The MOSIX Virtual OpenCL (VCL) Cluster Platform

Amnon Barak and Amnon Shiloh
Computer Science, Hebrew University

mosix@mosix.org
Overview

The VCL cluster platform is an OpenCL platform that can transparently run unmodified OpenCL applications on a cluster with many devices, as if all the devices are on each hosting node.

Motivation: ease the development and running of concurrent HPC applications.

The VCL programming paradigm:
- CPU (administrative) part runs on a single (multi-core) hosting node.
- GPU (computing) parts runs on cluster-wide devices.

Targeted for:
- Parallel, intensive computing applications that can utilize many devices concurrently.
- Many users that share a pool of devices, e.g. in a cloud.
**Background**

Most OpenCL platforms consist of vendor specific hardware devices and matching **SDKs**:
- A **compiler** of OpenCL programs (kernels).
- A **runtime environment** that assign kernels to local devices.

Currently, applications that utilize GPU/APU devices, run their kernels only on devices of the same computer where the applications run. **VCL** is a **runtime environment** that extends OpenCL to support cluster and cloud computing:
- Implements the OpenCL standard.
- Supports any 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 GPU devices.
The VCL Runtime Model

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

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 available cores in the hosting node.

Kernels can run on multiple devices: GPUs, APUs.
- Location of devices is transparent to the process.
The VCL Programming Paradigm

Combines the benefits of the MPI approach and the OpenMP approach.

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

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

Outcome:
- Full benefit of VCL manifest with 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.
- Can even benefit MPI applications with intensive GPU computing, by running the CPU part on a single multi-core computer.

Challenge: extend VCL to run CPU threads on multiple nodes.
The VCL Components

VCL consists of 3 components:

- The VCL library - implements the OpenCL library.
- The broker - routing and arbitration daemon.
- The backend server daemon.
The VCL Library

A cluster implementation of the OpenCL library according to the formal spec.
  - Allows 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 host.
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-wide devices.
- Responsible for authentication and access permissions.
- Routes messages between the VCL library and the backend OpenCL server daemon.
  - Separates applications from the network layers, to prevent blocking.
The Backend Server Daemon

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

CPU Process uses local & remote devices

VCL Library

Broker

Backend daemon

Hosting node

Remote node

GPU Devices

GPU Devices

Broker

Backend daemon
Process Using Local and Remote Devices

CPU Process uses local & remote devices

Hosting node

Remote node

VCL Library

Broker

Backend daemon

GPU Devices

Backend daemon

GPU Devices

Broker
VCL Overhead to Start a Kernel

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

On ATI 6970 via Infiniband.

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.

Identical results with 2-4 devices per node.

<table>
<thead>
<tr>
<th>Buffer Size</th>
<th>OpenCL Library (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>
</tbody>
</table>

Average Δ Overhead | 35μs | 111μs
## Selected SHOC Benchmark Runtimes

<table>
<thead>
<tr>
<th>Application</th>
<th>OpenCL Library (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>

Similar results for the ATI-stream SDK benchmark.

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, connected by Infiniband*.

<table>
<thead>
<tr>
<th>Number of Iterations</th>
<th>OpenCL Library (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 Alg. 1 GPU</td>
<td>30 min</td>
<td>80</td>
</tr>
<tr>
<td>Parallel OpenCL Alg. 2 GPUs</td>
<td>1:42 min</td>
<td>1412</td>
</tr>
<tr>
<td>Parallel OpenCL Alg. 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 (part time with help of experts).

*Joint work with Y. and J. Smith, Hadassah Medical School
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 ready-to-run 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 low-level mini-language geared for network optimizations.

The Many GPUs Package (MGP)*:

- A high-level language extensions to transparently utilize many devices.
  - Implemented for C++ and OpenMP.

SuperCL

Extension of the VCL library to reduce network overheads:

- A **mini-programming language** for performing multiple OpenCL operations on remote devices, without having to involve the host in between.
- Run a sequence of kernels and/or memory-object operations in a single library call.
- Can fetch data directly from/to remote nodes.
- Parallelization is achieved by multiple calls to SuperCL on different nodes.
- Open-end research platform for further optimizations and extensions for running CPU processes on multiple nodes.
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.
– Library 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).
  - 64 lines.

![Diagram showing a hosting node and GPU nodes](image)
**Scatter**

Stripes are allocated 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 remote nodes

Hosting node  GPU nodes
Gather

Stripes are gathered from GPUs to the hosting node.
- Or to a file via the hosting node.
**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.
**Target Architecture for HPC**

- **Multi-core APUs**, with a CPU and several GPUs.
  - Viewed as one big GPU.
- **Cluster nodes**: low-cost/low-power servers with several APUs, connected by superfast bus.
- **Hosting nodes**: many-cores workstations and servers, with large memory.
  - To support multi-threaded applications.
- **Low latency communication**:
  - Via a common bus in the same node.
  - LANs between different nodes.
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.

As in the MOSIX cluster OS that provides a SSI, VCL makes it easier to run OpenCL applications on clusters.

- Scalability depends on the tradeoff between compute vs. communication.
Ongoing and Planned Projects

- On-line algorithms for dynamic resource management - cluster-wide load-balancing, fair-share
- How to choose the “best” device.
- Load-balancing CPUs vs. GPUs.
- SuperCL for further runtime optimizations.
- Extend VCL to run CPU threads on multiple nodes.
- Scheduling algorithms in a multicore APUs (several APUs on same Motherboard) and APU clusters.
- Extend the MGP API with MapReduce.
- R&D of OpenCL applications – MD, CFD, P. medicine
- Tune MPI to use VCL – all processes run in the hosting node, with fast communication via shared-memory.
Future Plans

Continue to develop, maintain and provide VCL services to:

- Cluster distributors and end-users.
- Cloud providers.

Build a “large” GPU/APU HPC cluster for testing the performance and scalability of parallel applications.

The latest distribution of VCL for Linux platform is available at:

http://www.MOSIX.org-txt_vcl.html
Disclaimer & Attribution
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.
The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.
NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.
ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.
The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD’s positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.