Distributed Heterogeneous Computing with *cl*OpenCL

Tiago Ribeiro[1] Mário Afonso[1] José Rufino[1] Albano Alves[1]

[1] Polytechnic Institute of Bragança, Bragança, Portugal
{tiago.f.ribeiro, mario.jd.afonso}@alunos.ipb.pt / {rufino, albano}@ipb.pt

Context:

Clusters of heterogeneous computing nodes provide an opportunity to significantly increase the performance of parallel and High-Performance Computing (HPC) applications, by combining traditional multi-core CPUs coupled with accelerator devices, interconnected by high throughput and low latency networking technologies. However, developing efficient applications to run in clusters that integrate GPUs and other accelerators often requires a great effort, demanding programmers to follow complex development methodologies in order to suit algorithms and applications to the new heterogeneous parallel environment.

OpenCL is an open programming standard for heterogeneous computing. It suffers, however, from a major limitation: applications can only make use of the local compute devices, present on a single machine. *cl*OpenCL (cluster OpenCL) supports the simple deployment and efficient running of OpenCL-based parallel applications that may span several cluster nodes, expanding the original single-node OpenCL model.

*cl*OpenCL Architecture:

An OpenCL application comprises an *host* program and a set of *kernels* intended to run on compute *devices*.

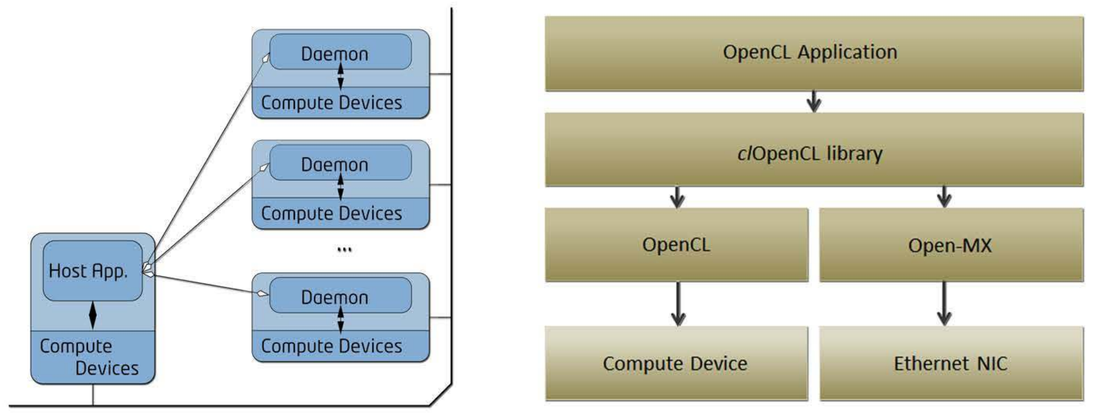
Figure 1 presents the i) *cl*OpenCL operation model, where a single *host* program interacts with multiple compute *devices* (local or remote), and ii) the software/hardware layers present in the *host* program.

Every call to an OpenCL primitive is intercepted by the *cl*OpenCL wrapper library which redirects its execution to a specific *cl*OpenCL daemon at a cluster node or to the local OpenCL runtime.

*cl*OpenCL daemons are simple OpenCL programs that handle remote calls and interact with local *devices*. Each user spawns its own set of daemons, eventually sharing particular compute devices with other cluster users.

A typical *cl*OpenCl application starts at a particular cluster node and will create OpenCl objects in cluster nodes. For each object, the *cl*OpenCL wrapper library returns a ``fake pointer" used as a global identifier, and stores the real pointer along with the corresponding daemon location.

The exchange of data between the wrapper library and remote daemons uses Open-MX, an open-source message passing stack over generic Ethernet, which provides low-level communication mechanisms at user-level space and allows to achieve low latency communication and low CPU overhead.



i) Operation Model ii) Layers

Figure 1: *cl*OpenCL Architecture.

Evaluation:

Testbed: 4-node Linux ROCKS commodity cluster; each node: Intel Q9650 3GHz CPU (12Mb L2 cache), 8Gb DDR3 1333MHz, 1Gbps Ethernet NIC SysKonnect SK-9871, NVIDIA GTX 460 GPU (1Gb GDDR5); one of the nodes with an additional GPU; 1Gbps Ethernet switch with jumbo frames (mtu 9000) active; AMD OpenCL SDK 2.6; NVIDIA CUDA 4.1.28; Open-MX 1.5.2.

Application: simple and "embarrassingly parallel" case study - *matrix product* for square matrices, of order n ∈ {8K, 16K, 24K} and single-precision (floats) elements; naive OpenCL kernel for sliced matrix product; multi-threaded (POSIX Threads) *host* OpenCL application with dynamic work distribution and load balancing (one thread per each cluster OpenCL device involved in the matrix product; number of work slices processed by a device depends of its relative computing power).

```
_kernel void matrix_mult(const int size, const int slice,
            __global float *A, __global float *B, __global float *C){
    int i, j, k; float v=0;

    i = get_global_id(0); j = get_global_id(1);
    for(k=0; k<size; k++)
        v += A[i*size+k] * B[j*size+k];
    C[i*slice+j] = v;
}
```

Figure 2: A simple kernel for sliced matrix product.

#C \ #G	2	3	4	5
1	GGC	GGC,G	GGC,G,G	GGC,G,G,G
2	GGC,C	GGC,G,C	GGC,G,G,C	GGC,GC,G,G
3	GGC,C,C	GGC,G,C,C	GGC,GC,G,C	GGC,GC,GC,G
4	GGC,C,C,C	GGC,GC,C,C	GGC,GC,GC,C	GGC,GC,GC,GC

Table 1: Performance Optimal Combinations of OpenCL Devices

Configurations: The *host* application component always started in node-0, the node with the most performant set of OpenCL devices (1 CPU and 2 GPUs). Using all devices of node-0 is the best scenario a traditional OpenCL application can exploit in our cluster; from there onwards, the only way to increase performance is to use remote devices, e.g., through *cl*OpenCL. Remote devices combine in many ways with devices of node-0. Table 1 shows only the subset of combinations that, for a certain number of CPUs (#C) and GPUs (#G), offer the best performance; devices are denoted by C and G and organized in comma separated groups (the 1st group is from node-0, the 2nd from node-1, etc.); any combination in Table 1 obeys to the same general rule: it uses the maximum possible number of local devices (on node-0) and it scatters the remote devices as much as possible.

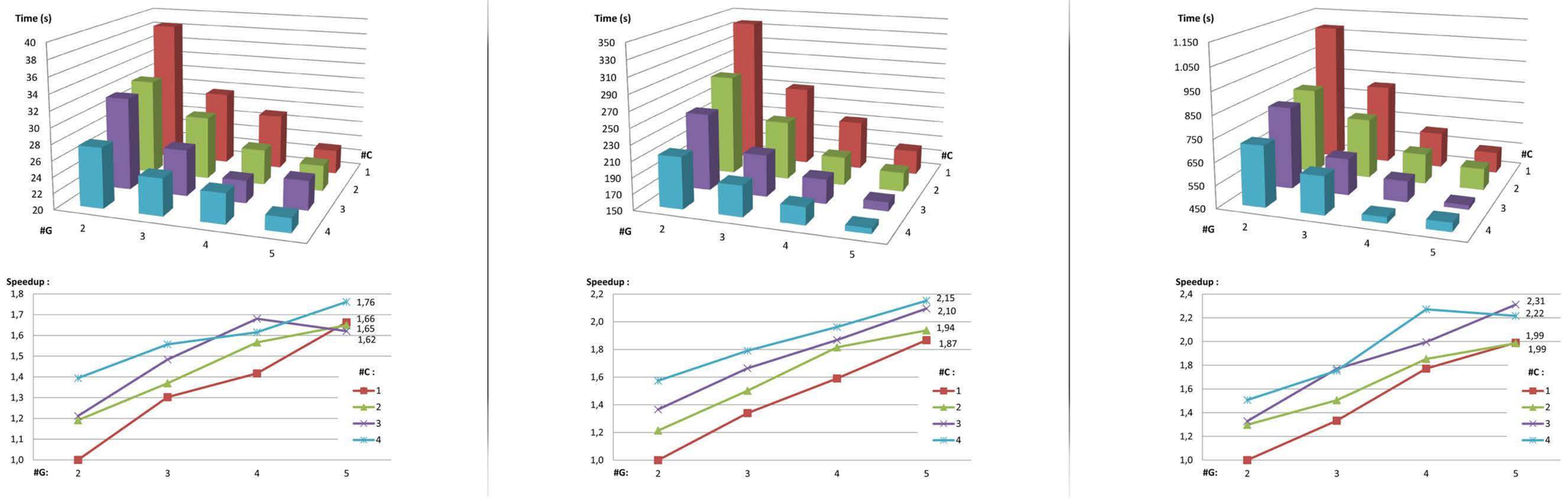


Figure 2: Execution time and Speedups for orders 8K, 16K, 24K.

Execution Time and Speedups: Figure 2 shows the matrix product times and speedups relative to combination GGC (speedup = 1), with matrices of order 8K, 16K and 24K, for all device combinations of Table 1. As expected, the matrix produc times increase, but so also speedup, showing that the scalability of *cl*OpenCL improves with the problem size.

n	S^{e}_{max}	S^{t}_{max}	$R^{e/t}_{max}$
8K	1,76	13/5 = 2,6	67,7%
16K	2,15	14/5 = 2,8	76,8%
24K	2,31	14/5 = 2,8	82,5%

Table 2: Effective to Theoretical Speedup Ratios

Effective to Theoretical Speedup Ratios: Compared to OpenCL, speedups achieved by *cl*OpenCL may seem modest. Table 2 clarifies the merits of *cl*OpenCL. For each matrix order (n), the table shows the maximum effective (measured) speedup of *cl*OpenCL over OpenCL (S^{e}_{max}), the maximum theoretical (nominal) speedup of *cl*OpenCL over OpenCL (S^{t}_{max}), and how close is the first one from the second one (the ratio $R^{e/t}_{max} = S^{e}_{max} / S^{t}_{max}$). The values of S^{t}_{max} result from the observation that, in our cluster, a GPU processes twice the work slices than a CPU; such allows to derive the speedup achievable by an hypothetical node that could host all the CPUs and GPUs of the fastest device combination.

Conclusions:

*cl*OpenCL enables the execution of OpenCL applications in heterogeneous clusters and has two main advantages over similar projects: it is able to take full advantage of commodity networking hardware through Open-MX, and programmers/users do not need special privileges neither exclusive access to scarce resources to deploy the desired running environment. An embarrassingly parallel program, using different problem sizes and combinations of local/remote devices, showed that *cl*OpenCL is useful for exploiting multi-node-multi--GPU environments. Future work on *cl*OpenCL will expand the OpenCL standard coverage, add support for BSD sockets and extend the evaluation with further parallel applications.

INSTITUTO POLITÉCNICO DE BRAGANÇA Escola Superior de Tecnologia e Gestão

Funded by:
FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

Project Part-Financed by the European Union
European Regional Development Fund