Displaying the Power of Heterogeneous Computing on GPUs

Emily Craig, Carina Salcedo, Elizabeth Wang
Lawrence Livermore National Laboratory

Goal

Demonstrate speedups gained by incorporating a GPU in a way that can be used for demonstrations to the general public

Introduction

Central Processing Unit (CPU) – standard processor used in computers

- Designed to perform a small number of tasks at a time as quickly as possible
- Reaching a limit in performance speed due to power and heat limitations

Graphics Processing Unit (GPU) – massively parallel processor developed for rendering graphics

- Designed to perform a vast number of tasks concurrently
- Now being used in conjunction with CPUs to accelerate many types of computationally expensive tasks

We chose to compare the speed of computation with and without a GPU accelerator for three applications in different fields of science

Applications- LAMMPS

Large-scale Atomic/Molecular Massively Parallel Simulator, a molecular dynamics simulation

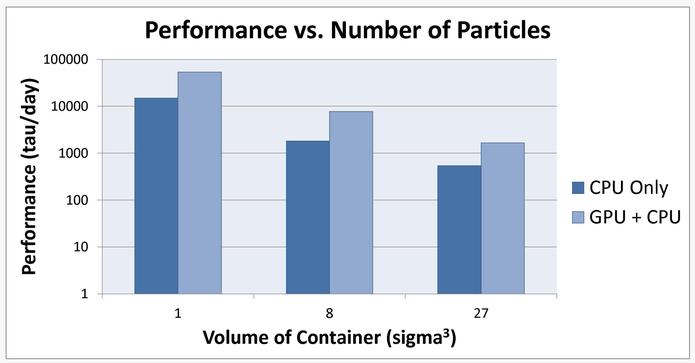
- Can be used to model atoms or simulate particle movement
- Generates a fixed number of particles inside a cube and animates them, tracking where they are and calculating their new positions
- Used Lennard-Jones simulation

Challenges

- Compiling application for the GPU
- NVIDIA Jetson Board is a non-standard hardware and software environment

Results

- The GPU ran 3 to 4.3 times faster than the CPU alone
- Drastic decrease in performance speed due to very limited memory
- External memory limitations caused crash when container dimensions increased to four



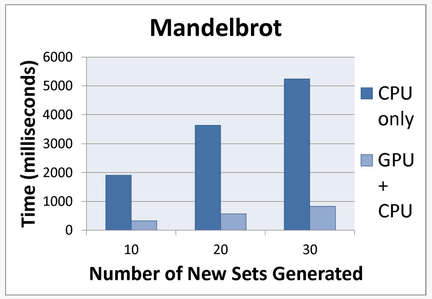
Applications – Mandelbrot Set

Set of complex numbers c for which the function $z_{n+1}=z_n^2+c$ does not diverge when iterated from $z_0=0$

- Coloration based on divergence rate lends itself to visualization
- Prebuilt demo on our Jetson Boards with a visual simulation

Results

- In each run, the GPU was about 6 times faster than the CPU



Applications – John the Ripper

Password cracking program

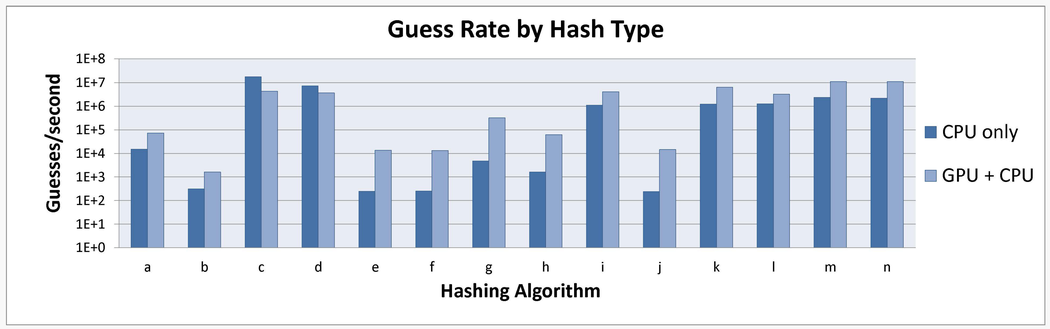
- Recognizes a variety of different types of hashing algorithms
- First uses a word list to guess, then guesses by brute force

Challenges

- Better support for OpenCL than CUDA, but our GPU was incompatible with OpenCL
- Supports cracking fewer hashing algorithms with GPU than on CPU alone

Results

- For 12/14 hashing algorithms, the GPU code was at least twice as fast as the CPU code
  - In four cases, the GPU was more than 50 times faster
- For the remaining two hashing algorithms, the CPU was 2-4 times faster than the GPU



Next Steps

- Install and tune GUIs for LAMMPS and John the Ripper
- Design and implement a GUI that incorporates all three applications

Acknowledgements: David Fox, Joseph Han, Pam Hamilton

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-698611

Proof-of-Concept for Heterogeneous GPU Computing:
Lattice-Boltzmann Fluid Simulations with Image Capture Analysis for User-drawn Boundary Conditions

Andrew Dawson*, Shingo Lavine+, Xavier Quinn‡

*: Rensselaer Polytechnic Institute (Troy, NY) +: Brown University (Providence, RI) ‡: Union College (Schenectady, NY)

Abstract

This project demonstrates a Lattice-Boltzmann Method fluid simulation running on a heterogeneous GPU architecture using captured graphical input from the user. Components for the software stack were identified and deployed on the heterogeneous processor, and python scripts are used to translate information from image analysis to simulation.

Introduction

What is Heterogeneous Computing?

- Utilizing more than one kind of processor, e.g. CPU and GPU, to solve a problem
- Can have significant performance and power usage advantages over homogeneous systems

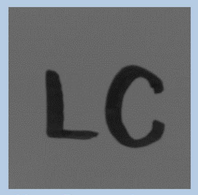
The new Sierra system to be deployed at LLNL as part of the CORAL procurement is an example of a heterogeneous system where IBM Power processors are coupled with Nvidia GPU accelerators. The objective of this project is to explore a heterogeneous system while creating an interesting interactive simulation. The data flow involves the user drawing a shape while an edge-detection algorithm interprets the drawing to define boundary conditions for a fluid simulation.

Methods and Results

The Jetson TX1 device from Nvidia combines a multi-core ARM CPU with 256 CUDA-capable Maxwell GPU cores. While the board is commonly used for deep learning and computer vision embedded applications such as autonomous vehicles, here the platform is being used as a proxy for a cluster compute node. The Jetson TX1 board was able to support the LBM simulation along with the image capture analysis. Multiple boards were clustered allowing problems to be solved with a distributed parallel algorithm leading to computational speedups.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Simulation Workflow



User Drawn Image

Image Capture Input

Gstreamer[1], a framework for creating various media-based applications, is used to capture an image.

OpenCV Edge Detected Image

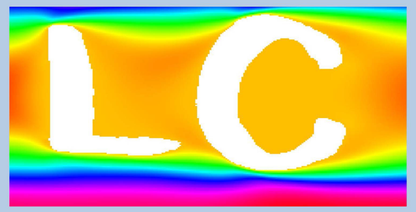
Edge Detection

Algorithms within OpenCV[2], an open-source computer vision and image recognition library, identify edges within the image

Python Script Cropped Image and Filled In

Image Processing Script

A python script translates the edges generated by OpenCV into boundary conditions readable by sailfish. The python script enters the boundary values and executes the sailfish simulation.



Sailfish Fluid Simulation

Fluid Simulation

Sailfish[3] solves the fluid dynamics equation using a Lattice-Boltzmann Method (LBM). In contrast to more traditional methods (e.g. finite volume), LBM could be more parallelizable on GPU architectures.

Jetson TX1

User Iteration

Discussion

One motivation for this project was to create an environment in which a user could quickly demonstrate fluid flow over rudimentary airfoils. The program allows for simulation flexibility by allowing the user to draw or photograph real objects and could theoretically be expanded to allow for 3D spatial capture as input for simulations. Using image capture input enables users to demonstrate a proof-of-concept prototype with minimal programming.

Furthermore, this project demonstrates the viability of using LBM physics simulations on a CPU/GPU architecture. Future high performance computing resources will have heterogeneous architectures and LBM algorithms may enable better utilization of the capabilities.

Next Steps:

- CPU vs GPU/CPU Performance Benchmarking
- Developing a GUI
- Clustering additional nodes
- Strong and weak scaling studies
- 3D Spatial Capture

Acknowledgements:

Joseph Han (Mentor), Thomas Bennett, Ian Lee, David Fox, Pam Hamilton, Kim Cupps, Giuseppe Di Natale: Lawrence Livermore National Laboratory (Livermore Computing)

References:

1: https://gstreamer.freedesktop.org/
2: http://opencv.org/
3: http://dx.doi.org/10.1016/j.cpc.2014.04.018

Review and Release Number:

Monitoring HPC Clusters Using Sqrl

Delaney Gill-Sommerhauser, Danielle Larson, Christopher Moussa

Lawrence Livermore National Laboratory

Abstract

Sqrl is a computer inventory application written in Golang that gathers and exports extensive system information to an outside source for analysis and visualization of the data. The purpose of this project is to provide accessible/scalable data to users who are interested in finding out information about their Linux systems in order to better understand efficient utilization of available nodes in a cluster.

Introduction

In large clusters of computers, it can be difficult to extract information concerning the physical node; however, this process can be automated through a program. Previously existing packages provided select information about the node, such as network name, number of cores, free and used memory, hard drive type, kernel version, and type of operating system, but none contained exhaustive system specifications. Our goal was to create a single program that provided all of this information, called sqrl. We aim to research the possibilities of implementation of sqrl in large-scale systems to find the most efficient and practical way to monitor systems.

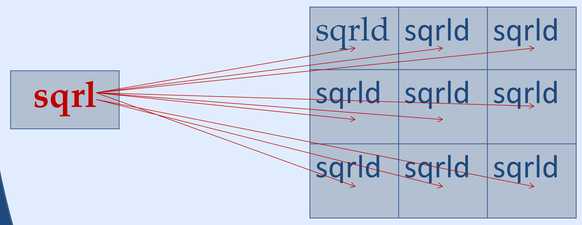


Figure 1: sqrl can be easily run on all nodes of a cluster with one command. With much larger clusters having thousands of nodes, this program alleviates the need to address each node individually.

Methods

We leveraged previously existing Go packages in order to create sqrl. Our completed application allows the user to customize what is logged and exported.

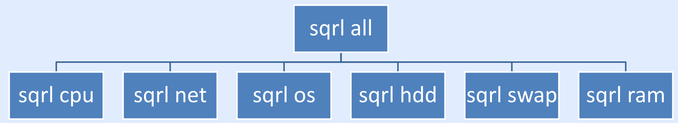
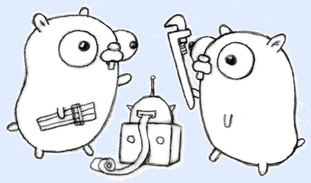


Figure 2: The commands and subcommands of sqrl that allow the user to customize logs and output

We looked into different ways sqrl could be automated and scheduled to run:

- Cron (through the operating system)
- Internal task scheduling
- HTTP server

After exploring pros and cons of each process, we decided on a combination of both internal task scheduling and an HTTP server.



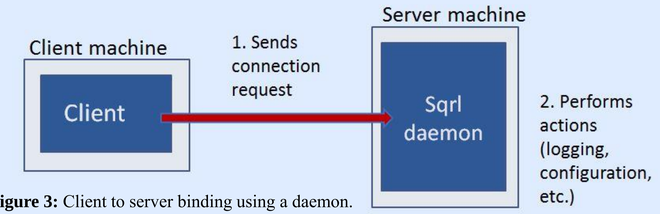


Figure 3: Client to server binding using a daemon.

Why We Chose It

Internal task scheduling and an HTTP server provide many benefits for continuously and efficiently logging system information. While our internal scheduler will update and log information on a set time (versus the operating system's own cron scheduler), using an HTTP server provides the ability to run sqrl at any time. Sqrl, when running in daemon mode, allows the application to accept requests that could update the frequency of logging and verbosity of information. It also provides scalability; whether there are five machines or 5,000, each individual machine logs its own information, allowing an application like Splunk to collect and visualize that information. In case of an emergency, the HTTP server in sqrl allows the user to make configuration changes to all of the nodes in a cluster.

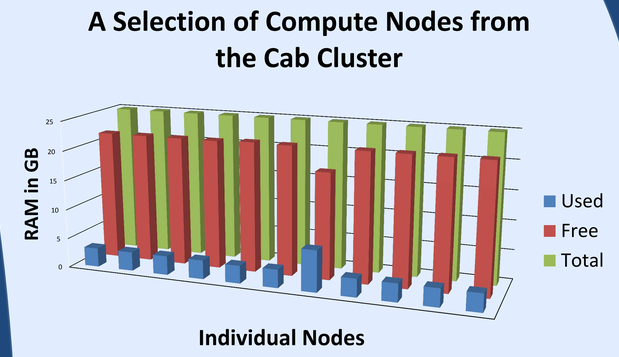


Figure 4: Graph of memory usage on a selection of nodes in the Cab cluster.

Discussion

The data that sqrl provides can be used in many different applications. For example, sqrl could provide the data needed to recognize correlations between running jobs and memory usage, allowing users to better utilize system resources. Depending on the memory requirement of a particular job, sqrl's exported data could be evaluated using analytic tools to estimate the most efficient node-to-memory distribution for certain jobs. Because sqrl is automatically run, it is easily accessible and provided in real-time to users.

While the combination of an HTTP server and daemon is optimal, there are still some downfalls to this method. The downside to the HTTP server is its lack of security; as of now, anyone can access the server if needed. Consistent reliability of the network also raises another concern because users would not be able to access all of the machines through a management node if a network connection is broken.

To extend this project, a point to address would be security. For example, adding authentication would ensure possession of how sqrl runs is limited to a select number of privileged users. Another area for extension would be logging dynamic information each time the program is run, while the static information, such as operating system info, is only logged per specific request, removes unnecessary logs of information.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Acknowledgements:
Derek McQuay
Pam Hamilton
Livermore Computing

Tory: A Computer Inventory Script for HPC

M. Abdalla, K. Johnston, J. Walker

Objective:

To develop a script that gathers basic system information about a Linux based cluster and output results in various formats.

```
$ python tory -h
usage: PROG [OPTION] CMD…

optional arguments:
optional arguments:
  -h, --help            show this help message and exit
  -m MAX_RECORDS, --max-records MAX_RECORDS
                        Maximum number of records to show
  -j, --json            convert to json
  -r, --read            convert to human readable
  -d, --database        send to database
  -s, --show_database   show contents in database

commands:
  CMD
    disks               Get info on disks
    network             Get info on network
    cpu                 Get info on CPU
    ram                 Get info on RAM
    user                Get info on users
    simple              Get info on simple
    package             Get list of packages or search for
one package
```

Why this is Cool?

On large systems, such as Sequoia, parts are continuously being replaced and software is constantly being added or updated due to the needs of users. This project will produce a fast and easy way to monitor these changes. Tory will give us an actual count of hardware or software.

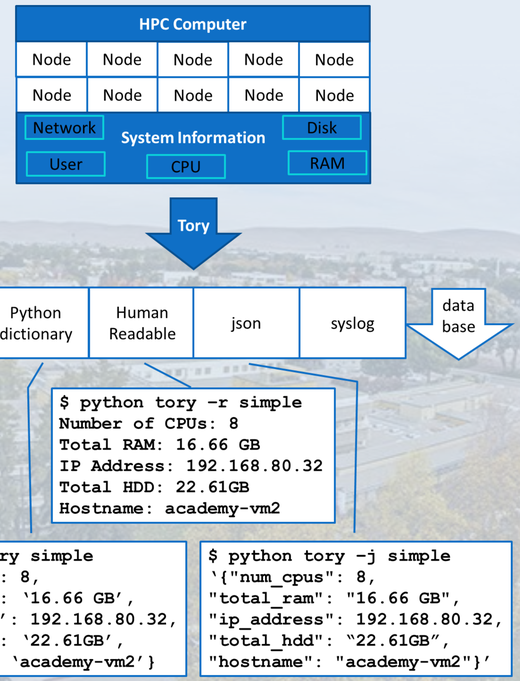
Method:

We used python modules, such as psutil, and Linux system calls to gather system information, such as network, CPU, and RAM. We then stored or displayed results in human readable format, json, and others. We wrapped up our project using the “Package Manager by Python”, pip, to package Tory and make it available for download on PyPi and the open source channel GitHub. We made Tory an interactive search tool, where users can query a specific target or request a list of items.

Links:

GitHub: https://github.com/tbenz9/tory

PyPi: https://pypi.python.org/pypi/tory/0.1



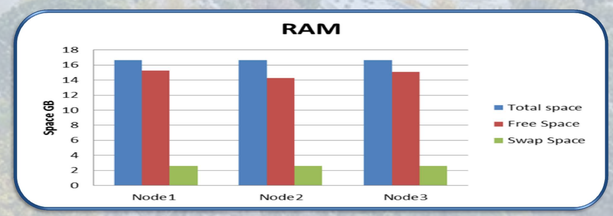
Tools:

python · CentOS · github SOCIAL CODING · SQLite · SPHINX PYTHON DOCUMENTATION GENERATOR · Flask web development, one drop at a time

Expected deliverable:

There is still much work for Tory, such as:

- Graphic displays of information.
- Alerts for mapping inventory scripts.
- Deploying Tory on a large scale, HPC computers.
- Make a robust Tory, that can handle any unexpected situation.
- Insure that our code follows Python Pep 8 style guide.
- Create documentation and man page.



Example of visual output for RAM information

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Lawrence Livermore National Laboratory