Procedia Computer Science

Volume 80, 2016, Pages 98–107

ICCS 2016. The International Conference on Computational Science

A Performance Characterization of Streaming Computing on Supercomputers

Stefano Markidis[1], Ivy Bo Peng[1], Roman Iakymchuk[1], Erwin Laure[1], Gokcen Kestor[2], and Roberto Gioiosa[2]

[1] Computational Science and Technology Department, KTH Royal Institute of Technology, Stockholm, Sweden
{markidis, bopeng, riakymch, erwinl}@kth.se

[2] Computational Science and Mathematics Division, Pacific Northwest National Laboratory, WA
{Gokcen.Kestor, Roberto.Gioiosa}@pnnl.gov

Abstract

Streaming computing models allow for on-the-fly processing of large data sets. With the increased demand for processing large amount of data in a reasonable period of time, streaming models are more and more used on supercomputers to solve data-intensive problems. Because supercomputers have been mainly used for compute-intensive workload, supercomputer performance metrics focus on the number of floating point operations in time and cannot fully characterize a streaming application performance on supercomputers. We introduce the injection and processing rates as the main metrics to characterize the performance of streaming computing on supercomputers. We analyze the dynamics of these quantities in a modified STREAM benchmark developed atop of an MPI streaming library in a series of different configurations. We show that after a brief transient the injection and processing rates converge to sustained rates. We also demonstrate that streaming computing performance strongly depends on the number of connections between data producers and consumers and on the processing task granularity.

Keywords: High-Performance Computing, Streaming Computing, Data-Driven Applications, Big Data

1 Introduction

Supercomputers consist of computing units and associated memories on computing nodes that are connected via an high performance interconnection network. These networks are characterized by small latency in the microsecond range and by a large bandwidth of the order of GB/s [3]. The dominant programming system to implement communication across the network is MPI [6] that provides a set of subroutines to explicitly send and receive data from one

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.301

process to another. Traditionally, supercomputers and MPI have been used for solving Partial Differential Equations (PDE). Examples of these equations are Navier-Stokes equations for Computational Fluid Dynamics, Maxwell equations for electromagnetic problems, Schrödinger equation for Quantum Mechanics problems. Typically, these equations are discretized in space on a computational grid and advanced in time with several computational steps. The computational workload is divided among processes associated to a specific computing unit. Each process computes the calculations relative to a small part of the grid. Interconnection communication is needed to move data from processes mapped to different computing nodes. The communication occurs regularly at each computational cycle, and it is typically coarse-grained as a relatively large amount of data on ghost cells are communicated among processes. In these problems, the time taken for communication is typically a fraction of the computation cost. For this reason, these problems are referred as *compute-intensive*. Because supercomputers are mainly used for solving compute-intensive problems, the widely used metric to characterize an application performance is the number of floating point operations per second (FLOPS). This quantity is measured using the High-Performance Linpack (HPL) benchmark [11]. More recently the High Performance Conjugate Gradient (HPCG) benchmark has been used [5]. Both the HPL and HPCG benchmarks measure FLOPS to solve a linear system.

Lately, supercomputers are more and more used for solving a different class of problems that require the analysis and classification of large amount of data. The reason for this is that conventional computing resource cannot support the analysis of such large amount of data in a reasonable amount of time (*Big Data*). In this class of problems, the number of data movements across memory spaces and interconnection network is much larger than the number of floating point operations. These problems are also called *data-intensive*. A typical example of these problems in the scientific domain is the analysis of the results from large experiments, such as the Large Hadron Collider (LHC) and the future Square Kilometer Array (SKA). For instance, the LHC facility stores 30 PB data per year. These data is analyzed to identify presence of the Higgs boson and dark matter in the LHC data. As a second example, the SKA antennas will provide a data rate in the order of 10 PB/s for radioastronomy studies [2]. Additional examples of data-intensive problems include the analysis of data from spacecrafts for climate studies, from seismology centers for earthquake observation and prediction, and from Next Generation Sequencers (NGS) for bioinformatics research.

Data-intensive problems are typically formulated as a pipeline of several tasks that act on a input data set to provide an output data set. For instance, the SKA task pipeline will include signal processing, handling correlation and beam-forming, image generation and calibration [2]. Such a pipeline can be conveniently expressed with the streaming computing model. A stream is an irregular data flow. Streams link different tasks of the pipeline of the data-intensive application by providing input and output of the tasks. Communication in streaming systems is irregular and fine-grained as stream communication units are typically small in size. The main feature of streaming computing is that the results of each computational task are calculated on-the-fly. The streaming model requires a single-pass analysis: stream elements (SE) are consumable so that each stream element is only accessed and processed once. For this reason, streaming computing allows for processing on-the-fly large data sets that do not fit to the memory of the computing system without requiring the use of a batch system or intermediate data storage. Because data is processed on-the-fly and computation must be completed before disregarding the input data, tasks of the pipeline are strongly coupled and the data injection and the processing capability are deeply correlated in the streaming system. Popular streaming frameworks, such as Spark [12] and Flink [7], support applications written in Java, Python, Scala but they are not specifically designed for running streaming computing on Supercomputer with