above discussion, we showed that 8 channels deliver 128 bytes per local clock, so 16 channels deliver 256 bytes per clock. This is a data payload bandwidth of 2 Terabits per second input and output of each link (bundle interface) for every DPC, each memory controller and every link of every router in the system. Every DPC can receive and send 32 double precision numbers on every local clock. This is compared to the Sunway [2], with a system interface at the MPE/CPE chip of 16 Gbytes/second = 128 Gbits/sec with a latency of 1 microsecond. The ratio of 2 Tbits/256 Gbits is roughly 16. The Sunway is roughly 93 Petaflops of performance, or 9% of an exaflop. Delivering 16 times the bandwidth for 11 times the performance has a reasonable expectation of reaching exascale system performance. The transfer request messages can remove all the main memory address calculations from the DPCs, sending instead just the necessary parameters for the transfers. This results in energy reduction for many programs in the DPC, by removing the address calculations from the DPC, and placing them in the memory controllers. Task messaging is carried out in a separate channel from data transfers and their requests, systematically separating data from task control and configuration. Therefore, data cannot be confused with task control and configuration infromation by these separate channels. This removes an avenue of injection for malicious software such as viruses and root kits.

2.1. STAR Communications Network

A STAR communications network is a point-to-point network of nodes, each acting as sources, destinations, and routing nodes, called STAR Trinary Routers (STRs).

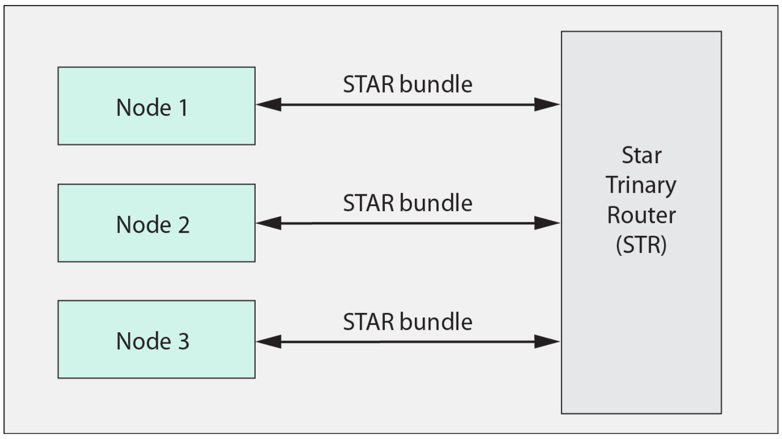

Figure 6. STAR Trinary Router

Each STR has three bidirectional STAR bundle links to nodes, or other STR instances, as shown in Figure 6. Each node may be a module within a chip, or a chip. The interface between a STAR bundle and data processing circuitry, whether in an STR, or a core module, is through a STAR bundle module.

Figure 7 shows a STAR in-chip network, including a binary graph of channel bundles, whose nodes are interfacing through bundle modules to a core, or core modules, called the Programmable Execution Modules (PEMs) 0 to 3. The STRs inside a chip are laid out as a communication module paired with a module of cores as a unit. In particular, the PEM and the STAR bundle module with the STR are laid out as a unit.

Figure 8, on the left side, shows a STAR channel core as part of STAR bundle module 1. The data STAR channel core interfaces a data channel of the STAR channel bundle shown in the middle. This interface includes an Incoming Message Processor (IMP) and an Outgoing Message Processor (OMP). The IMP and the OMP belong to the data STAR channel core 1. They interface with a core, possibly in a core module, or in the PEM, or in the STR. On the right side, each of the STAR channels of the second STAR bundle module interface through a corresponding channel core to another core, core module, PEM, or STR, either within a chip, or between chips.

Consider block LU decomposition [6] on a matrix with 32 million rows and columns, which is the approximate size required by the Linpack specification for a computer to be exascale. Assume the block size is 32 rows and columns. This matrix will be processed for eight or more

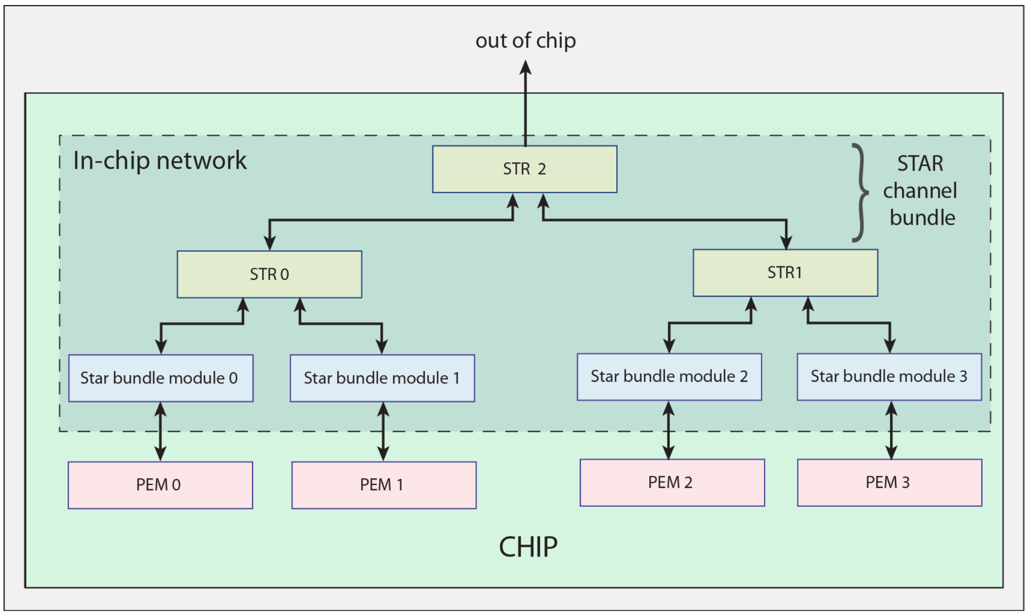

Figure 7. A STAR in-chip network

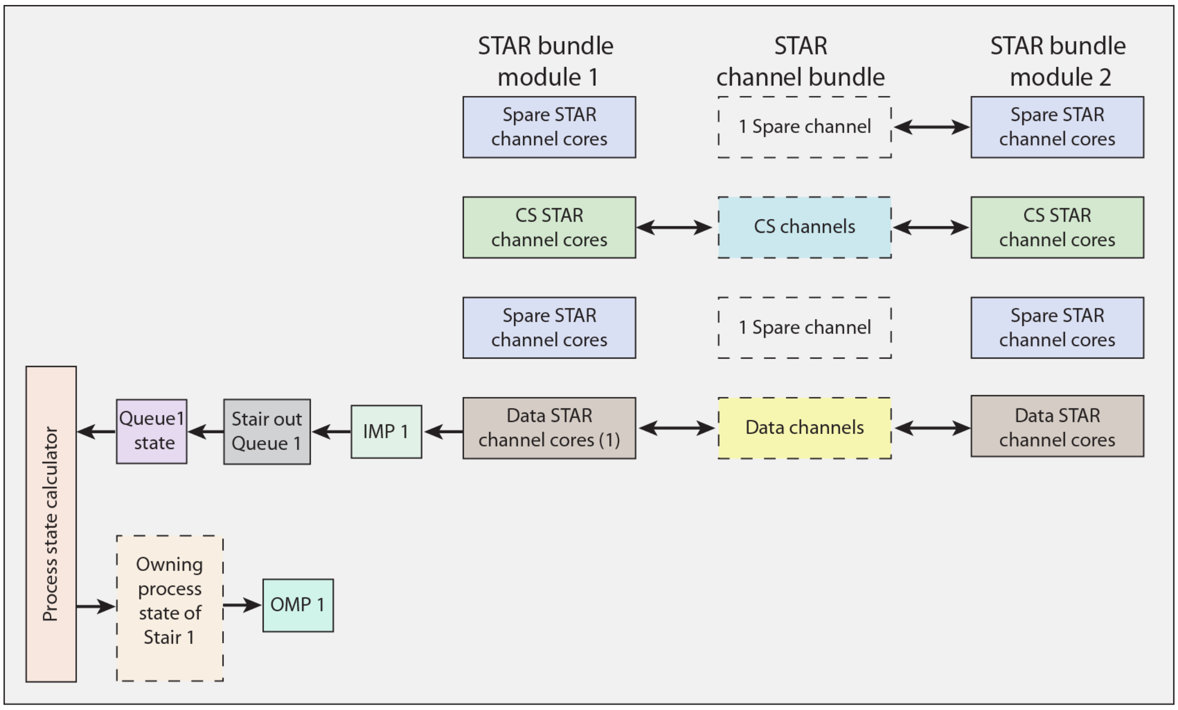

Figure 8. Interfacing a STAR channel bundle between two bundle modules

hours to meet the HPL standard of sustained performance of an exaflop [14]. At the start, the probability of row swapping is essentialy certain, as there is one chance in 32 million the pivot is in the diagonal entry. This persists for most almost all of the runtime of the algorithm, say for 32 million minus 32 of the diagonal entries. When processing of the next block column begins, that block columns needs to be loaded, which means that almost all of the rows have been swapped. This can be visualized as a swarm of insects all taking off and landing somewhere else in a random pattern. These swarms of messages must be efficiently delivered. For this to happen, each message needs to clear its inhabited pipe stage in one local clock, or the communication clogs, and the supercomputer will fail to perform Linpack at an exaflop. Also, if even one of these messages fails to be delivered, the Linpack benchmark may fail.

2.2. Optical Implementation of a STAR Channel

Communication in exascale systems requires optical communications for, at least, the following reasons.

- Assume a system includes the DPC chips arranged in chip stacks, on some form of Printed Circuit Board (PCB) interfaced through some form of motherboard to the rest of the system. Assume that the DPC states are stored in DRAM local to the chip stack so that the 20 Mbytes travels no more than 10 cm to its backup store. Otherwise communications on the PCBs is likely to clog.
- Delivering 800 bits per nanosecond into or out of one chip cannot be reasonably done with our classic level-shifted electronic signaling. Such signaling is best represented by LVDS (Low Voltage Differential Signal) protocols [5] for integrated circuits, which have a typical upper limit of about 1 bit per nanosecond per coupling. These couplings typically do not support bidirectional simultaneous signaling.
- 800 bits per clock requires at least 1600 pins, in each direction. Even in a ball grid array for a chip 5 cm on a side, this will saturate the active logic pins.

Electronically traversing 10 cm across 800 signal pairs is an RF noise nightmare with today's PCB technologies. While the basic PCB layout rules of thumb say that these 800 differential signal pairs are traveling at 1/2 the speed of light (30 cm per nanosecond) the quarter wave length of a 1 Ghz signal is 15 cm. However, the rising and falling edges of the differential signal pair have a large high frequency component of roughly 10% of the wavelength of the overall signal, or about 10 GHz. This means that anything more than about 1.5 cm can act as an antenna for the differential edges, so many of these signal paths can cross couple due to the antenna affect [20]. These very wide signal paths are assumed to be synchonized. Thirdly, consider the sheer number of interfaces. The roughly 500 K to a million DPC tells us, based upon the other problems just outlined, that the electromagnetic couplings of the past are not reliable enough for this function.

The above shows that communication between these DPC, their memory controllers in the chip stacks, and nodes connecting these chip stacks, PCBs and motherboards need to be optical in their transport of the data. The 100 bytes (800 bits) per nanosecond clock is 8 times the maximum rate of 100 Gbit optical Ethernet. This bandwidth must be achieved with a multi-fiber protocol approach. Consider multi-fiber links between these components in a HPC system sustainably generating 10^{18} floating point operations per second (flops). These links are going to require Error Detection and Correction (EDC) circuitry at each receiver for each end of each optical fiber. These EDC circuits are going to need to respond with good, corrected or uncorrectable error detection status very quickly. Each transmitter and receiver side circuit is going to have to fix itself locally, save and restore the communication stream disrupted by the detected error, and keep track of what was sent until it is certain that it was correctly received. Again, all of this needs to be local to its side of the link, or else the system can clog. Consider the individual messages communicated on a STAR message channel.

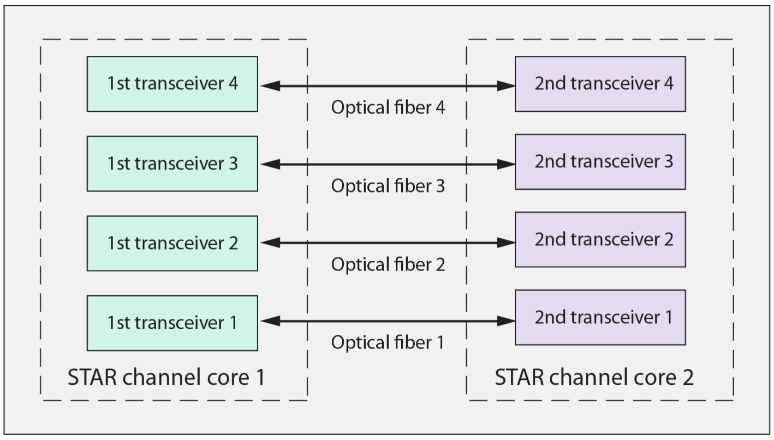

Figure 9. Example of fault resilient STAR optical implementation

Figure 9 shows an implementation consistent with today's 100 Gbit Ethernet opto-transceivers, using four optical fibers and associated transceivers, between two STAR channel cores. The STAR channel cores each interface to this single STAR message channel, which supports four degrees of freedom in automated, local, fault resilient response to an uncorrectable received message fault. In the best of situations, during normal operations, two transceiver pairs can be operated locally at 130-140 GHz for 100 Gbit/sec transfers across two of the optical fibers, delivering a STAR message every nanosecond to collectively deliver 200 Gbits per second across the two fibers as the first degree of freedom in the fault resilience strategy. In other situations, three transceiver pairs are operated locally at 100 GHz, for 66-70 Gbit/sec to collectively transfer 200 Gbits/sec across three optical fibers delivering a STAR message every nanosecond, as shown in Figure 10 as the second degree of freedom. Consider implementations where each STAR message is delivered by one optical fiber and its corresponding circuitry. The local STAR channel core, either receiving a message with an unrecoverable error, or in transmitting that message, can directly determine which fiber, and circuitry, failed. This information can be used by both sending and receiving STAR channel cores to resiliently respond to the error.

TS k stands for the Training Sequence of STAR message k, fork=1:7. In communication protocols, a training sequence refers to a known sequence of channel states which are transmitted to "train" the synchronization circuitry, such as phase locked loops or delay locked loops. The training sequence is often incorporated into each message or burst, to account for timing drift between the transmitter and the receiver. Note that all four optical fibers in Figure 9, can be operated to deliver the STAR message on every local clock cycle as the third degree of freedom in the communication fault resilience strategy. This has been not been shown in detail, but operates much the same as in Figure 10. To summarize, when everything is working optimally in a STAR optical channel, two of the four optical fibers are operated to achieve the collective delivery of a STAR message every (local) nanosecond. When the pairs of fibers and components are not able to operate at 130-140Ghz, a component trio operates at a lower frequency as outlined in Figure 10. When the trios can no longer operate in this fashion, then all four optical fibers and their

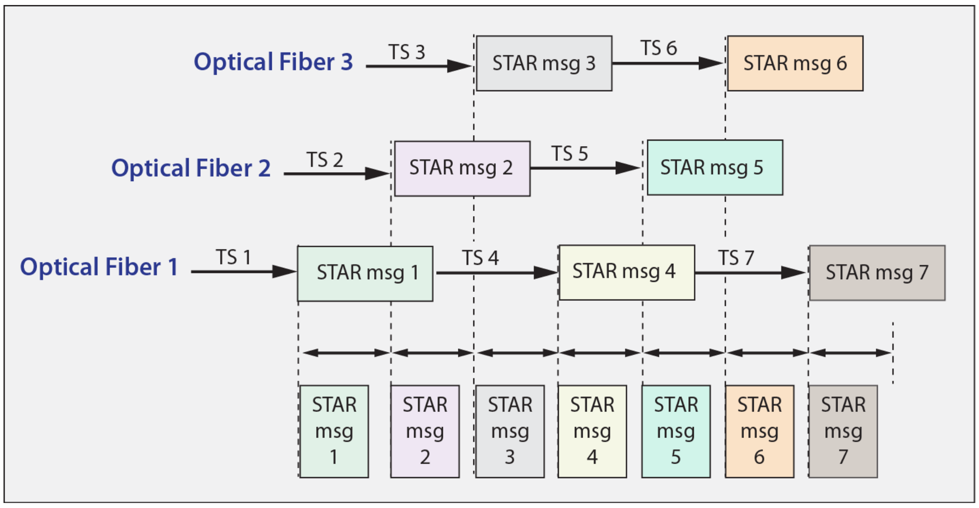

Figure 10. Transmitting and receiving STAR messages using three optical fibers

transceivers are operated at a second, lower frequency to collectively deliver the STAR message every nanosecond.

Figure 11 shows some of the details of the STAR channel cores supporting the interactions within this STAR channel. While there are four optical fibers being operated, each fiber supports communication in each of two channel directions, labeled channel directions one and two. During normal operations at the receiver in STAR channel core 1, ER1 in the upper left hand corner is not asserted, and the incoming selector selects the received message from channel direction one, shown along the left side. At the transmitter in STAR channel core 2 near the bottom, the outgoing selector stimulates the OMP 2. Consider the automated response when operating all four optical fibers fail in one direction. The faulty link, from transmitters to receivers in that one direction is automatically replaced by the spare optical fibers and circuitry in that direction.

Figure 12 shows the interactions in one channel direction, and in the spare channel direction, in greater detail. During normal operation, OMP 2 is stimulated with the package and the context from the outgoing context generator, unless the resend queue has a backlog. When the resend queue has a backlog, the package and context is sent from the queue directly, and any newly generated package and context is added to the existing resend queue. During normal operations the package and context are logged at the resend queue, in case this message fails to be received correctly. Assume the receiver's EDC pipe detects that the received message has an uncorrectable error. This indicates the received message is fatally flawed.

When the EDC pipe reports the uncorrectable error, ER 1 is asserted and a source error message is sent by one or more of the control and status channels from the destination to the source. When all four of the optical fibers for this message channel are already in use, the link treats both source and destination as tainted in this direction. Up until all four optical fibers are in use, the link will attempt repair by using either a different combination of optical fibers, or more optical fibers while operating at a slower local clock, at both the source and the destination. Assuming that all four optical fibers are in use, both the optical fibers, the source and the destination circuitry are replaced by the spare channel resources in the same direction. The operations, when completed, have replaced the left side of Figure 12 with the right hand side. This is the fourth degree of freedom in the communication fault resilience strategy.

Components, such as the DPC, typically have a binary tree interface for the STAR network as shown in Figure 7. Figure 13 shows a different level in a system, where binary trees are shown

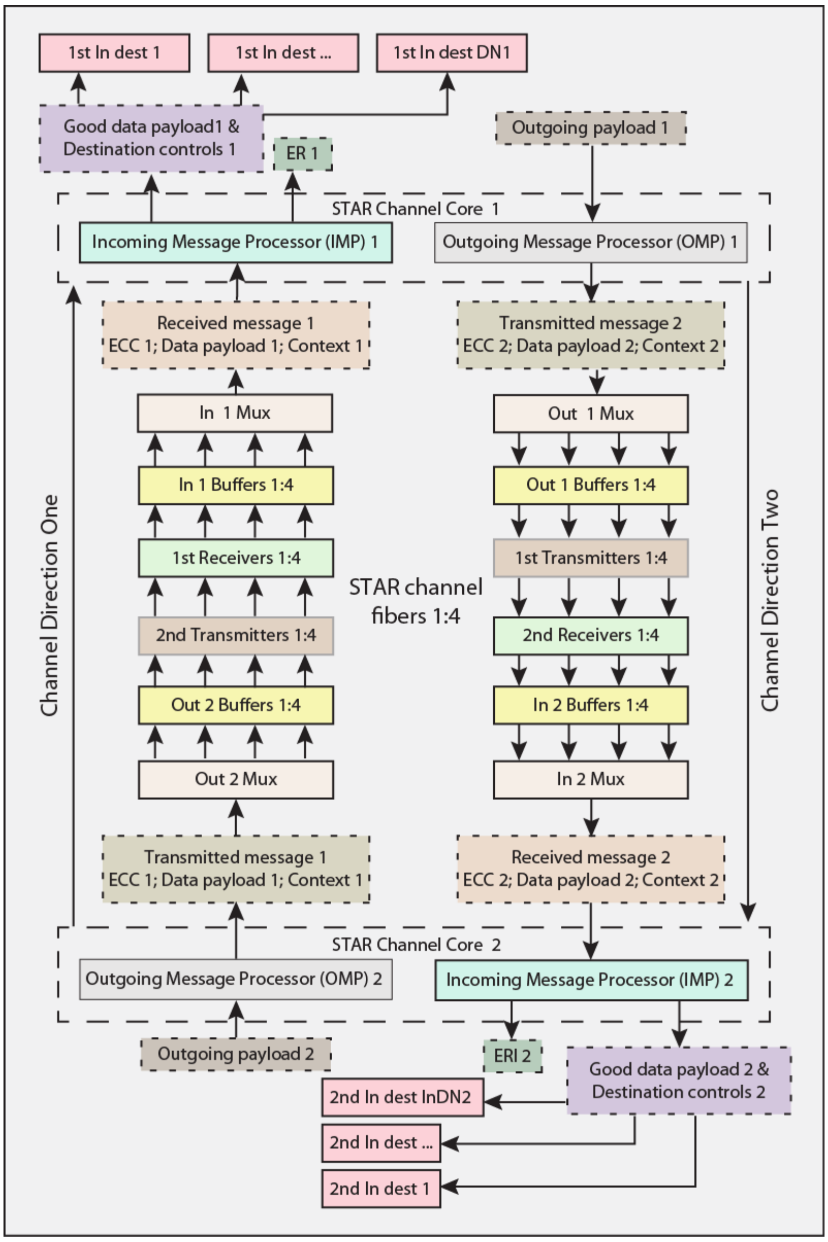
1st In dest 1
1st In dest ...
1st In dest DN1
Good data payload1 & Destination controls 1
ER 1
Outgoing payload 1
STAR Channel Core 1
Incoming Message Processor (IMP) 1
Outgoing Message Processor (OMP) 1
Received message 1 ECC 1; Data payload 1; Context 1
Transmitted message 2 ECC 2; Data payload 2; Context 2
In 1 Mux
Out 1 Mux
In 1 Buffers 1:4
Out 1 Buffers 1:4
1st Receivers 1:4
1st Transmitters 1:4
STAR channel fibers 1:4
2nd Transmitters 1:4
2nd Receivers 1:4
Out 2 Buffers 1:4
In 2 Buffers 1:4
Out 2 Mux
In 2 Mux
Channel Direction One
Channel Direction Two
Transmitted message 1 ECC 1; Data payload 1; Context 1
Received message 2 ECC 2; Data payload 2; Context 2
STAR Channel Core 2
Outgoing Message Processor (OMP) 2
Incoming Message Processor (IMP) 2
Outgoing payload 2
ERI 2
Good data payload 2 & Destination controls 2
2nd In dest InDN2
2nd In dest ...
2nd In dest 1

Figure 11. Interaction between two STAR channel cores across the four optical fibers

on the left as Link 1 and Link 2. If the whole system was composed entirely of these binary tree interfaces, the HPC components would be vulnerable to bottlenecks. Extending the network to a Mostly Binary Tree through the network and operation of STRs to form the PCB interface at Link3 and Link 4, enables optical PCBs etc., to have dual, or better, STAR bundle interfaces, eliminating the bottlenecks. This is another level in the understanding of how to keep an exascale computer fed with data.

Consider the communication between one of these supercomputers and its data center. The management requirement for such a communications interface is that it does not stall the supercomputer, and can keep up with its output. Figure 14 shows using the mostly binary STAR network to allow one cabinet to provide 15 or more STAR bundles to interface with the data center. Using these STAR bundles, each cabinet can use the 16 STAR data channels per bundle, each with 4 Ethernet optical fibers through the STAR bundle to ethernet nodes, as shown. This supports the cabinet simultaneously communicating with 500 or more Ethernet networks (15

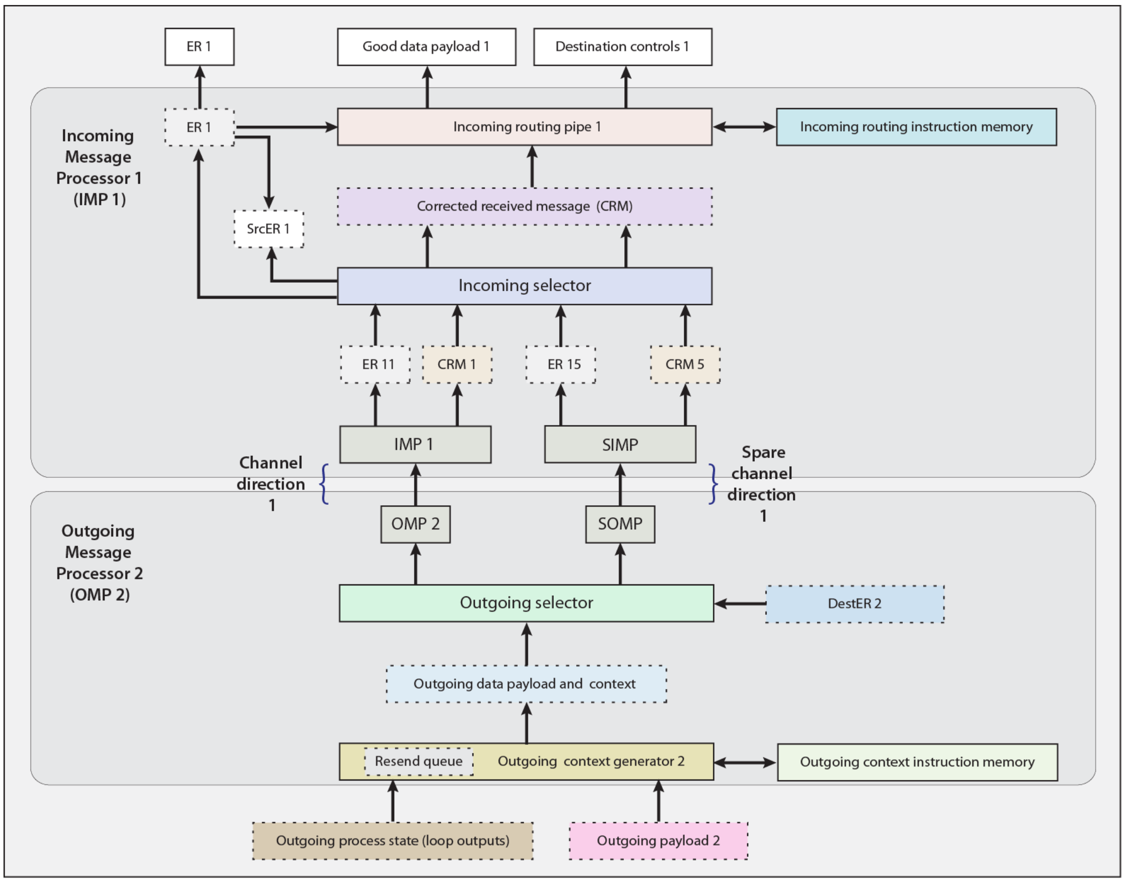
ER 1
Good data payload 1
Destination controls 1
Incoming Message Processor 1 (IMP 1)
ER 1
Incoming routing pipe 1
Incoming routing instruction memory
SrcER 1
Corrected received message (CRM)
Incoming selector
ER 11
CRM 1
ER 15
CRM 5
IMP 1
SIMP
Channel direction 1
Spare channel direction 1
OMP 2
SOMP
Outgoing Message Processor 2 (OMP 2)
Outgoing selector
DestER 2
Outgoing data payload and context
Resend queue
Outgoing context generator 2
Outgoing context instruction memory
Outgoing process state (loop outputs)
Outgoing payload 2

Figure 12. Detailed view of STAR channel core 1 interacting with STAR channel core 2

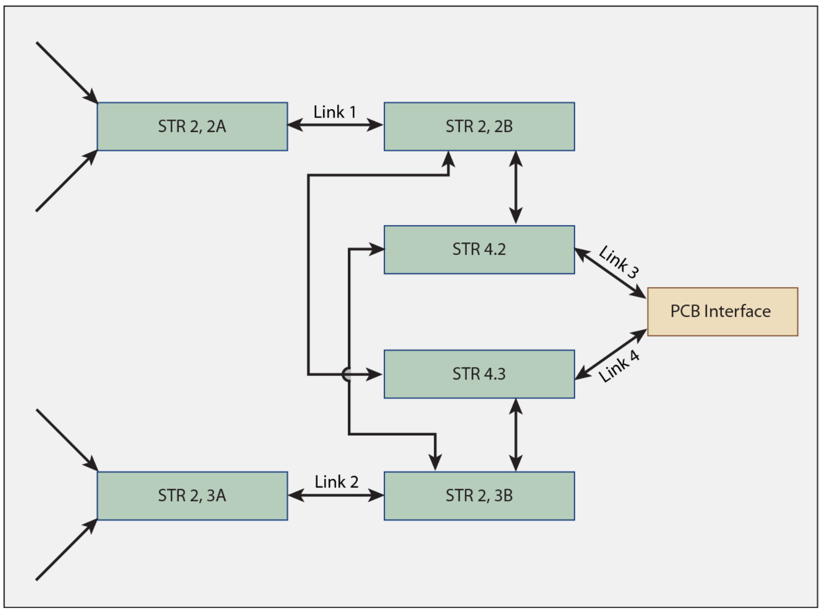

Figure 13. STAR binary tree components extended to make a mostly binary network component

bundles*16 channels/Bundle * 4 fibers/channel= 960), each with 100 Gbit/second bandwidth. This delivery resolves the communication bandwidth requirements to, and from, supercomputers for years to come. Note that this is the third level needed to understand how to keep an exascale computer fed with data.

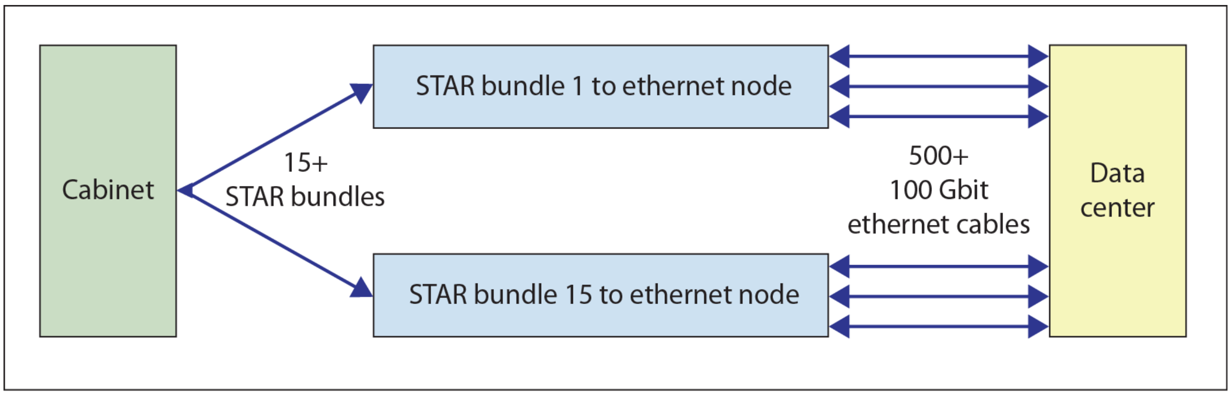
Cabinet
15+
STAR bundles
STAR bundle 1 to ethernet node
STAR bundle 15 to ethernet node
500+
100 Gbit
ethernet cables
Data
center

Figure 14. Cabinet to data center communications with the mostly binary STAR network

3. Transforming Today's Vulnerable Data Center

Today's data centers need to interact with something like the world wide web or the internet, let's call it a merged data and task-instruction external stream shown in the top right corner in Figure 15. The interactions can be regimented so that data communications go through an external data portal and the task-instruction related communications go through an external task portal. Further, one portal's communications cannot be intercepted by the other portal. There are a number of options for achieving this, such as different physical transport layers, differing communications protocols, and distinct encryption mechanisms, any, and all, of which can be implemented.

Second, the data center's internal communications are systematically segregated into one, or more, data related networks and task-instruction related networks. Each of these segregated networks has its own memory controllers. Each of these memory controllers only handles one of the two segregated information types. The data memory controller communicates between the internal data network and the various data related drives, DRAM and SRAMs. Also, the internal data network communicates only across data related channels to invulnerable handheld devices and invulnerable networked sensors [12]. The task-instruction memory controller only communicates between the internal task network and the task related drives, DRAMs, and SRAMs. Also, the internal task network communicates only across task-instruction information channels with the invulnerable handheld sensor and invulnerable handheld devices . Third, the cores, core modules, PEMs, or chips, separately communicate with the task-information messages, and the data-related messages, delivered by these segregated internal networks. These cores, core modules, etc. cannot alter their instruction memory nor their task control based upon data memory comminucation or access. With this approach, the data center of the future can be constructed to systematically exclude the possibility of data memory faults triggering the injection of viruses and root kits.

4. Summary

The STAR message protocol solves, or enables the solution of, each of the following problems:

- Sending an MPI message locks up a buffer until the message has completed being sent.
- Receiving an MPI message locks up the buffer until the message is received and it has either been processed, or cleared from the buffer.
- Short messages can be stalled by long messages at a router transfer point in MPI.

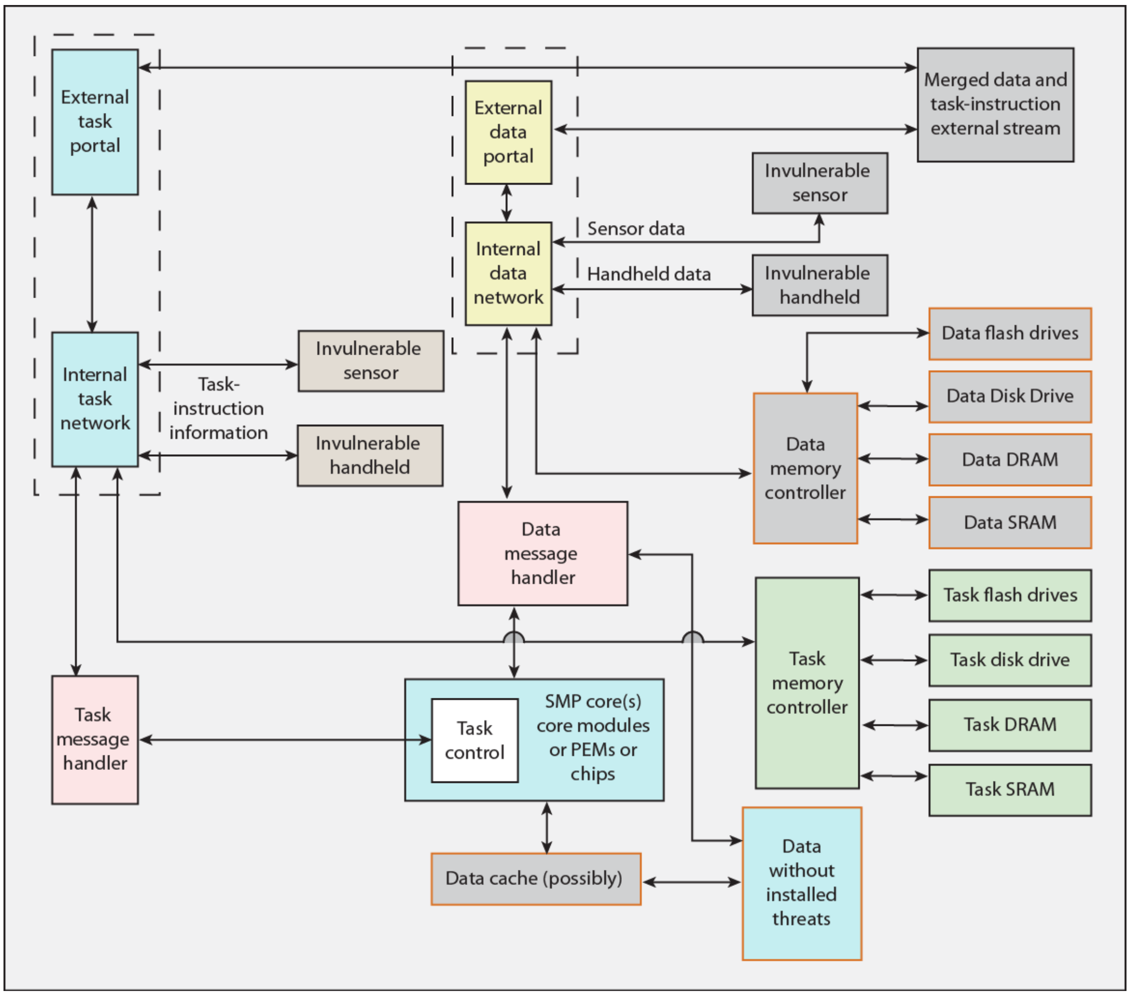

Figure 15. Tomorrow's invulnerable data center

- The common data memory fault security problem, which can inject viruses and rootkits into a computer system.
- The bandwidth problem involved in saving the state of DPCs in exascale systems.
- The data center problem of feeding enough data into, and out of, an exascale system so the exascale system is not stalled.
- The failure rate problem for optical communications in exascale systems.
- The fault resilient response to uncorrectable errors in a received optical message.

References

1. Chisnall, D.; Rothwell, C.; Watson, R. N. M.; Woodruff, J.; Vadera, M.; Moore, S. W.; Roe, M. Davis, B.; Neumann, P.G. (March 2015) "Beyond the PDP-11: Architectural support for a memory-safe C abstract machine", ASPLOS '15, March 14–18, 2015, Istanbul, Turkey, ACM 978-1-4503-2835-7/15/03, DOI: 10.1145/2694344.2694367

2. Dongarra; Report on the Sunway TaihuLight System; (2016), University of Tennessee, Oak Ridge National Laboratory, Dept Electrical Engineering and Computer Science, Tech Report, UT-EECS-16-742, US

3. DOE-ASCAC Subcomittee Report, Top Ten Exascale Researcgh Challenges, (2014 Feb 10), USA

4. Dresler; "United States v Morris", Cases and Materials on Criminal Law. St. Paul, MN: Thomson/West. ISBN 978-0-314-17719-3.

5. "AN-5017 LVDS Fundamentals", (2005) https://www.fairchildsemi.com/application-notes/AN/AN-5017.pdf

6. Golub, van Loan; Matrix Computations (4th ed); (2013) Johns Hopkins University Press, Kindle Edition, Baltimore, Maryland, US

7. Gropp, Huss-Lederman, Lumsdaine, Lusk, Nitzberg, Saphir, Snir; MPI-The Complete Reference vol 2, The MPI extensions, (1998) MIT Press, Cambridge, Mass., US

8. Gropp, Lusk, Skjellum; Using MPI: Portable, Parallel Programming with the Messaging Passing Interface (3rd ed), (2014) MIT Press, Cambridge, Mass. US, DOI: 10.1524/9783486841008

9. Gustafson; Beating Floating Point at its Own Game: Posit Arithmetic; http://supercomputingfrontiers.com/2017/wp-content/uploads/2017/03/2_1100_John-Gustafson.pdf (2017)

10. Heroux, Dongara, Luszcek; HPCG Technical Specification, SAND 203-8752, www.osti.gov/scitech/biblio/1113870f (2013), Sandia National Labs, US

11. HPC Advisory Council, Introduction to High-Speed InfiniBand Interconnect, US

12. Jennings; (Dec 2016) "Simultaneous Multi-Processor Cores for Efficient Embedded Applications", delivered in Barcelona, Spain, and scheduled for publication in the Journal of Computing

13. The Mitre Corporation, (September 2011) "2011 CWE/SANS Top 25 Most Dangerous Software Errors" © 2011 The Mitre Corporation, http://cwe.mitre.org/top25/, downloaded Oct 1, 2016

14. Netlib; "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers", http://www.netlib.org/benchmark/hpl/, hosted by The Innovative Computing Laboratory, University of Tennessee

15. Quinn; Parallel Programming in C with MPI and OpenMP, (2004) McGraw-Hill Companies, Inc. New York, New York, US

16. Saad; Interative Methods for Sparse Linear Systems (2nd ed); DOI: 10.1137/1.978089871800 (2003) SIAM, Philadelphia, PA, US

17. Snir, Otto, Huss-Lederman, Walker, Dongarra; MPI-The Complete Reference vol 1, The MPI Core (2nd ed), (1998) MIT Press, Cambridge, Mass., US

18. Supalov, Semin, Klemm, Danken; Optimizing HPC Applications with Intel(R) Cluster Tools, (2014) Springer's Open Access, DOI: 10.1007/978-1-4302-6497-2

19. Szekeres, L.; Payer, M.; Wei, L. T.; Sekar, R. (May 2014) "Eternal War in Memory", IEEE Security and Privacy, 1540-7993/14 © 2014 IEEE, pp 45-53, DOI: 10.1109/msp.2014.44

20. Texas Instruments, "SCAA082 High-Speed Layout guidelines" `www.ti.com/lit/an/scaa082/scaa082.pdf` (2006)