The VirtualCL (VCL) Cluster Platform

Amnon Barak and Amnon Shiloh
Computer Science, Hebrew University

http://www.MOSIX.org/txt_vcl.html

Background

Most applications that utilize OpenCL™ devices (CPUs, GPUs, Phi, accelerators), run their kernels only on local devices, in the same computer where the applications run.

The VCL cluster platform is a wrapper for OpenCL that allows most unmodified applications to transparently utilize many OpenCL devices in a cluster as if all the devices are on the local computer.

Main features:

- Supports OpenCL device from all vendors.
- Provides a shared pool of devices to users of several hosting nodes.
  - **There is no need to coordinate which devices are allocated to each user.**
  - **Applications can even be started from workstations without OpenCL devices.**

Motivation: ease the development and running of parallel applications.

Targeted for:

- Parallel applications that can utilize many devices concurrently.
- Many users that share a pool of devices, e.g. in a cloud.

The VCL Runtime Model

VCL is designed to run applications that combine a CPU process with parallel computations on many OpenCL devices.

The CPU process runs on a single “hosting” node.

- Responsible for the administration and overall program flow.
  - May perform some computation.
  - Can be multi-threaded, to utilize locally available cores.

OpenCL Kernels run on cluster-wide devices:

- Location of devices is transparent to the process.

The VCL Programming Paradigm

Combines the benefits of the OpenMP and MPI approaches.

The CPU programmer benefits from reduced programming complexity of a single computer - availability of shared-memory, multi-threads and lower level parallelism **(as in OpenMP).**

Kernels: independent programs that can run on cluster-wide devices (like in MPI).

Outcome:

- VCL benefits applications that utilize many devices concurrently.
- The VCL model is particularly suitable for applications that can make use of shared-memory on many-core computers.

The VCL Components

VCL consists of 3 components:

- **The VCL library.**
- **The broker routing and arbitration daemon.**
- **The backend server daemon.**

The VCL Library

The VCL library allows most unmodified OpenCL applications to transparently utilize any number of OpenCL devices.

- Manages the data-base of OpenCL objects.
- Can work with any OpenCL device.

Since network latency is the main limiting factor when communicating with remote devices:

– The VCL library optimizes the network traffic by minimizing the number of round trips required to perform OpenCL operations.

- Multiple buffers are sent together.
- Kernels are sent together with their parameters.
- Queues and events are handled on the hosting node.

The Broker

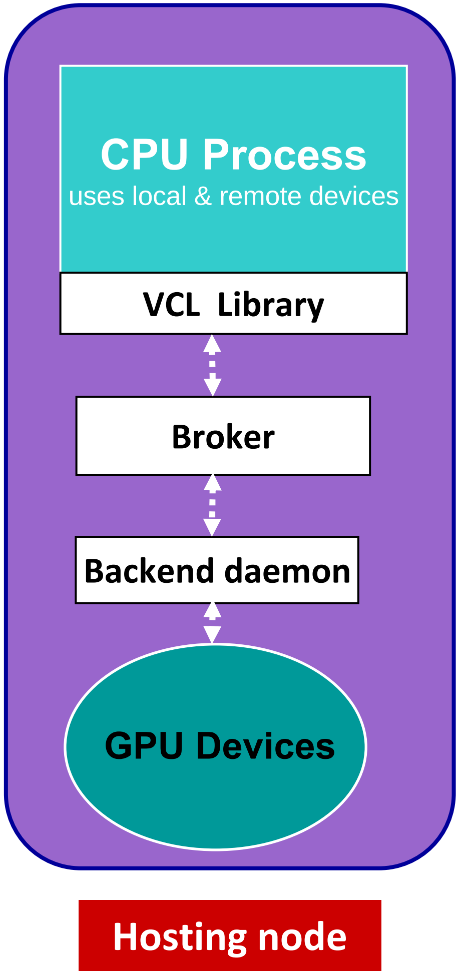
Routing and Arbitration Daemon:

- Collects current information about available devices in the cluster.
- Matches requests for devices by the VCL library with available cluster devices.
- Responsible for authentication and access permissions.
- Routes messages between the VCL library and the backend server daemon.
  - Separates applications from the network layers, to prevent blocking.

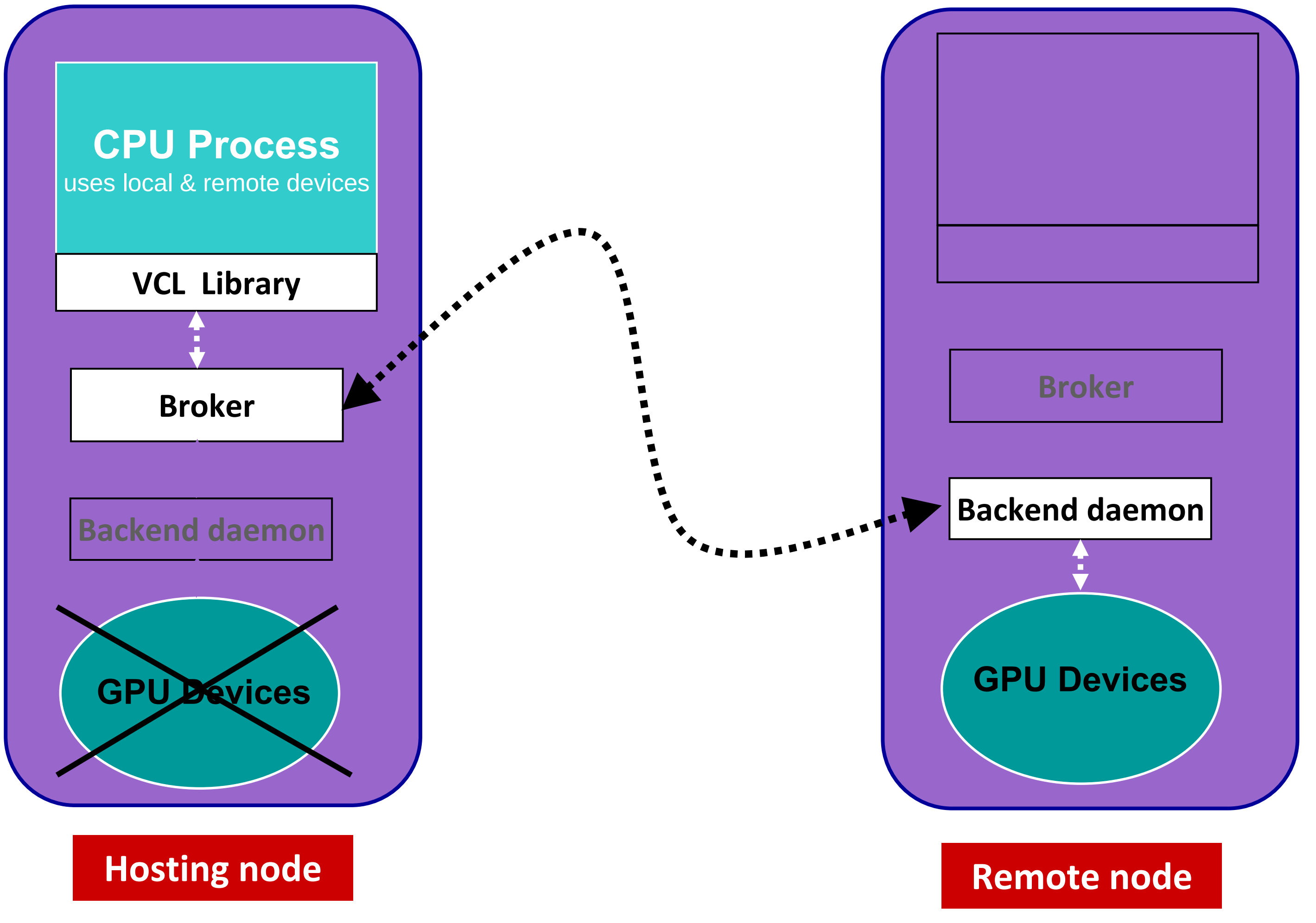
The Backend Server Daemon

- Reserves devices for contexts of VCL library clients.
  - For security, only one client per device.
- Performs operations on behalf of the VCL library clients.
- Uses any standard OpenCL SDK (on the node where it runs).
- Continuously reports device availability to the brokers.

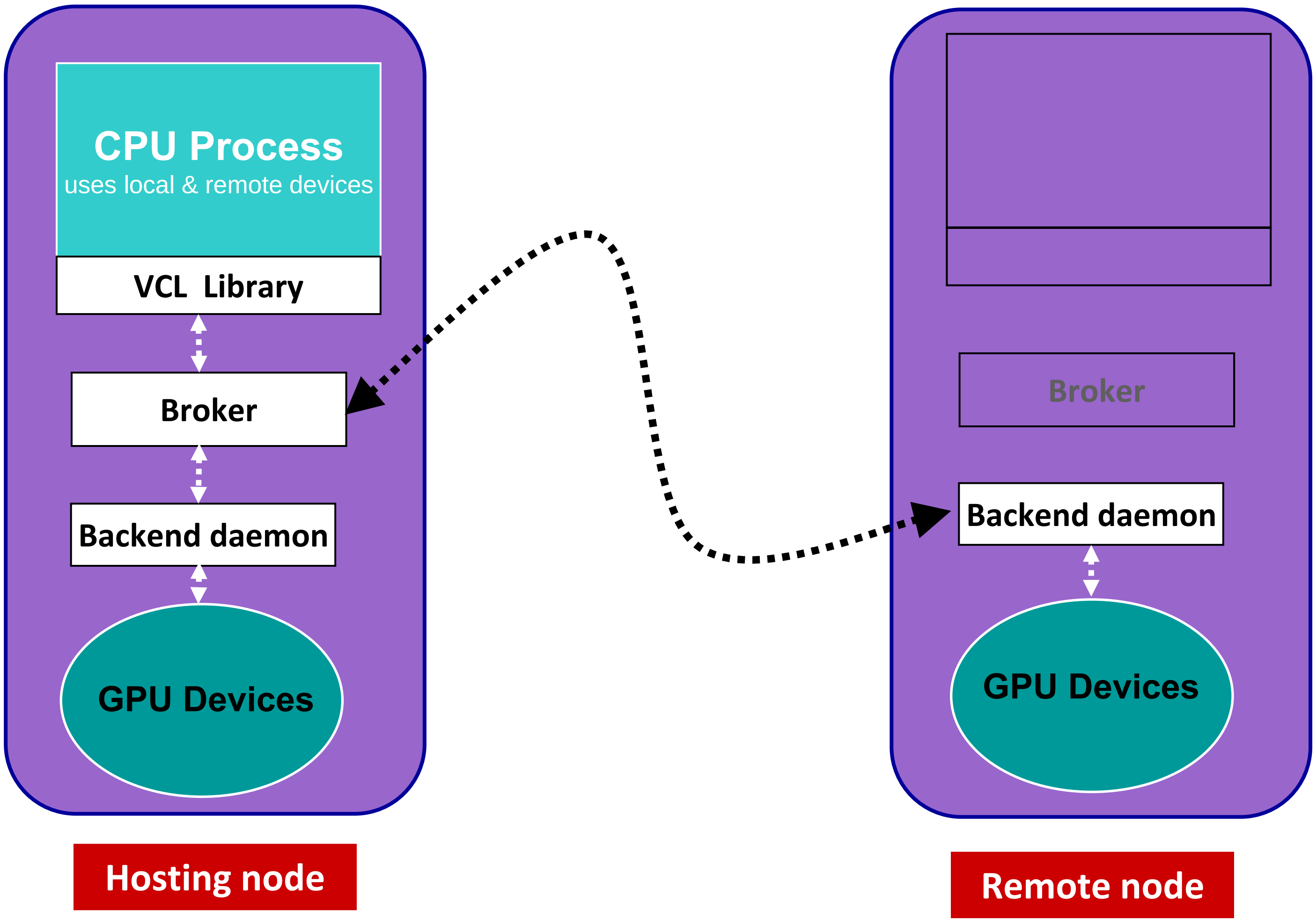
Example: Process Using Local GPUs



Process Using Remote GPUs



Process Using Local and Remote GPUs



VCL Overhead to Start a Kernel

Runtime (ms) vs. Buffer size to run 1000 pseudo kernels.

Native: OpenCL library on local device.
Local = VCL on local device.
Remote = VCL on a remote device.

Outcome: a fixed overhead by VCL for all buffer sizes.

Buffer Size	Native time (ms)	VCL Overhead on Local device	VCL Overhead on Remote Device
4KB	96	(96)+35	(96)+113
16KB	100	(100)+35	(100)+ 111
64KB	105	(105)+35	(105)+ 106
256KB	113	(113)+36	(113)+ 105
1MB	111	(111)+34	(111)+ 114
4MB	171	(171)+ 36	(171)+ 114
16MB	400	(400)+36	(400)+ 113
64MB	1,354	(1,354)+33	(1,354)+ 112
256MB	4,993	(4,993)+37	(4,993)+ 111
Average Overhead		Δ = 35μs	Δ = 111μs

Selected SHOC Benchmark Runtimes

Application	Native time (Sec.)	VCL Times (Sec.)		Comments
		Local	Remote	
KernelCompile	5.91	5.93	5.94	VCL only transfer source code
FFT	7.29	7.15	7.33	Small data, long compute - ideal for VCL
MD	14.08	13.66	13.80	Small data, long compute - ideal for VCL
Reduction	1.60	1.58	2.88	Moderate compute - moderate performance
SGEMM	2.11	2.13	2.43	Much data & compute - reasonable overhead
Scan	2.53	2.54	6.57	Large data, little compute – poor remote results
Sort	0.98	1.04	1.53	More compute – better results
Spmv	3.25	3.30	5.91	Huge data, significant compute - moderate result
Stencil2D	11.65	12.48	18.94	Huge data - bandwidth to remote device limiting factor
S3D	32.39	32.68	33.17	Small data, long compute - ideal for VCL

Outcome: more computing power, but network latency/bandwidth are limiting factors for I/O intensive applications.

SHOC - FFT Performance on a Cluster

256 MB buffer, 1000 – 8000 iterations on 1, 4 and 8 nodes, each with 1GPU, connected by Infiniband*.

Number of Iterations	Native time (Sec.)	4 Nodes		8 Nodes	
		Time (Sec.)	Speedup	Time (Sec.)	Speedup
1000	42.34	19.27	2.19	16.29	2.60
2000	82.25	30.11	2.73	22.03	3.73
4000	162.17	52.58	3.08	33.37	4.86
8000	321.91	97.53	3.29	55.95	5.74

***A. Barak, T. Ben-Nun, E. Levy, and A. Shiloh, ``A package for OpenCL based heterogeneous computing on clusters with many GPU devices'', Proc. PPAAC, IEEE Cluster 2010.**

Personalized Medicine Example

Pinpoints a selected number of genes and their corresponding weights to determine response to a clinical medication or treatment.

Requires parallel operations (t-test) on all permutations of genes of a group of patients*.

Program	Runtime	Speedup
High level package (CPU)	~40 hours	1
C++ code	~25 hours	1.6
Mix: C++ and OpenCL	~ 1.5 hours	26
Serial OpenCL - 1 GPU	30 min .	80
Parallel OpenCL - 2 GPUs	1:42 min.	1412
Parallel OpenCL - 4 GPUs	0:54 min.	2666

Outcome: GPU times makes it possible for day-to-day use, to run tests on much larger groups of patients.
Program development: 2 months (by a CS student).
Optimizations: 8 months (with help of experts).

***Joint work with Y. and J. Smith, Hadassah Medical School**

VCL Support for SLURM

Provides a per-job private ad-hoc VCL cluster, based on SLURM's allocation rather than having a fixed cluster.

- **Includes the necessary SLURM prologs and epilogs to establish and destroy this private cluster.**
- **Informs SLURM when VCL detects insufficient OpenCL devices.**
- **Includes instructions for SLURM administrators and users on how to incorporate VCL into SLURM.**

VCL Support for MPI and Multitasking

Includes a pre-allocation option, to prevent improper competition for devices between ranks.

Includes an option to ban unwanted devices, making them invisible to the application.

Other Benefits

Break down of VCL to independent components allows the introduction of various runtime services:

- **A single queue per application** for all devices.
  - Improves the overall utilization.
- **Scheduling:** assign the next kernel to the best available device.
  - Optimizations: if possible, a kernel is assigned to a device that already holds its input buffers.
- **Buffer management:** allocation and release of buffers on devices and tracking of their available memory.
- **Task dependencies:** a task may run once all its input memory buffers updated by preceding tasks.

Optimizations and Extensions

Direct loading of memory objects from remote files – no need to read data via the hosting node.

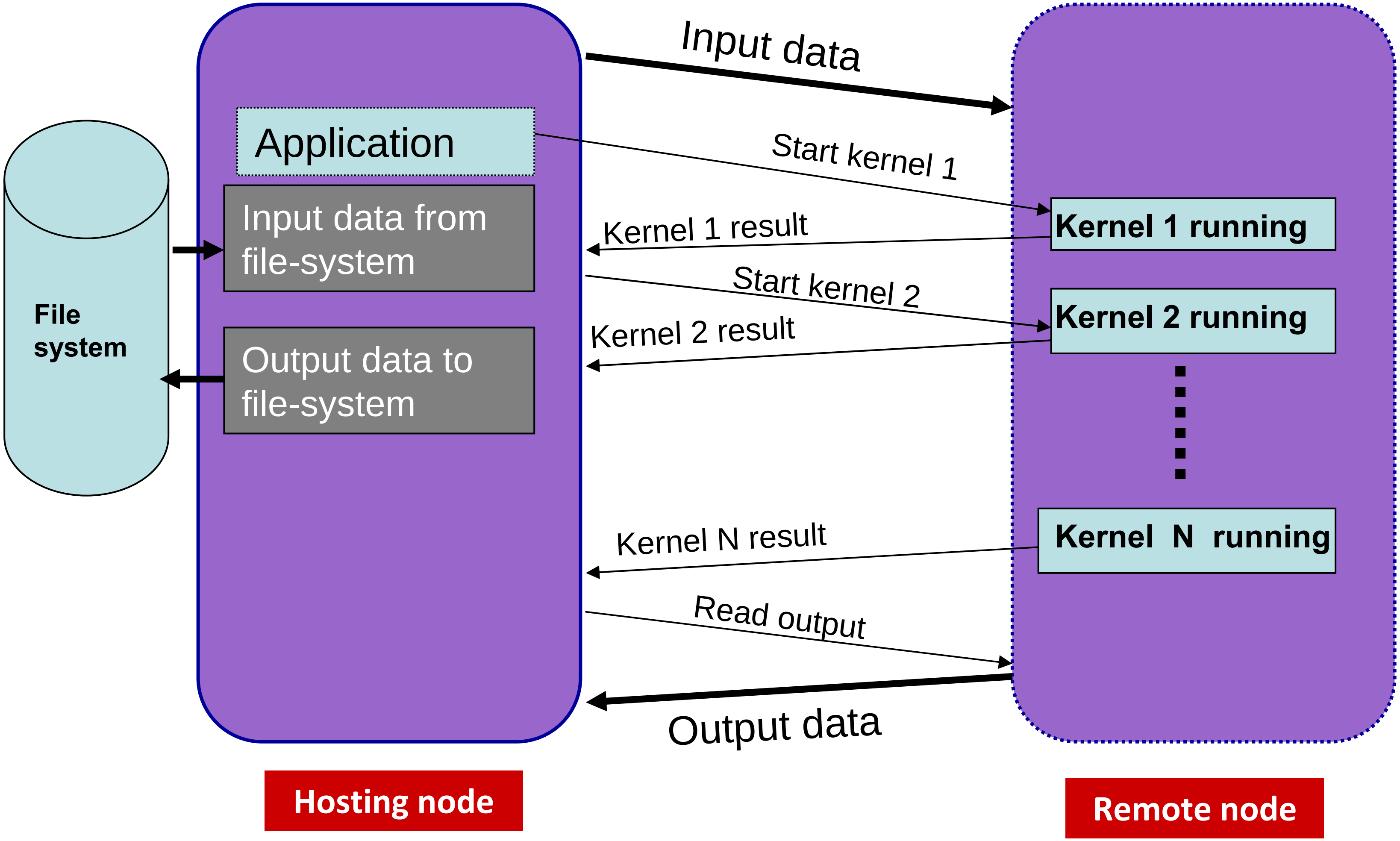
SuperCL – a mini-programming language for performing multiple OpenCL operations on remote devices without involving the host in between.

The **Many GPUs Package** (MGP)*:

- C++ and OpenMP extensions to transparently utilize many devices.

*** A. Barak, T. Ben-Nun, E. Levy, and A. Shiloh, ``A package for OpenCL based heterogeneous computing on clusters with many GPU devices'', Proc. PPAAC, IEEE Cluster 2010.**

Before SuperCL



Latency due to several round-trips

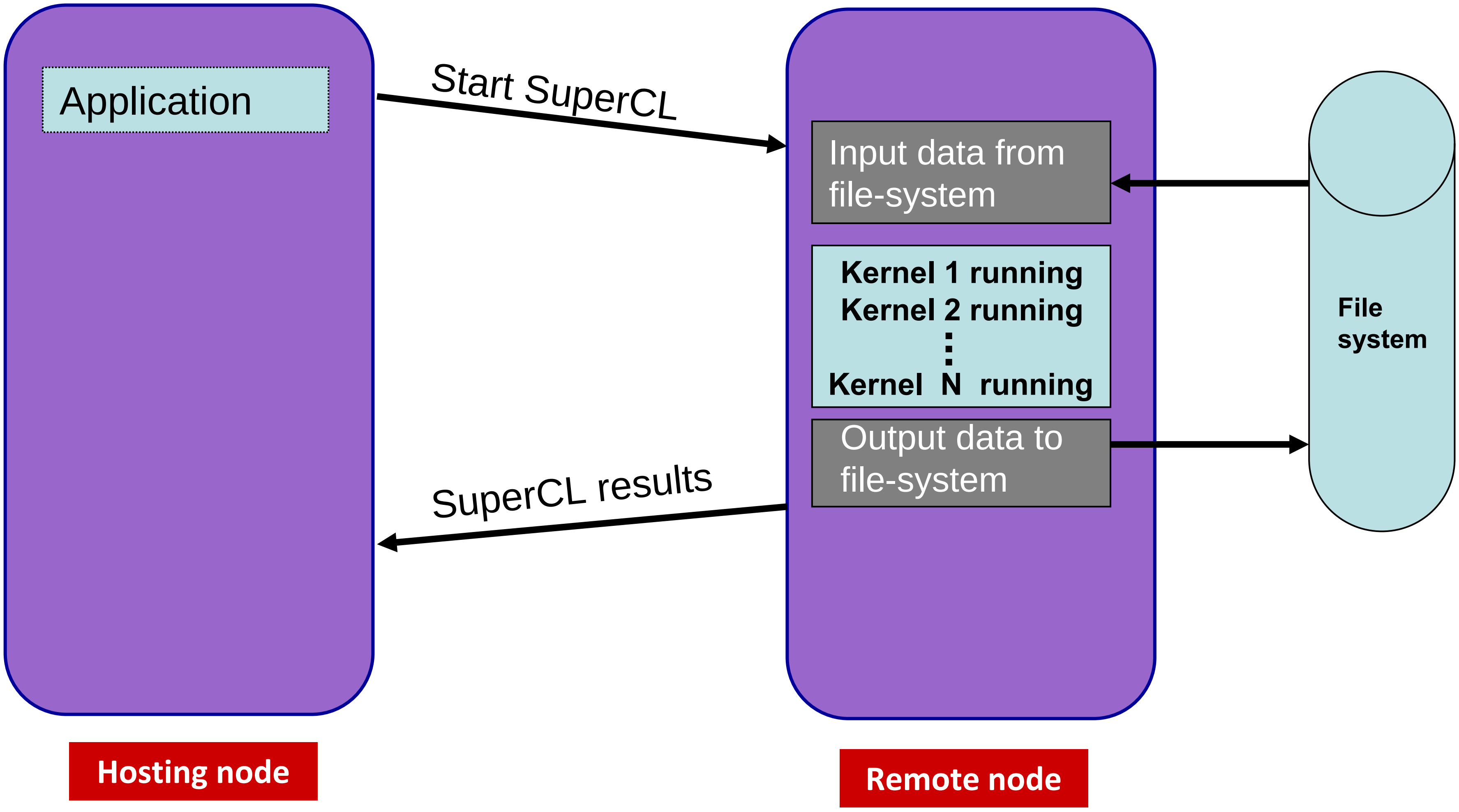
SuperCL

A **mini-programming language** for reducing network overheads.

- Run a sequence of kernels and/or memory-object operations in a single library call.
- Direct file I/O to/from OpenCL memory-object, so that the data needs not pass through the host.
- Asynchronous transfer of data with the host, to prevent waiting on latency.
- Wide range of logic/control available for complex high-level algorithms at the remote end, thus relieving the host CPU.

- **Open-end research for further optimizations and extensions for running applications on multiple nodes.**

With SuperCL



One round trip

Extensions of the C++ and OpenMP API's

High-level language extensions for managing parallel jobs on many GPUs:

- Devices are automatically handled by VCL.
- Supports advanced features such as scatter-gather and profiling of kernel times.

Example: the Scatter-Gather API allows buffers to be divided into disjoint segments that can be transparently scattered to multiple devices.

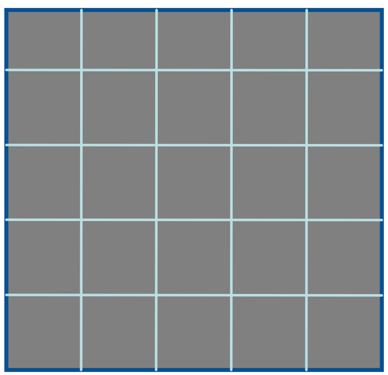
Geared for tasks that need to perform:

- Subdivision of arrays (matrices).
- Boundary exchanges.
- Gather (merge disjoint arrays).

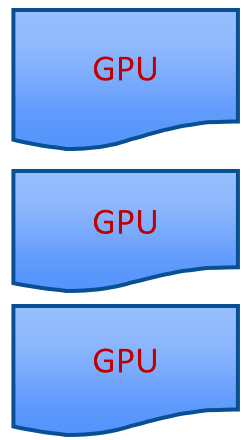
Scatter-Gather Example

Stencil2D, a 9-point weighted average application from SHOC.

- MPI implementation - uses grid-blocks: ~655 lines-of-code.
- OpenMP implementation uses stripes (easier to manage and scale-up).
  - Only 64 lines.



Hosting node

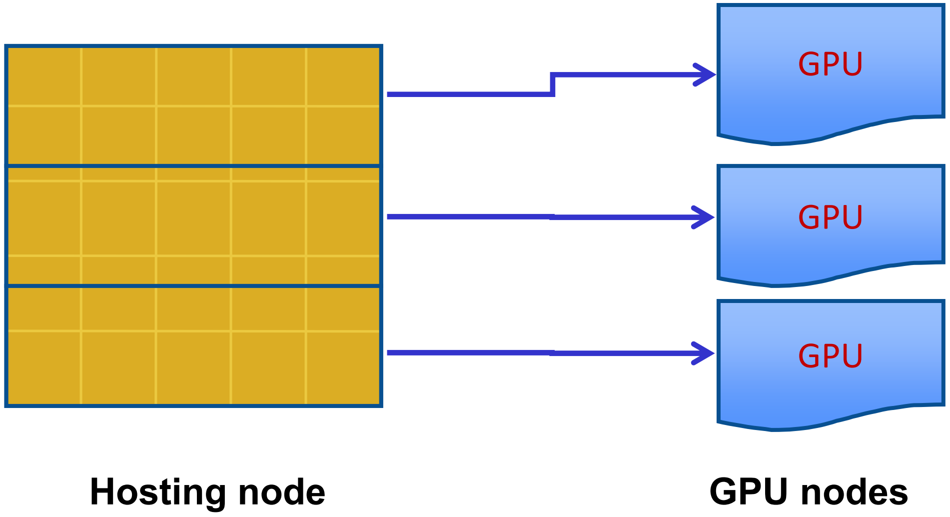


GPU nodes

Scatter

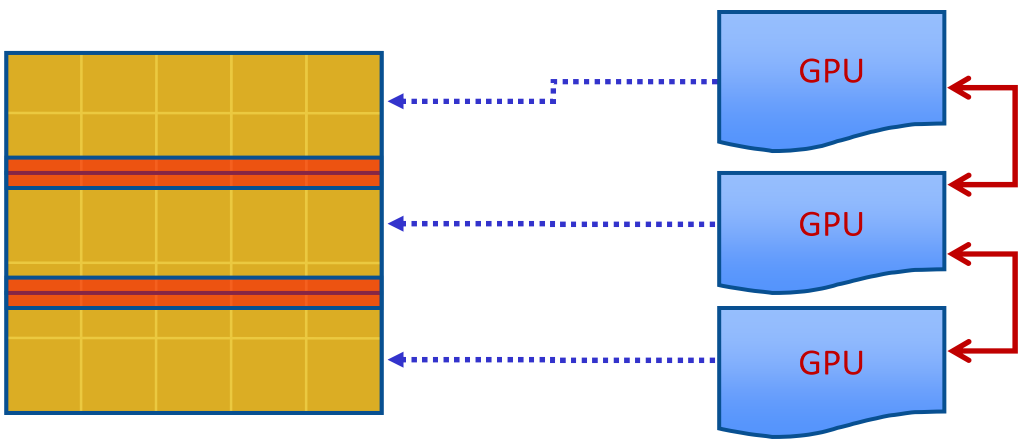
Stripes are sent to different GPUs.

-Useful to run large matrices that do not fit in a single GPU or even in the hosting node.



Exchange

Direct boundary exchanges between GPU nodes

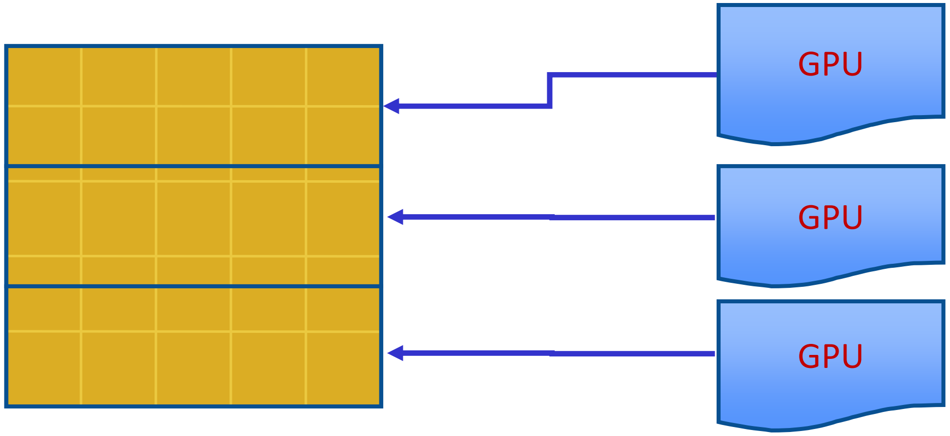


Hosting node **GPU nodes**

Gather

Stripes are gathered from GPUs to the hosting node.

- Or to a file.



Hosting node **GPU nodes**

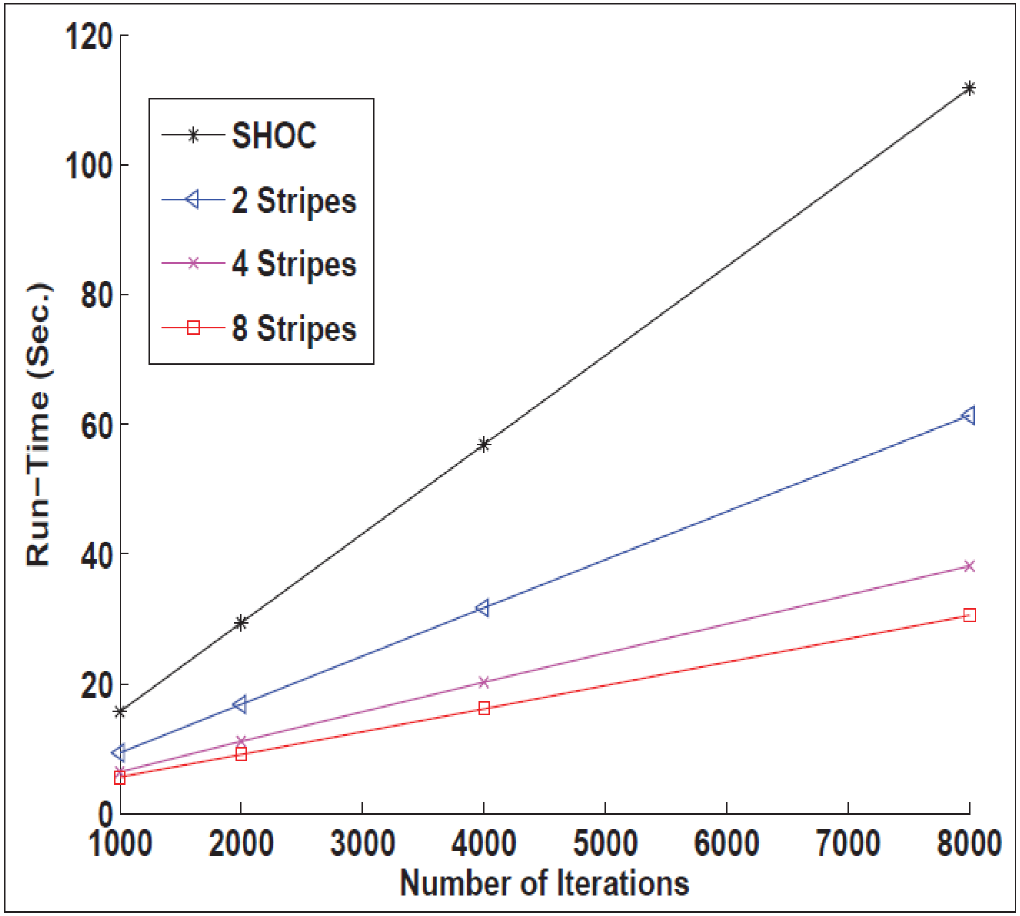
Scatter-Gather Performance - Stencil2D

8kX8k (256MB) matrix vs. number of iterations.

SHOC times (single GPU, no buffer net transfer, no boundary exchanges)

vs.

OpenMP with 2, 4 and 8 stripes, each on a different node, including transfer of buffers over the network and boundary exchanges.



Conclusions

Heterogeneous computing can dramatically increase the speedup of many applications.

– Due to the programming complexity, it is necessary to develop tools for debugging, monitoring, program optimizations, scheduling, resource management and make it easy to run.

VCL makes it easier to run applications on clusters with many OpenCL devices.

– Scalability depends on the tradeoff between compute vs. communication.