Grappa: a scalable platform for irregular* data analytics in the cloud

Jacob Nelson, Brandon Myers, Brandon Holt, Preston Briggs, Luis Ceze, Mark Oskin
University of Washington

Simon Kahan
Pacific Northwest National Laboratory

http://sampa.cs.washington.edu/grappa

*__Irregular data analytics applications__ such as graph analysis __do not fit bulk-synchronous execution models like Map-Reduce__ because they have highly unpredictable data access patterns.

Mass-market computer systems' performance relies on predictable behavior. Therefore achieving good performance at scale on irregular applications is challenging.

Scaling irregular applications to support ever-growing amounts of data matters! Think social networks analysis, ad placement, recommendation systems, fraud detection, data-driven science, ...

Grappa is a platform for irregular and highly parallel data analytics applications. It is designed to run on mass-market systems such as clusters of commodity computers and public clouds.

At its heart are two components: a latency-tolerant execution engine and an intelligent network layer. Together, they mitigate the problems of irregular application behavior by

- exploiting fine-grained task parallelism to tolerate the increasing communication latency due to irregular data accesses
- aggregating remote references from disparate tasks to make better use of network bandwidth at scale.

The application developer need only express parallelism, not decide when and how to exploit it.

Tolerate global memory latency

Memory operations that reference remote memory are turned into active messages directed to a *delegate* core on the remote node. After issuing the message, the requester yields the processor until the reply arrives. The delegate core performs the operation. Read-modify-write cycles are performed entirely on the delegate.

Tolerate local memory latency

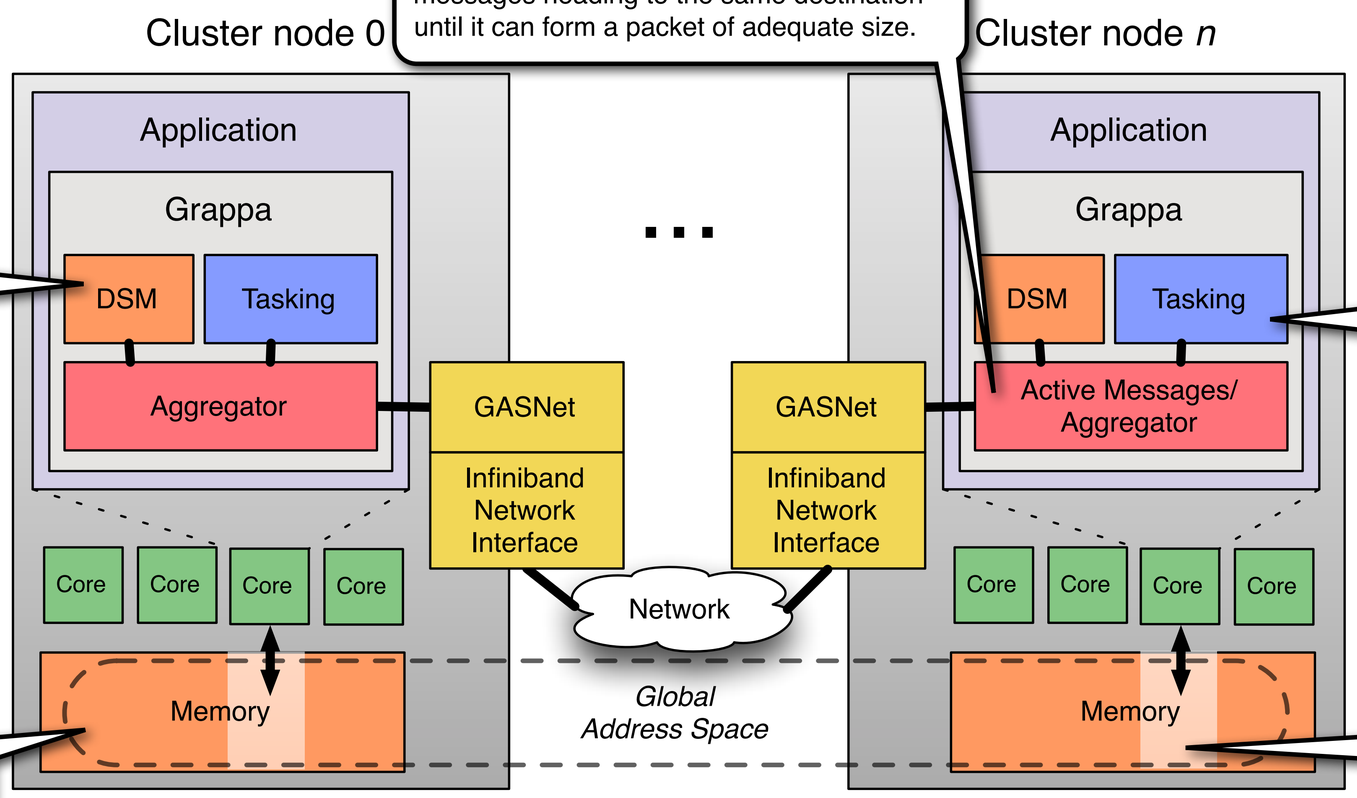
By issuing software prefetches and yielding on likely cache misses, Grappa exposes more memory parallelism to the processor and increases node-local random access bandwidth.

Exploit locality when available

When there is locality to be exploited, global memory can be accessed through a software caching layer with user-controllable granularity.

Mitigate low network injection rates

Mass-market networks require large packets to fully utilize their bandwidth. Grappa delays messages heading to the same destination until it can form a packet of adequate size.



Global-view task parallelism

Execution starts with a single `main()` function. The programmer expresses parallelism with explicit task spawns or with `parallel_for()`. The runtime chooses where and when to run tasks, but in order to exploit locality the programmer can optionally constrain a task to a particular core.

Lightweight context switching

Grappa executes tasks cooperatively with compiler-assisted quick context switching. Thousands of concurrent tasks are multiplexed onto each core.

Dynamic load balancing

To keep the system well-utilized, under-utilized cores perform work-stealing.

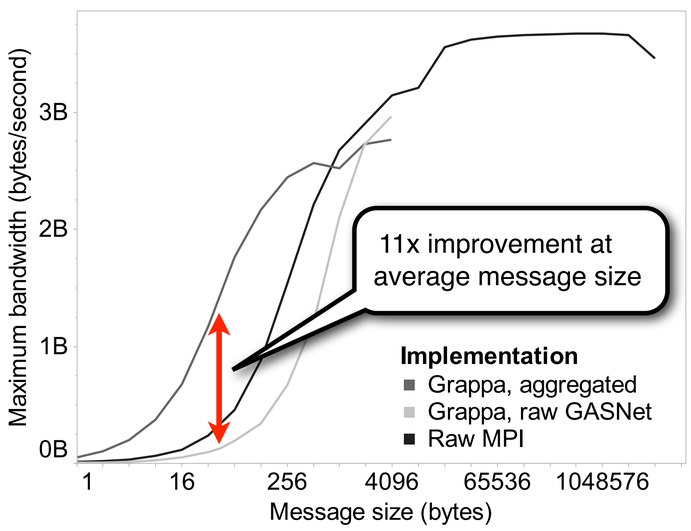
Global shared memory

Grappa's shared heap is constructed out of chunks allocated on each node in the system. Contiguous addresses are spread across the chunks in a block-cyclic fashion.

High-throughput fine-grained synchronization

Each synchronization variable is matched with a *delegate core*. All accesses to that variable are performed serially on that core using active messages. This allows Grappa to provide atomic semantics without locks or atomic operations.

Message aggregation



Benchmarks

Hardware:

- Grappa: PNNL's AMD Interlagos cluster, 2.1-GHz, 64GB/node, Mellanox 40Gb/s Infiniband
- XMT: PNNL's 128-processsor Cray XMT-1