Multi-Threaded Design and Implementation of Parallel Pipelined STAP on Parallel Computers with SMP Nodes

Alok Choudhary

Wei-keng Liao,
Pramod Varshney, and
Donald Weiner

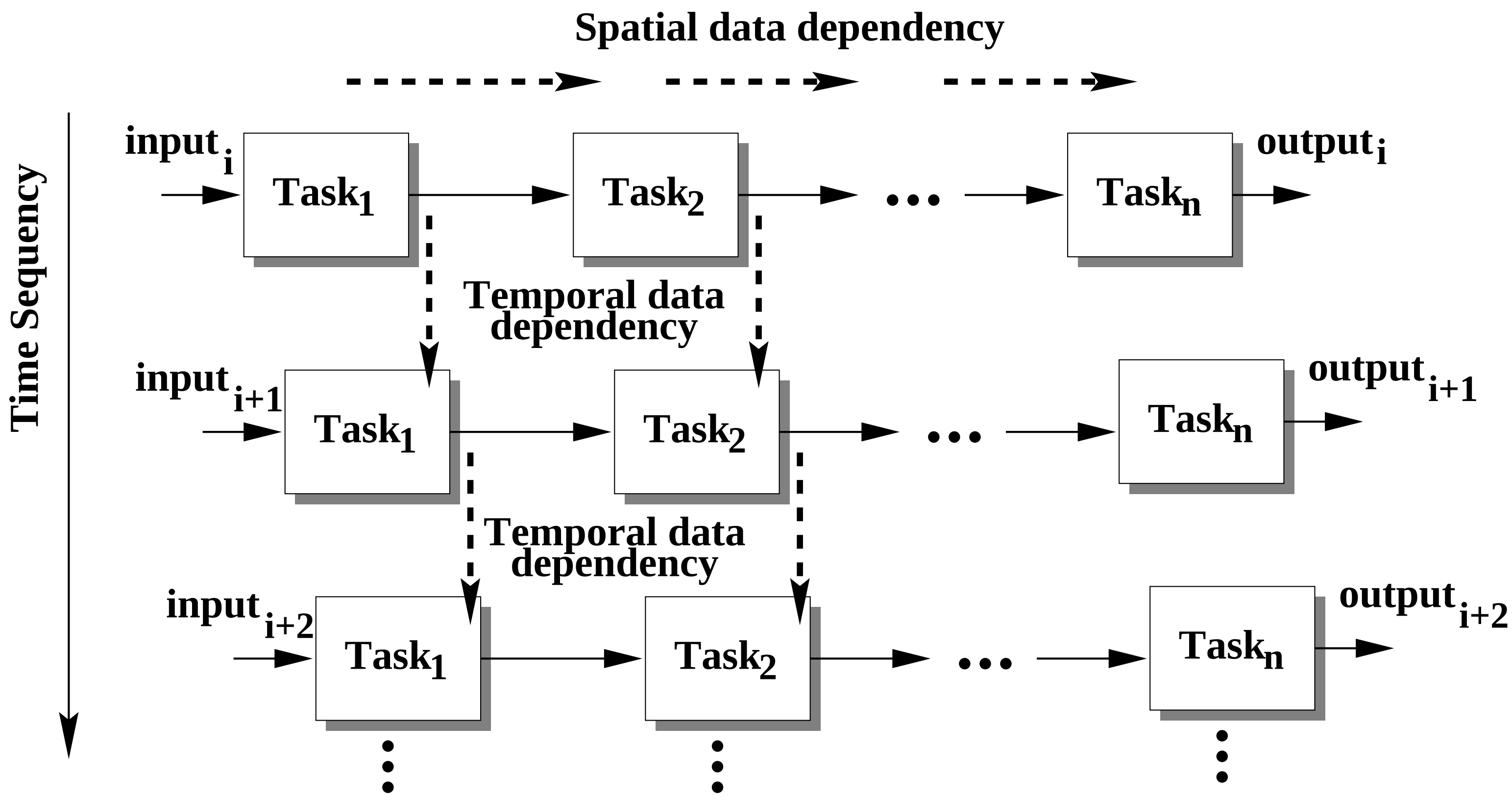
ECE Department
Northwestern University

EECS Department
Syracuse University

Outline

- Parallel Pipeline Computation Model
  - Data Dependencies
  - Throughput and Latency
  - Data Redistribution
  - Processor Assignment
- Design and Implementation
  - Multi-threads on Intel Paragon
- Performance Results
  - Overall System Performance Evaluation
  - Tradeoff Between Throughput and Latency

Parallel Pipeline Computation Model



Throughput and Latency

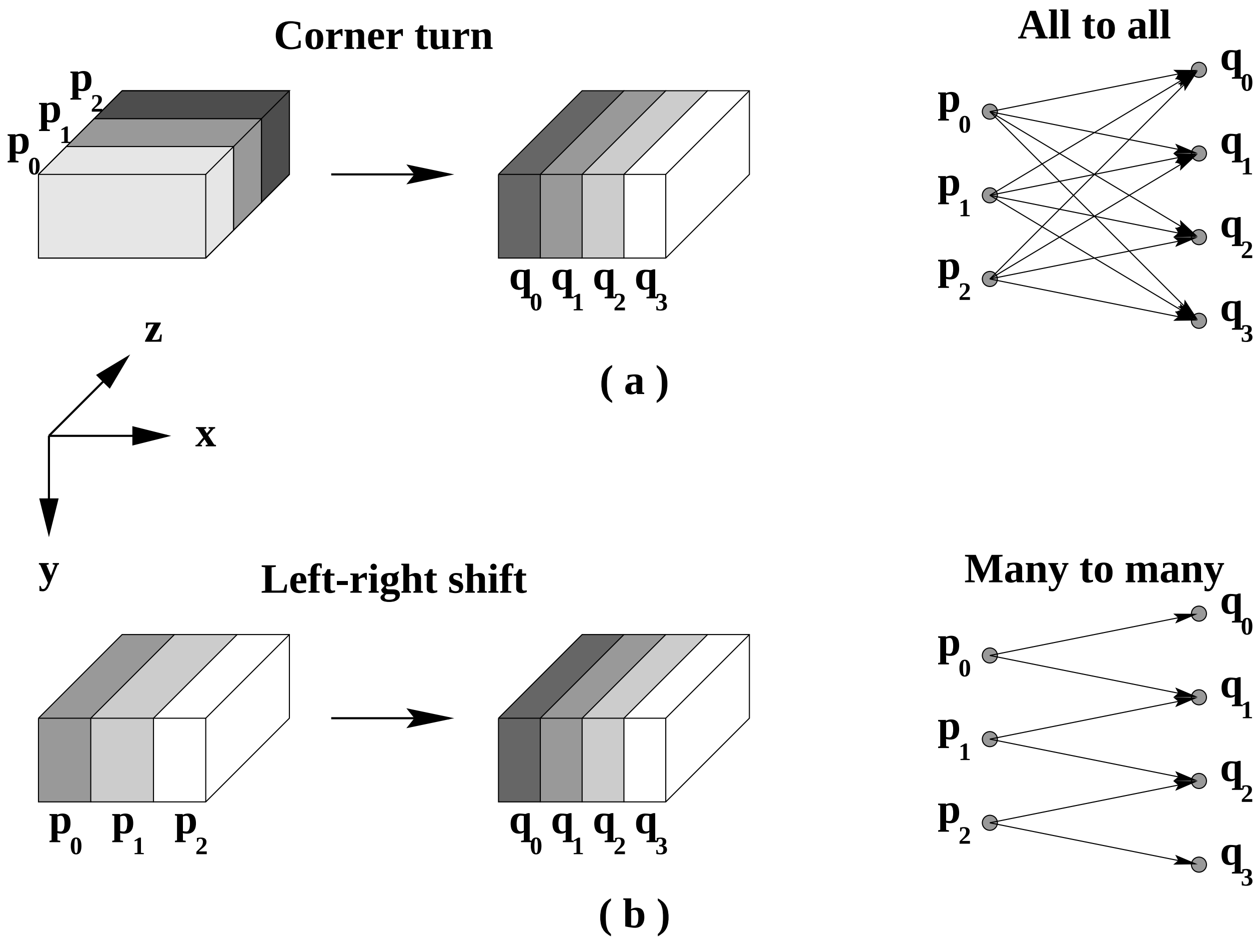
- **Throughput depends on the task with maximal execution time**

$$throughput = \frac{1}{\max_{0 \leq i \leq n} T_i}$$

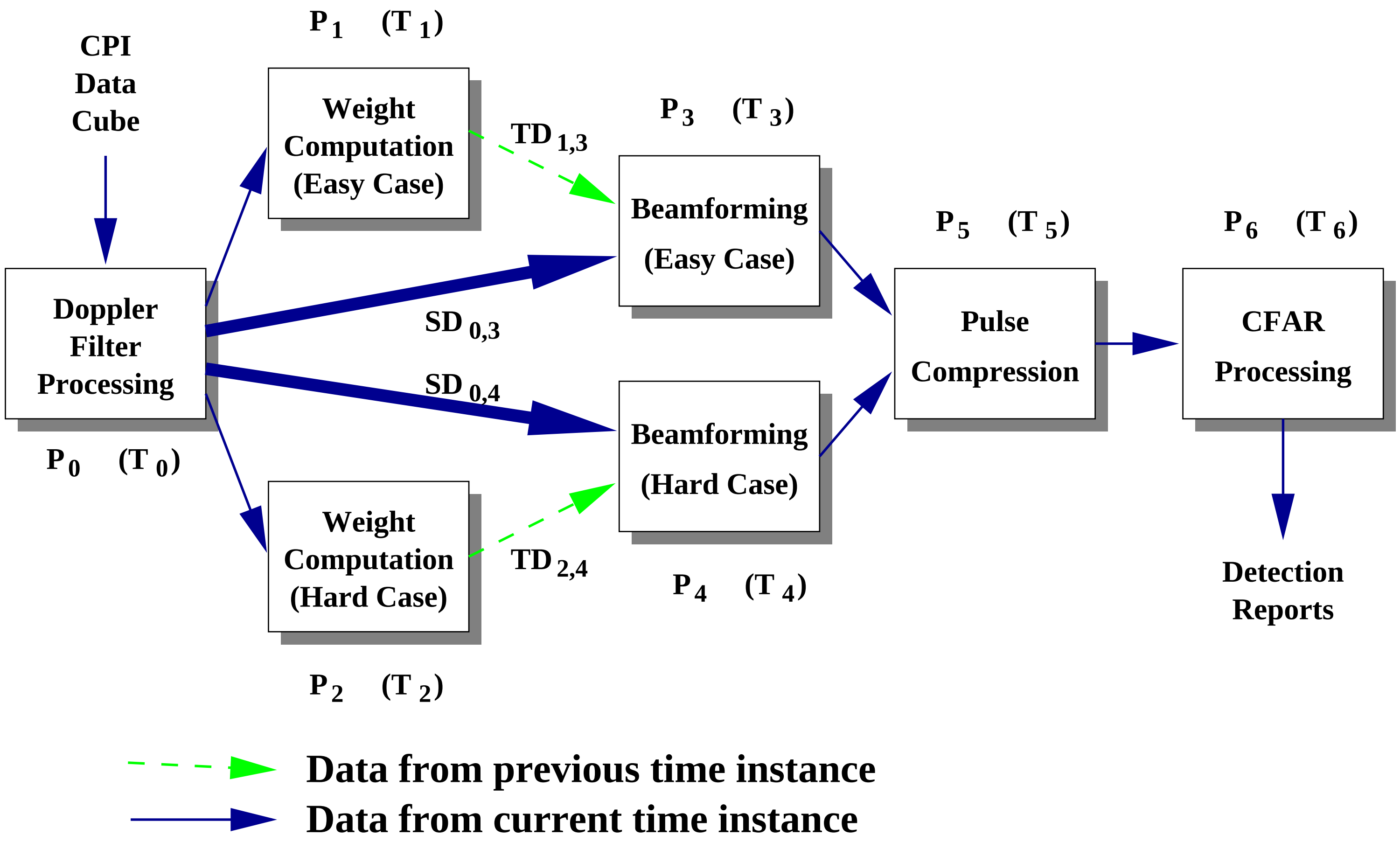
- **Latency does not include the temporal data dependency**

$$latency = \sum_{i=0}^{n} T_i - \sum_{i \in TTD} T_i$$

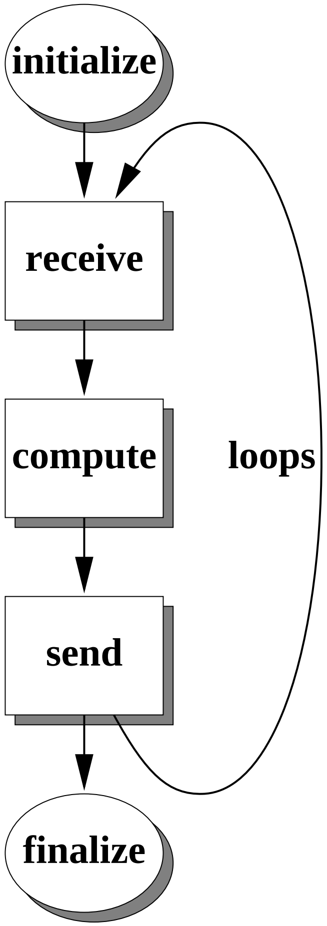
Data Redistribution



Implementation and Data Flow



Execution Flow on One Compute Node

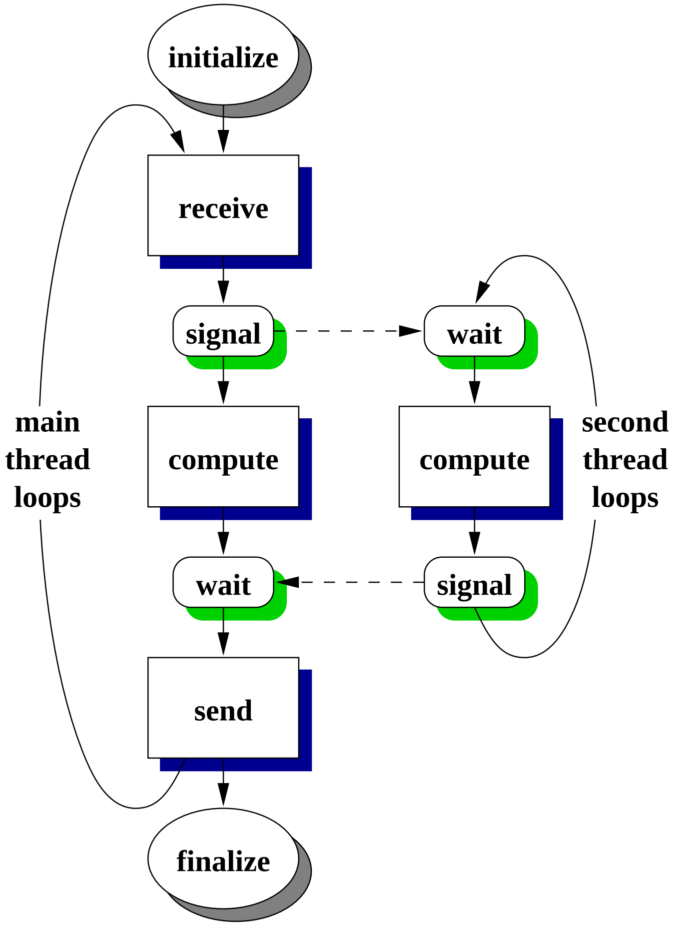


- Execution flow consists of: receive, compute, and send phases
- Communication between two groups of compute nodes
- Message packing/unpacking in send/receive phase
- Double buffering is used for asynchronous send and receive
- Symmetrical Multiple Processor (SMP) node can improve performance

Multi-threading on Paragon SMP System

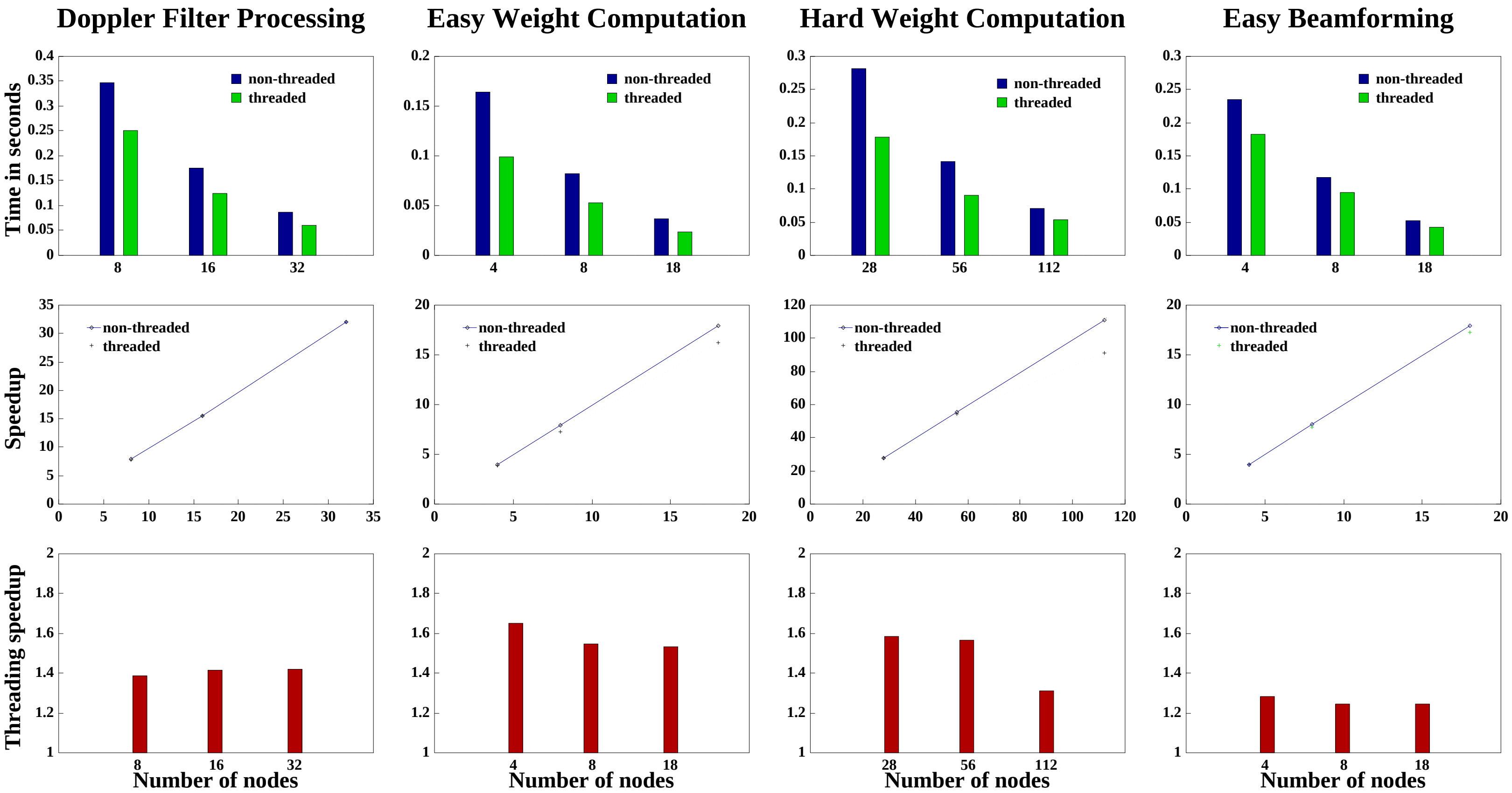
- AFRL Paragon has 232 SMP nodes
- Each SMP node has 3 processors
- Processors have private cache but share main memory
- Multi-threaded environment is supported using POSIX threads
- Each SMP node is configured as 2 application processors and 1 message co-processor by running UNIX OSF/1
- Communicate with other nodes through message co-processor
- OSF/1 allows users to access two application processors only
- The message co-processor is dedicated to message passing

Multi-thread Design and Implementation

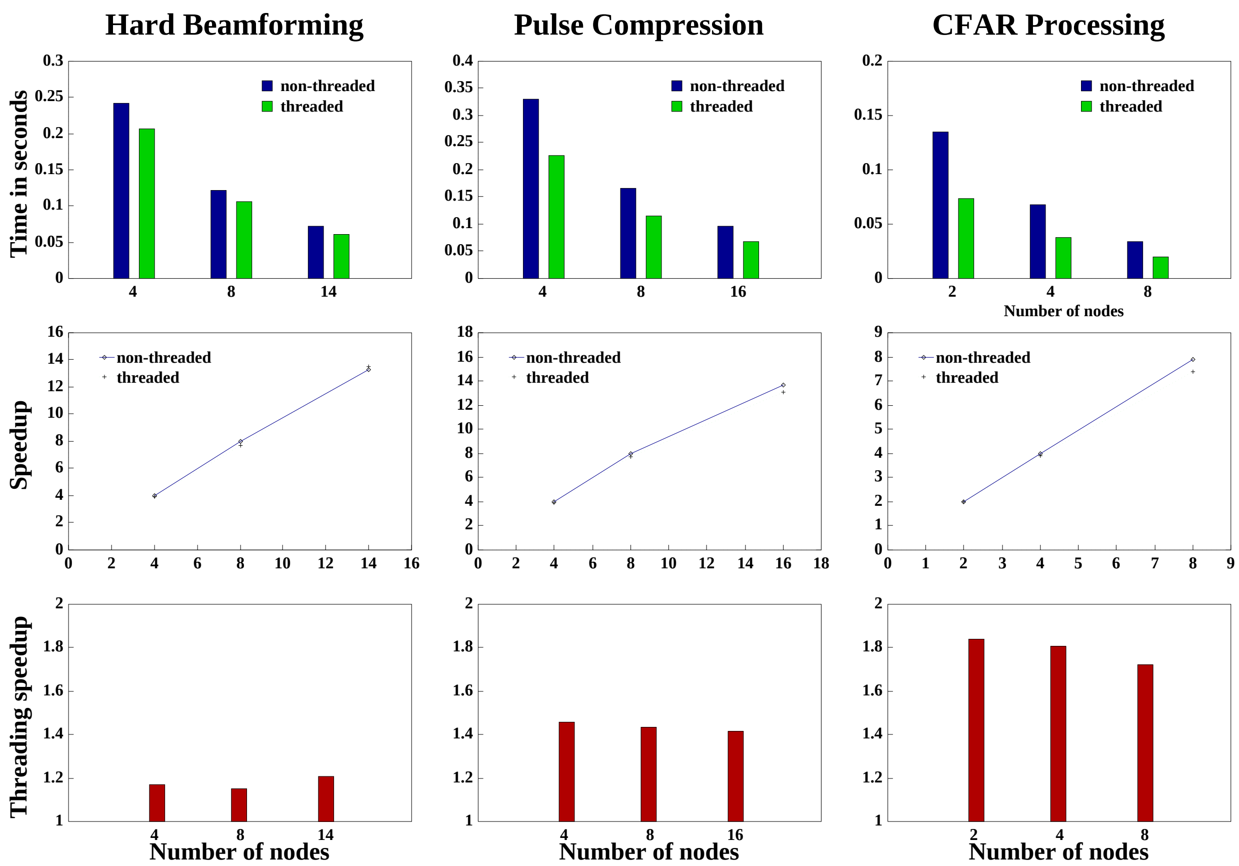


- Two POSIX threads implemented in the compute phase
- Signal operations use a mutually exclusive access semaphore
- Only main thread performs communication
- Second thread is signaled by main thread when input is received
- No multi-threads in send and receive phases because Paragon communication library (NX) is not thread-safe

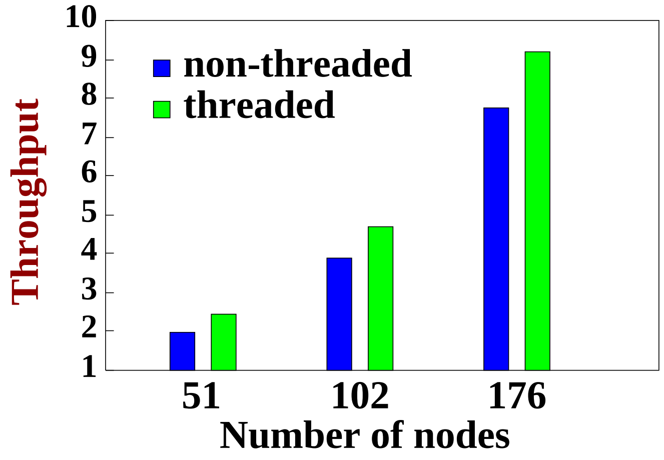
Multi-threading in Compute Phase (I)

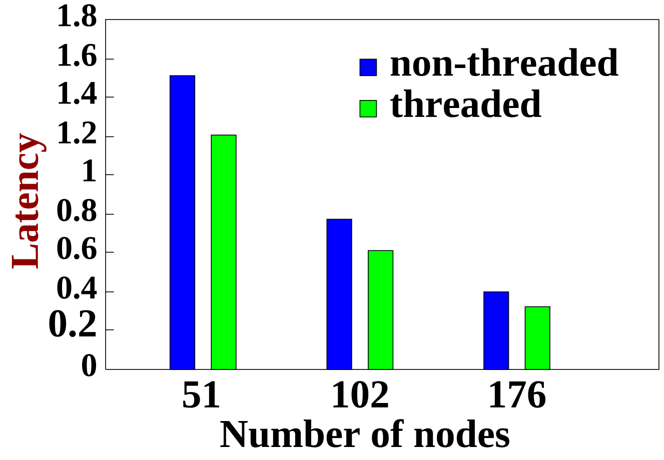


Multi-threading in Compute Phase (II)



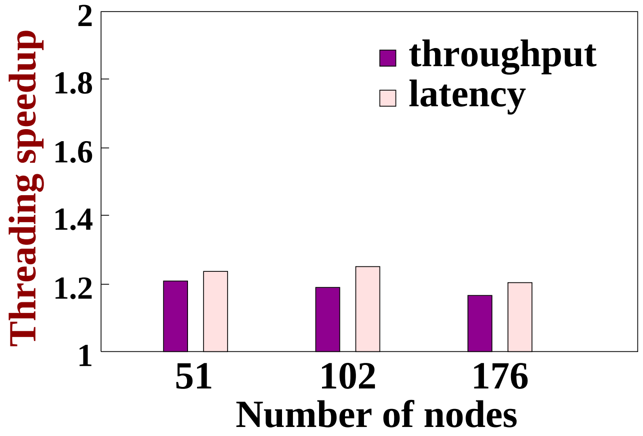
Multi-threading Performance Results





- Further improvement with multi-threading implementation
- Good scalability and speedups on throughputs and latencies
- Throughput of 9.19 CPIs per second using 176 nodes with two threads
- The corresponding latency is 325 ms per CPI

Threading Speedup



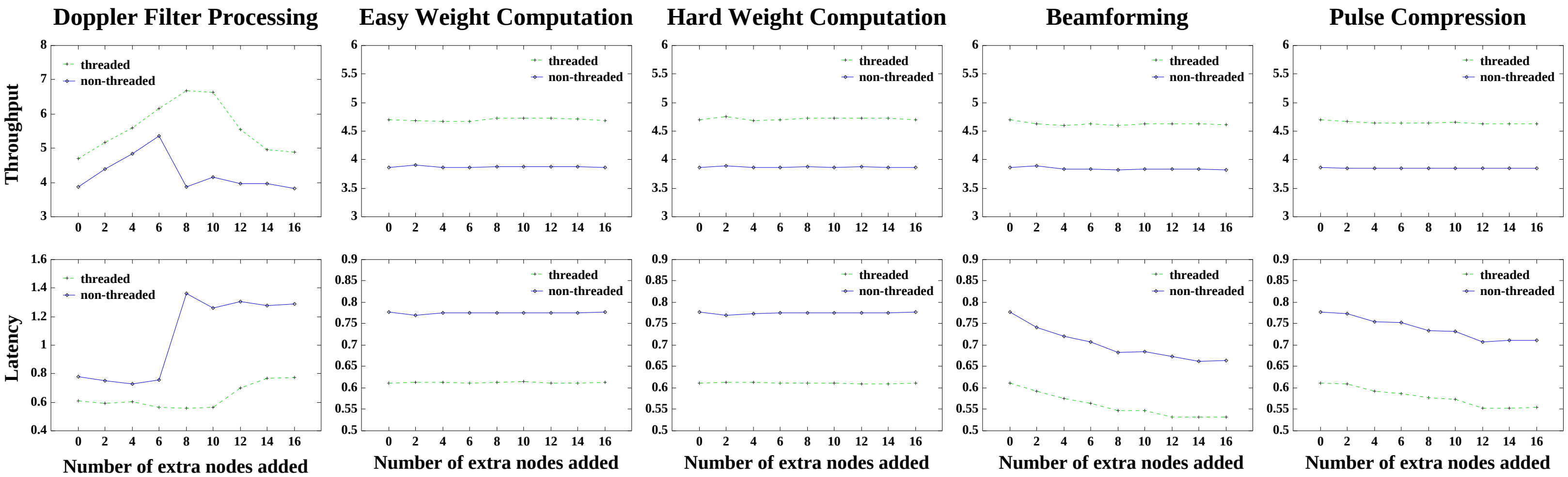
- The threading speedup is defined as

$$S = \frac{T_{non-threaded}}{T_{threaded}}.$$

- Thread scheduling is handled by OS kernel

- Users cannot control or get information about which processor runs which thread
- The linked thread-safe libraries most likely contains overheads of concurrent read/write operations
- Each thread executes independently but it shares resources with other threads (eg. memory)

Increase Nodes Assigned to Each Tasks



- Using the case with 102 nodes, add 2 nodes at a time to each task
- Extra nodes added to Doppler filter processing task, the throughput and latency improved
- Throughput improved when adding nodes to other tasks, because Doppler filter processing task is the bottleneck
- Latency reduced when adding nodes to last two tasks
- The performance is consistent with the equations

Adding 8 Nodes to the Case with 102 Nodes

	non-threaded			threaded		
# of nodes	102	110	impr %	102	110	impr %
throughput	3.87	4.83	25.1	4.69	5.61	19.7
latency	0.78	0.66	14.4	0.61	0.55	10.6

- Add 4 nodes to Doppler filter processing task and 4 to pulse compression task
- Total number of nodes increased by 7.8%
- According the throughput equation, nodes can be assigned to the task with maximum execution time to increase the throughput
- From the latency equation, nodes can be added to those tasks which benefit the most to reduce the latency

Conclusion

- Parallel pipeline computation model on commercial HPC systems
- Modular, scalable, and portable software design
- Real full application parallelized and implemented from AFRL
- Multi-threading implementation on AFRL Paragon
- Scalable performance on both throughput and latency
- Further optimizations
  - Inter-task data redistribution
  - Processor assignments