Performance Analysis of Julia for a set of parallel benchmarks

Kapil Agarwal and Kanu Sahai

What is Julia ?

- High level, high performance dynamic programming language for technical computing
- Facilitates a distributed parallel execution
- Multiple dispatch, dynamic type system, macros and metaprogramming facilities are few of its main features
- Has been benchmarked for serial algorithms against R and Python

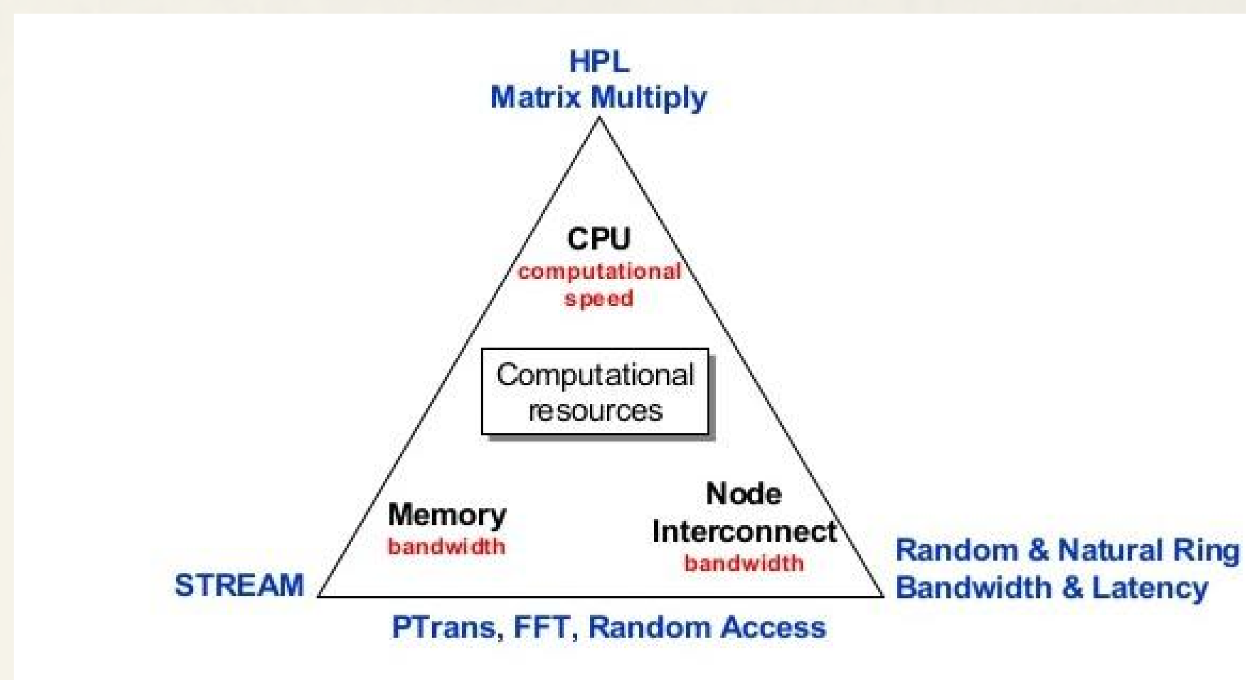
How does it work ?

- Multiprocessor environment - One master process and a team of workers
- Unlike MPI, communication is "one-sided", ie., no "msg_send" or "msg_recv"
- Remote Calls - Request by one process to call a certain function with some arguments on another process
- Remote Ref - Refers to an object stored on any process
- fetch - Explicit data transfer
- @spawnat - Evaluate an expression at specified processor
- Example -
  julia> r = remotecall(2, rand, 2, 2)
  julia> fetch(r)
  julia> s = @spawnat 2 1 .+ fetch(r)
- DArray - Distributed array. 'localpart' and 'localindexes' accessible on a process
- SharedArray - Each process has access to entire array

Limitations

- Calls like MPI_Bcast, MPI_Reduce, MPI_SendRecv don't exist in Julia as all communication is one-sided
- Julia implementation does not take advantage of high-speed low-latency communication hardware such as InfiniBand interconnects in a cluster
- Julia opens a TCP port between every pair of processes that exchange data unlike MPI
- Bug: DArray memory is not fully garbage collected. This issue has been reported to Julia

HPCC Benchmarks

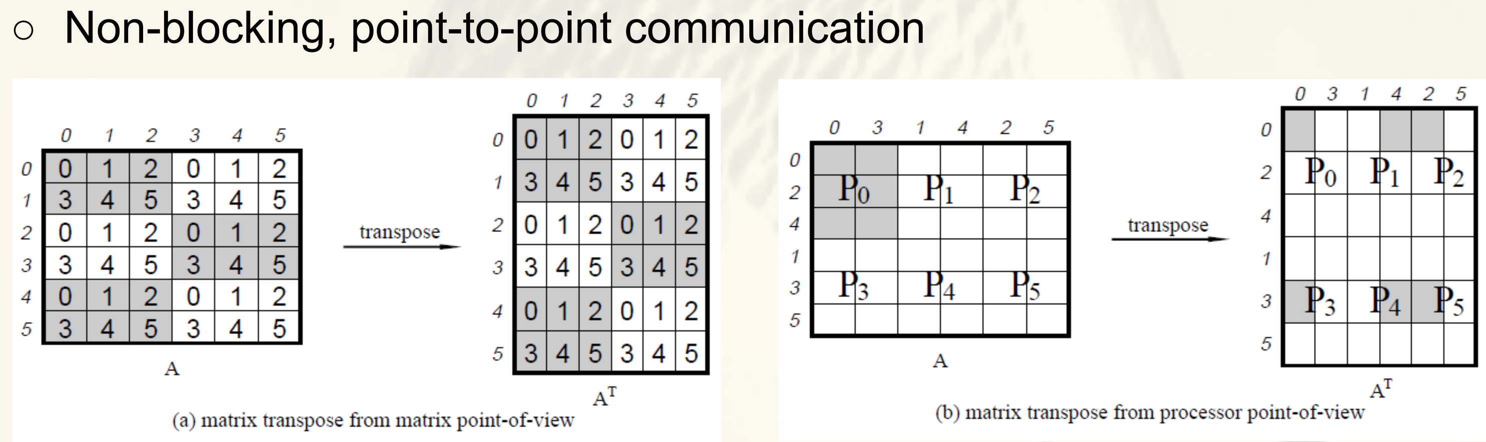


STREAM

- Measures sustainable memory bandwidth in GB/s.
- Measures the computation rate for vector kernels.
- Size of dataset should be greater than the sum of all last level caches available.
- Jinx L3 cache size is 8192 KB. We run STREAM on 16 cores, so we have used arrays containing 64 million elements each.
- Kernels-
  - ``Copy'' measures transfer rates in the absence of arithmetic.
  - ``Scale'' adds a simple arithmetic operation.
  - `Sum'' adds a third operand to allow multiple load/store ports on vector machines to be tested.
  - ``Triad'' allows chained/overlapped/fused multiply/add operations.

PTRANS

- Parallel matrix transpose
- It is a test of the total communications capacity of the network.
- Measures rate of data transfer from multiprocessor memory and excercises communications where pairs of processors exchange large messages simultaneously.
- Ideas:
  - Matrix distributed over P X Q processor template
  - Block scattered data distribution
  - Non-blocking, point-to-point communication



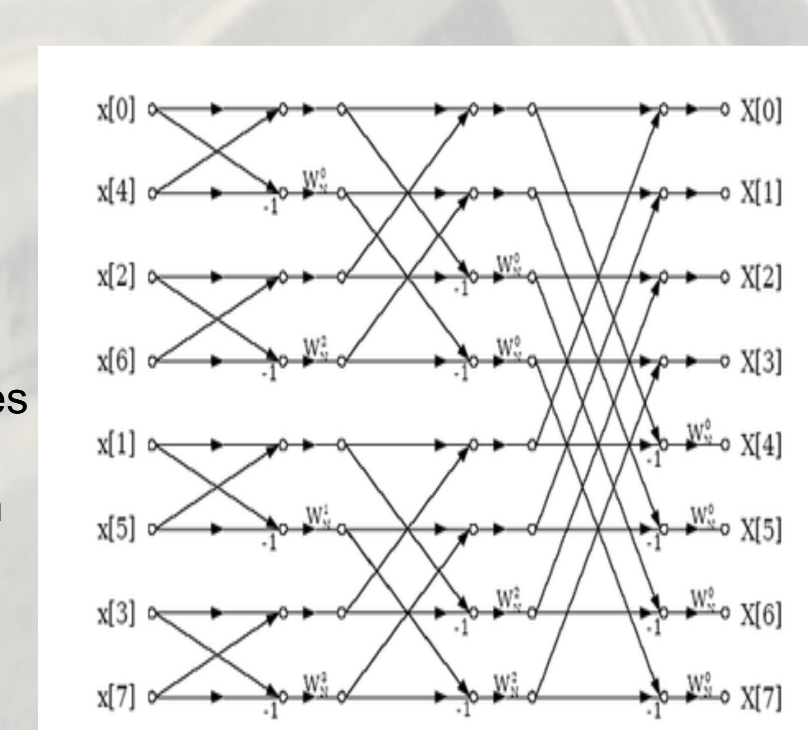
(a) matrix transpose from matrix point-of-view (b) matrix transpose from processor point-of-view

Random Access

- Performance metric - GUPS
- It denotes number of memory locations that can be randomly updated in one second
- It profiles both memory bandwidth and node-interconnect bandwidth
- Implementation -
  - DArray across processors of total size 2^n
  - Each processor generates a stream of random 64-bit ints of total size 2^{n+2}
  - From each 64-bit int, highest n bits determine memory location
  - Number at the location xor-ed with 64-bit int
- Two additional variables -
  - Acceptable error
  - Bucket of updates

Parallel FFT

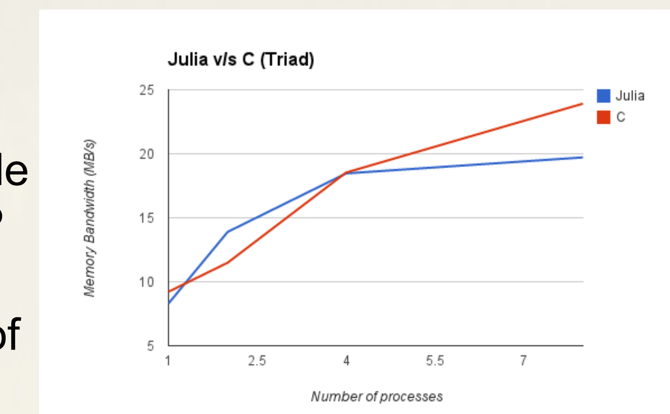
- Performance metric - MFLOPS
- It profiles both memory bandwidth and node-interconnect bandwidth
- Implementation -
  - Bit-reversed sequence of input array distributed across processes (DArray)
  - Each processor performs FFT on local chunk
  - log P rounds of inter-node communication as per Cooley-Tukey algorithm

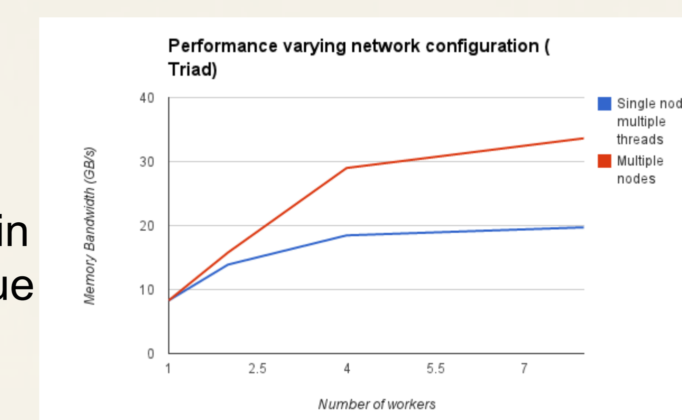


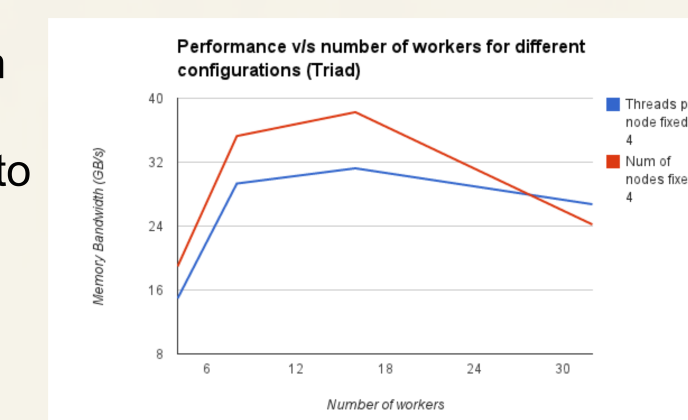
Results and Analysis

STREAM

- Performance of Julia comparable with C/OpenMP
- Sharing of L3 cache in case of single node
- Separate L3 cache in multi-node, less contention
- Initial increase in performance due to efficient resource utilization
- Less work than the number of workers leads to more memory latency

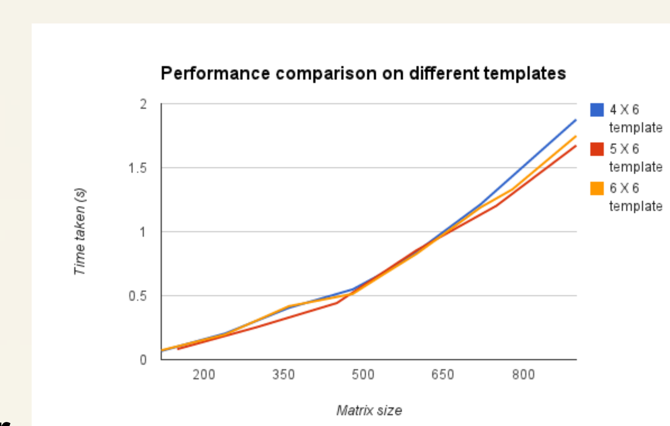






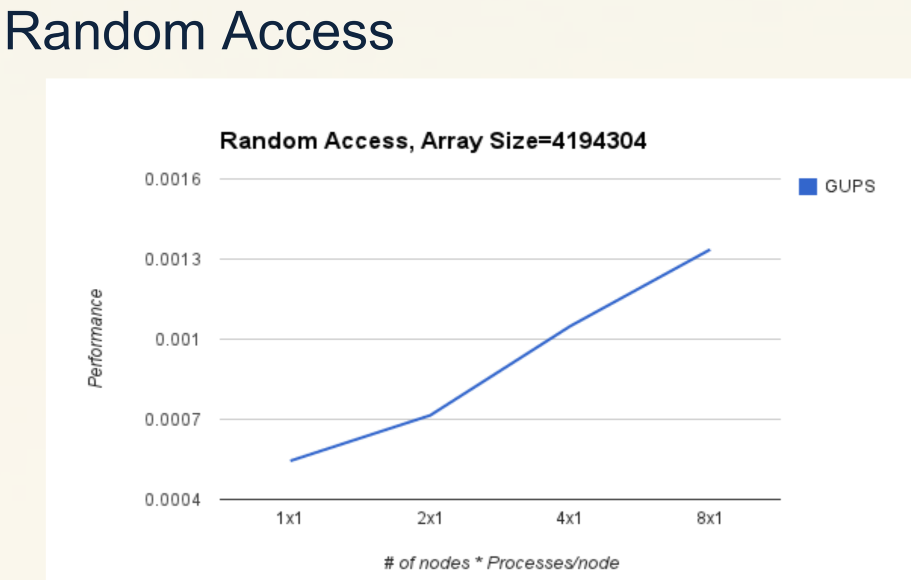
PTRANS

- Time to transpose increases as matrix size increases
- Similar performance for different P
- Time to transpose decreases as Q decreases



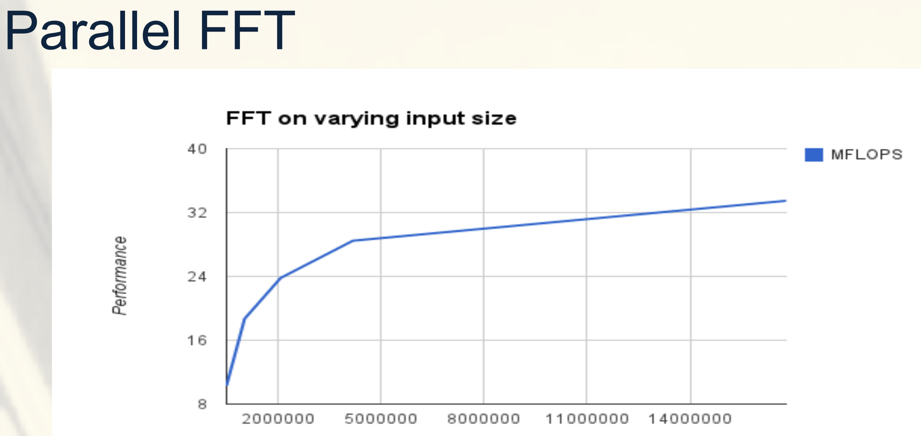
72 workers		64 workers		48 workers	
6 X 16	0.94 s	4 X 16	0.79 s	4 X 12	0.69 s
8 X 12	0.79 s	8 X 8	0.74 s	6 X 8	0.60 s
12 X 8	0.82 s			8 X 6	0.57 s
				12 X 4	0.59 s

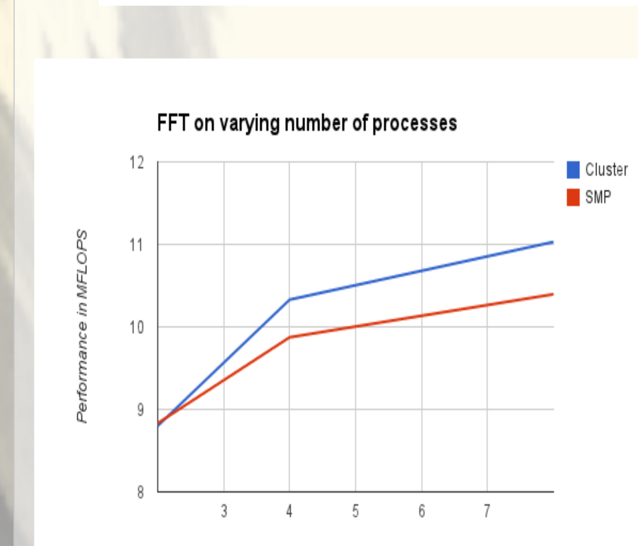
Random Access



- As nodes increase, GUPS increase
- Even though 1% error is acceptable, Julia gives 0% failure rate everytime

Parallel FFT





- Performance increases rapidly initially then slows down for both graphs
- Heavy local computation and excessive communication are bottlenecks respectively