The VirtualCL (VCL) Cluster Platform

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

- Hosting node
  - CPU Process
    - uses local & remote devices
  - VCL Library
  - Broker
  - Backend daemon
  - GPU Devices

- Remote node
  - Broker
  - Backend daemon
  - GPU Devices
Process Using Local and Remote GPUs

- CPU Process uses local & remote devices
  - VCL Library
  - Broker
  - Backend daemon
  - GPU Devices

Hosting node

Remote node

- Broker
- Backend daemon
- GPU Devices
# VCL Overhead to Start a Kernel

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

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

**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.
## Selected SHOC Benchmark Runtimes

<table>
<thead>
<tr>
<th>Application</th>
<th>Native time (Sec.)</th>
<th>VCL Times (Sec.)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Local</td>
<td>Remote</td>
</tr>
<tr>
<td>KernelCompile</td>
<td>5.91</td>
<td>5.93</td>
<td>5.94</td>
</tr>
<tr>
<td>FFT</td>
<td>7.29</td>
<td>7.15</td>
<td>7.33</td>
</tr>
<tr>
<td>MD</td>
<td>14.08</td>
<td>13.66</td>
<td>13.80</td>
</tr>
<tr>
<td>Reduction</td>
<td>1.60</td>
<td>1.58</td>
<td>2.88</td>
</tr>
<tr>
<td>SGEMM</td>
<td>2.11</td>
<td>2.13</td>
<td>2.43</td>
</tr>
<tr>
<td>Scan</td>
<td>2.53</td>
<td>2.54</td>
<td>6.57</td>
</tr>
<tr>
<td>Sort</td>
<td>0.98</td>
<td>1.04</td>
<td>1.53</td>
</tr>
<tr>
<td>Spmv</td>
<td>3.25</td>
<td>3.30</td>
<td>5.91</td>
</tr>
<tr>
<td>Stencil2D</td>
<td>11.65</td>
<td>12.48</td>
<td>18.94</td>
</tr>
<tr>
<td>S3D</td>
<td>32.39</td>
<td>32.68</td>
<td>33.17</td>
</tr>
</tbody>
</table>

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 1 GPU, connected by Infiniband*.

<table>
<thead>
<tr>
<th>Number of Iterations</th>
<th>Native time (Sec.)</th>
<th>4 Nodes</th>
<th>8 Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (Sec.)</td>
<td>Speedup</td>
<td>Time (Sec.)</td>
</tr>
<tr>
<td>1000</td>
<td>42.34</td>
<td>19.27</td>
<td>2.19</td>
</tr>
<tr>
<td>2000</td>
<td>82.25</td>
<td>30.11</td>
<td>2.73</td>
</tr>
<tr>
<td>4000</td>
<td>162.17</td>
<td>52.58</td>
<td>3.08</td>
</tr>
<tr>
<td>8000</td>
<td>321.91</td>
<td>97.53</td>
<td>3.29</td>
</tr>
</tbody>
</table>

**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*.

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

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 epiloggs 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.
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.

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

**Application**

Start SuperCL

**Hosting node**

Input data from file-system

Kernel 1 running
Kernel 2 running
... 
Kernel N running

Output data to file-system

**Remote node**

One round trip

**File system**
**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.

![Diagrams]( 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.

Hosting node

GPU nodes
Direct boundary exchanges between GPU nodes
Stripes are gathered from GPUs to the hosting node.
- Or to a file.
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.