Chapter 3

Runtimes, Software, API and Benchmarks

3.1 Introduction

After presenting the rules of HPC and the hardware that compose the cluster we need to introduice ways to target this supercomputer. Several options are present in the language, the multi-processing API, the distribution and the accelerators code. This chapter details the most important software options for HPC programming and include the choices we made for our applications.

Then it presents the software used to benchmark the supercomputers. We present here the most famous, the TOP500, GRAPH500 and GREEN500 to give their advantages and weaknesses.

Parler des pitfalls, load balancing, concurrence, ... NON dans la partie 2 avant la metrique mise en place ?

3.2 Sofware/API

In this section we present the main runtimes, API and programming language use in HPC and in this study in particular. The considered language will be C/C++, the most present in HPC world.

3.2.1 Parallel programming

PThreads

The POSIX threads API is an execution model available in most of the languages. It allows the user to define threads that will execute concurrently on the processor ressources using shared/private memory. PThreads is the low level handling of threads and the user need to handle concurrency with mutex, conditions variables and synchronization "by hand". This makes the PThreads hard to use in complex applications and used only for very fine-grained control over the threads management. Fine-grain Coarse-grain applications

OpenMP

Open Multi-Processing, OpenMP[1] [Cha08, Sup17], is an API for multi-processing shared memory like UMA and CC-NUMA. It is available in C/C++ and Fortran. The user is provided with pragmas and functions to declare parallel loop and regions in the code. In this model the main thread, the first one before forks, command the fork-join operations.

[1] http://www.openmp.org

The last versions of OpenMP also allow the user to target accelerators. During compilation the user specify on which processor or accelerator the code will be executed in parallel.

3.2.2 Distributed programming

In the cluster once the code have been developped locally and using the multiple cores available, the new step is to distribute it all over the nodes of the cluster. This step requires the processes to access NoRMA memory from a node to another. Several runtime are possible for this purpose.

MPI

The Message Passing Interface, MPI, is the most and widely spread runtime for distributed computing [Gro14, Gro15]. Several implementations exists from Intel MPI[2] (IMPI), MVAPICH[3] by the Ohio State University and OpenMP[4] combining several MPI work like Los Alamos MPI (LA-MPI). Those implementation follow the MPI standards 1.0, 2.0 or the latest, 3.0. Who define standards ?

Some MPI implementation offer a support for accelerators targeting directly their memory through the network without multiple copies on host memory.

Charm++

Charm++[5] is another API for distributed programming developped by the University of Illinois Urbana-Champaign. It is asynchronous messages paradigm driven. In contrary of runtime like MPI that are synchronous but can handle asynchronous, charm++ is natively asynchronous. It is based on *chare object* that can be activated in response to messages from other *chare objects* with actions and callbacks. The repartition of data to processors is completely done by the API, the user just have to define correctly the partition of the program. Charm++ also provide a GPU manager implementing data movement, asynchronous kernel launch, callbacks, etc.

A perfect example can be the hydrodynamics N-body simulation code Charm++ N-body Gravity Solver, ChaNGa [JWG+10], implemented with charm++ and GPU support.

Legion

The Legion[6] is a distributed runtime support but Stanford University, Los Alamos National Laboratory and NVIDIA. This runtime is data-centered targeting distributed heterogeneous architectures. Data-centered runtime focus to keep the data dependency and locality moving the tasks to the data and moving data only if requested. In this runtime the user define data organization, partitions, priviledges and coherence. Many aspect of the distribution and parallelization are then handle by the runtime itself.

3.2.3 Other tools

HPX ? Others ?

3.2.4 Accelerators

CUDA

The Compute Device Unified Architecture is the API develop in C/C++ Fortran by NVIDIA to target its GPGPUs. The API provide high and low level functions. The driver API allows a fine grain control over the executions.

[2] https://software.intel.com/en-us/intel-mpi-library
[3] http://mvapich.cse.ohio-state.edu/
[4] http://www.open-mpi.org
[5] http://charmplusplus.org/
[6] http://legion.stanford.edu/

The CUDA compiler is called NVdia C Compiler, NVCC. It converts the device code into Parallel Thread eXecution, PTX, and rely to the C++ host compiler for host code. PTX is a pseudo assembly language translated by the GPU in binary code that is then execute. As the ISA is pretty simple it able the user to work directly in assembly for very fine grain optimizations.

Specific tools have been made for HPC in the NVIDIA GPGPUs.

Dynamic Parallelism This feature allow the GPU kernels to run other kernels themself. This feature

Hyper-Q This technology enable several CPU threads to execute kernels on the same GPU simultaneously. This can help to reduce the synchronization time and idle time of CPU cores for specific applications.

NVIDIA GPU-Direct GPUs' memory and CPU ones are different and the Host much push the data on GPU before allowing it to compute. GPU-Direct allows direct transferts from GPU devices through the network. Usually implemented using MPI.

OpenCL

OpenCL is a multi-platform framework targeting a large part of nowadays architectures from processors to GPUs, FPGAs, etc. A large group of company already provided conform version of the OpenCL standard: IBM, Intel, NVIDIA, AMD, ARM, etc. This framework allows to produce a single code that can run in all the host or device architectures. It is quite similar to NVIDIA CUDA Driver API and based on kernels that are writen and can be used in Online/Offline compilation meaning Just In Time (JIT) or not. The idea of OpenCL is great by rely on the Indeed, one may wonder, what is the level of work done by NVIDIA on its own CUDA framework compare to the one done to implement OpenCL standards? What is the advantage for NVIDIA GPU to be able to be replace by another component and compare on the same level? Those questions are still empty but many tests prove that OpenCL can be as comparable as CUDA but rarely better[KDH10, FVS11].

In this study most of the code had been developped using CUDA to have the best benefit of the NVIDIA GPUs present in the ROMEO Supercomputer. Also the long time partnership of the University of Reims Champagne-Ardenne and NVIDIA since 2003 allows us to exchange directly with the support and NVIDIA developpers.

3.2.5 OpenACC

Open ACCelerators is a "user-driven directive-based performance-portable parallel programming model"[7] developped with Cray, AMD, NVIDIA, etc. This programming model propose, in a similar way to OpenMP, pragmas to define the loop parallelism and the device behavior. As the device memory is separated specific pragmas are use to define the memory movements. Research works[WSTaM12] tend to show that OpenACC performances are good regarding the time spend in the implementation itself compare to fine grain CUDA or OpenCL approaches. The little lack of performances can also be explaim by the current contribution to companies in the wrapper for their architectures and devices.

The runtimes, libraries, frameworks and APIs are summarized in Fig.3.1 They are used in combination. The usual one is MPI for distribution, OpenMP and CUDA to target processors and GPUs.

3.3 Profiling tools

Est-ce interessant? Plus tard? Ne pas en parler?

[7] https://www.openacc.org/

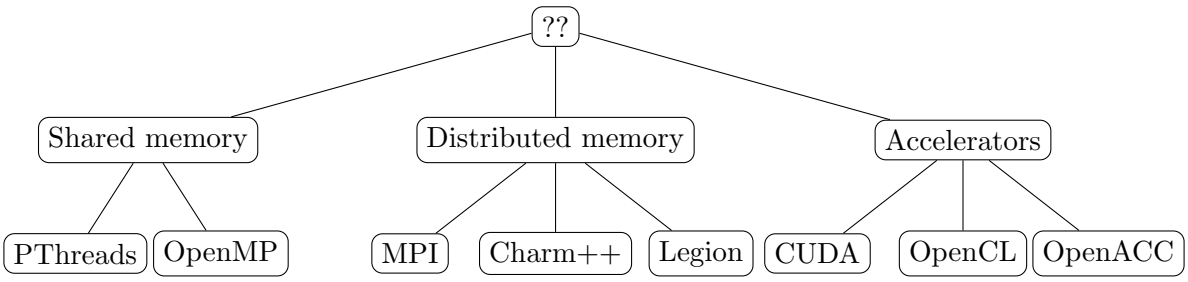

Figure 3.1: Runtimes, libraries, frameworks or APIs

3.3.1 Intel suite

3.3.2 MAQAO

3.3.3 Allinea/MAP

3.4 Benchmark

This section regroup a bunch of the most famous nowadays benchmarks for HPC.

3.4.1 TOP500

The most famous benchmark is certainly the TOP500[8]. It gives the ranking of the 500 most powerful, known, supercomputers of the world as its name indicates. Since 1993 the organization assembles and maintains this list updated twice a year in June and November.

This benchmark is based on the LINPACK[DMS+94] a benchmark introduced by Jack J. Dongarra. This benchmark rely on solving dense system of linear equations. As specified in this document this benchmark is just one of the tools to define the performance of a supercomputer. It reflects "the performance of a dedicated system for solving a dense system of linear equations". This kind of benchmark is very regular in computation giving high results for FLOPS.

3.4.2 GREEN500

In conjunction of the TOP500, the GREEN500 focus on the energy consumption of supercomputers. The scale is based on FLOPS per watts [FC07]. Indeed the energy wall is the main limitation for next generation and exascale supercomputers. In the last list, November 2017, the TOP3 machines are accelerated with PEZY-SC many-core devices. The TOP20 supercomputers are all equiped with many-cores architectures: 5 with PEZY-SC, 14 with NVIDIA P100 and 1 with the Sunway many-core devices. This show clearly that the nowadays energy efficient solutions resides in many-core architecture and more than that, hybrid supercomputers.

3.4.3 GRAPH500

The GRAPH500 benchmark focus on irregular memory accesses, and communications. It will be detailed in Part. II Chapter II in our benchmark suite. Etoffer

3.5 Conclusion

In this chapter we presented the most used software tools for HPC.

[8] http://www.top500.org

Bibliography

[Cha08] Barbara Chapman. *Using OpenMP : portable shared memory parallel programming.* MIT Press, Cambridge, Mass, 2008.

[DMS+94] Jack J Dongarra, Hans W Meuer, Erich Strohmaier, et al. Top500 supercomputer sites, 1994.

[FC07] Wu-chun Feng and Kirk Cameron. The green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12), 2007.

[FVS11] Jianbin Fang, Ana Lucia Varbanescu, and Henk Sips. A comprehensive performance comparison of cuda and opencl. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 216–225. IEEE, 2011.

[Gro14] William Gropp. *Using MPI : portable parallel programming with the Message-Passing-Interface.* The MIT Press, Cambridge, MA, 2014.

[Gro15] William Gropp. *Using advanced MPI : modern features of the Message-Passing-Interface.* The MIT Press, Cambridge, MA, 2015.

[JWG+10] Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V Kalé, and Thomas R Quinn. Scaling hierarchical n-body simulations on gpu clusters. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[KDH10] Kamran Karimi, Neil G Dickson, and Firas Hamze. A performance comparison of cuda and opencl. *arXiv preprint arXiv:1005.2581*, 2010.

[Sup17] Bronis Supinski. *Scaling OpenMP for Exascale Performance and Portability : 13th International Workshop on OpenMP, IWOMP 2017, Stony Brook, NY, USA, September 20-22, 2017, Proceedings.* Springer International Publishing, Cham, 2017.

[WSTaM12] Sandra Wienke, Paul Springer, Christian Terboven, and Dieter an Mey. Openacc—first experiences with real-world applications. In *European Conference on Parallel Processing*, pages 859–870. Springer, 2012.