# Grappa: Bridging the gap between mass-market clusters and irregular applications

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

Mass-market computer systems are designed to exploit spatial locality via cache and local memory to achieve high efficiency. Unfortunately, in irregular applications like graph analysis, spatial locality is often difficult—if not impossible—to express.

Graph analytics applications from domains such as business, national security, machine learning, data-driven science, and social network computing are increasingly important. These graphs are large; fast response requires multi-node systems. Partitioning these graphs so that communication between cluster nodes is minimized is challenging. As system size grows, this communication becomes the bottleneck and performance degrades.

Grappa is a latency-tolerant runtime for mass-market clusters that mitigates this degradation, allowing graph processing to scale up even in the presence of diminishing locality and increasing latency. Grappa works by:

- exploiting fine-grained task parallelism to tolerate the increasing latency
- aggregating remote references from disparate tasks to make better use of diminishing 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

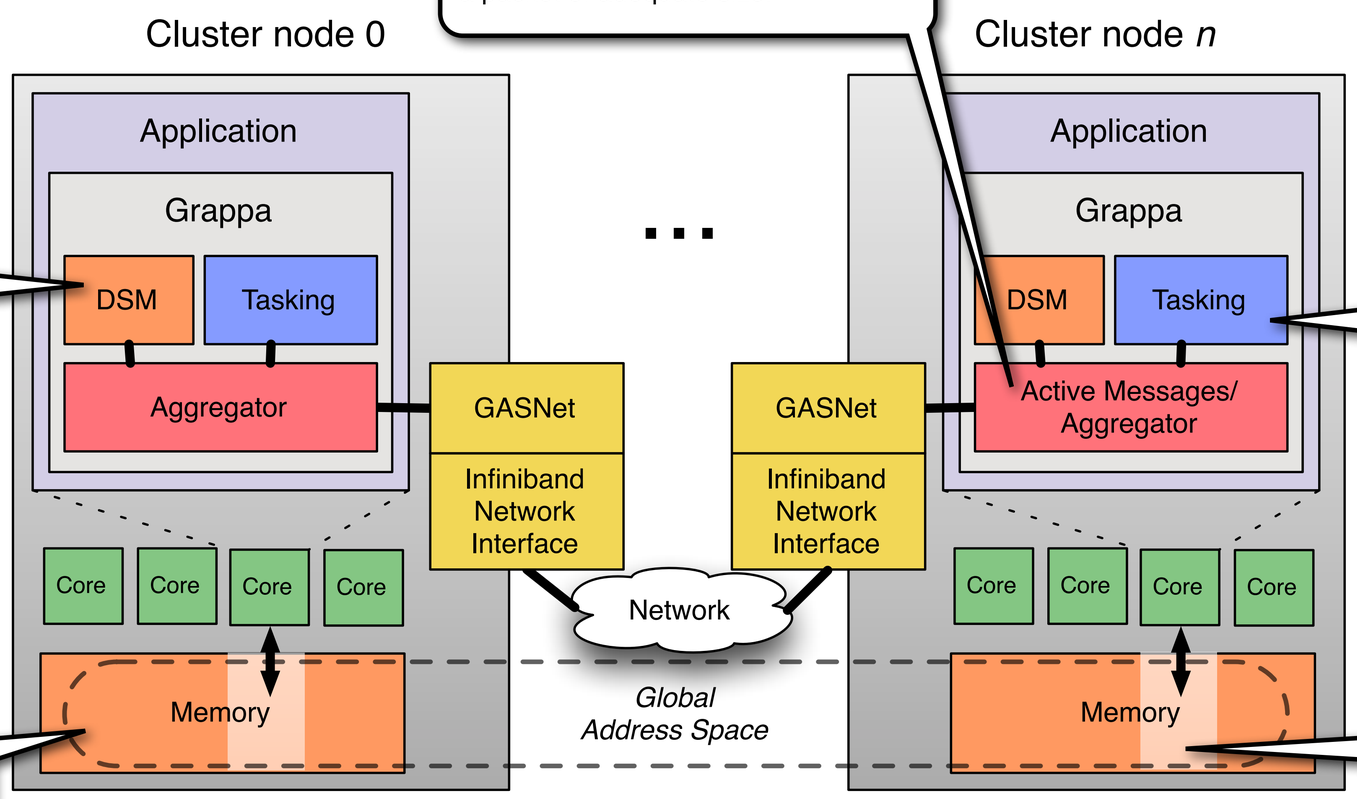
By issuing software prefetches and yielding on likely cache misses, Grappa exposes more memory parallelism to the processor and increases node-local random access bandwidth.

## 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.

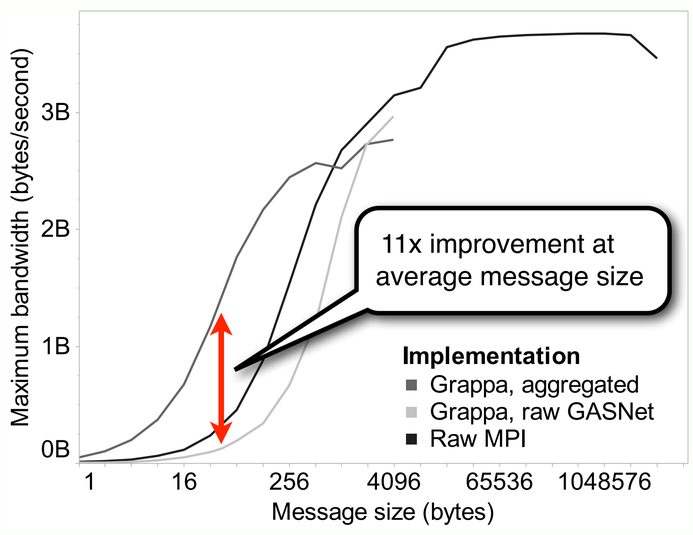
## Global shared memory

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