Yahya Alhinai

EE5371

October 28th, 2019

Problem 1:

The mean fundamental difference between Gustafson's approach and Amdahl's approach is the dependency of p, the amount of time spent on part of program that can be fully partitioned and done in parallel, over N, the number of possessors that are executing in parallel. Amdahl's approach assumes p is fully independent to N which is done by having a fixed sized problem and run various numbers of processors on it. Where Gustafson's approach identifies this issue and scale up the sized of the problem lineally with the number of processors with a scale up relationship of 1 to 1.

The paper *"Extending Amdahl's Law for the Cloud Computing Era"* has invoked a new argument to Amdahl's Law to explain a recent productive phase in the world of computing which is cloud computing. The original intention of Amdahl's Law is to explain the enhanced execution time that is done by multi-processor over single-process competing. In Díaz-del-Río's paper, however, he extends the arguments presented by Amdahl to cover cloud computing with some adjustment to it.

Amdahl's Law is even more relevant in the cloud computing era because the fact that cloud resources utilized the fundamental multi-processor properties, but with a feature to scaled up the number of virtual CPUs dynamically making it as large as desired to be. Adjusting the argument presented in Díaz-del-Río's paper to account for the communication delays.

The benefit of Amdahl's approach is to note the real speedup of a multi-process CPU over single- process CPU executing a certain program. The benefit of Gustafson's approach is account for the achieve efficient of parallel performance of the overall system. The benefit of Díaz-del-Río's approach is that in addition of the speedup of multi-processor, it accounts for the communication delays between local device and the cloud.

Problem 2:

The two systems were compared:

1. **System 1:** XPS 15 9570:

   Processor: Intel® Core i7-8750H

   Number of Cores: 6

   Base speed: 2.21 GHz

   Max speed: 4.10 GHz

   RAM: 32.0 GB

   L1, L2 and L3 caches: 384 KB, 1536 KB, and 9216 KB

2. **System 2**: CSE Labs UNIX KH4250-04:

   Processor: Intel Core i7-4790

   Number of Cores: 8

   Base speed: 3.60GHz

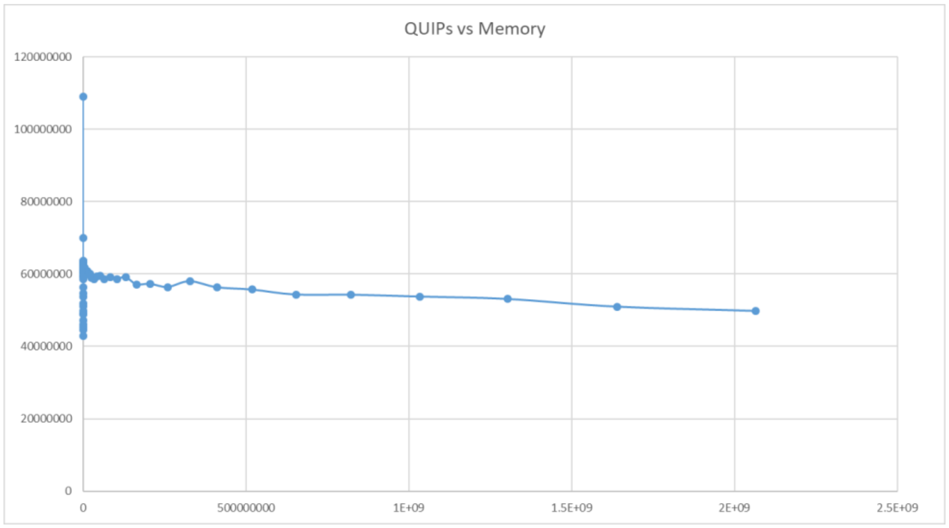
   Max speed: 4.00 GHz

   RAM: 32.0 GB

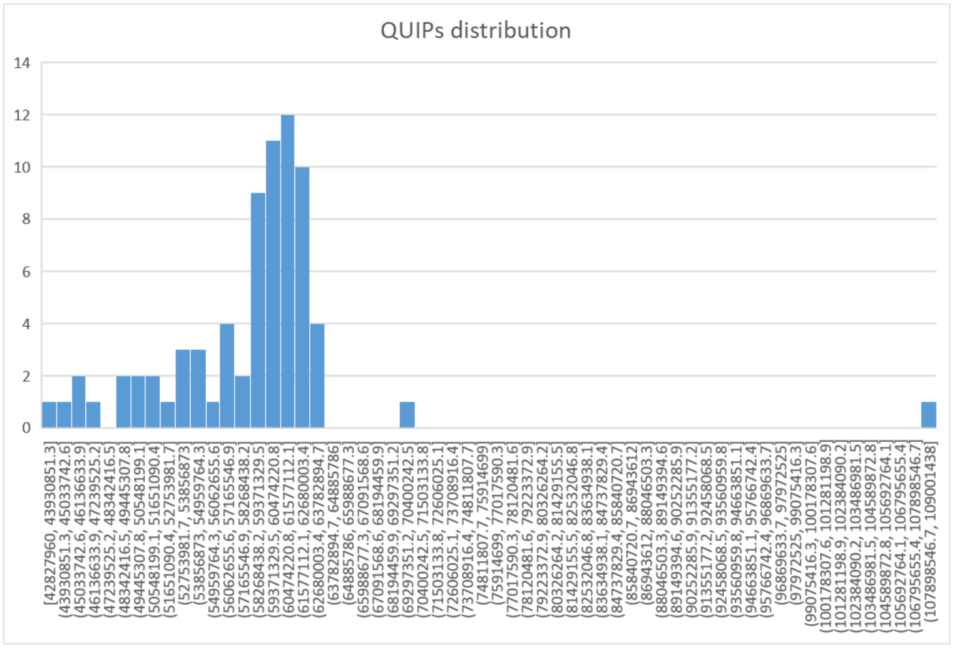
   L1, L2 and L3 caches: 32 KB, 256 KB, and 8192 KB

HINT testbench unitizes two different functionalities in running the program which are arithmetic computation and store in memory of each upper and lower bounding rectangles. This operation HINT is a replication of many real-world applications than referred to as steady progress toward improved quality application.

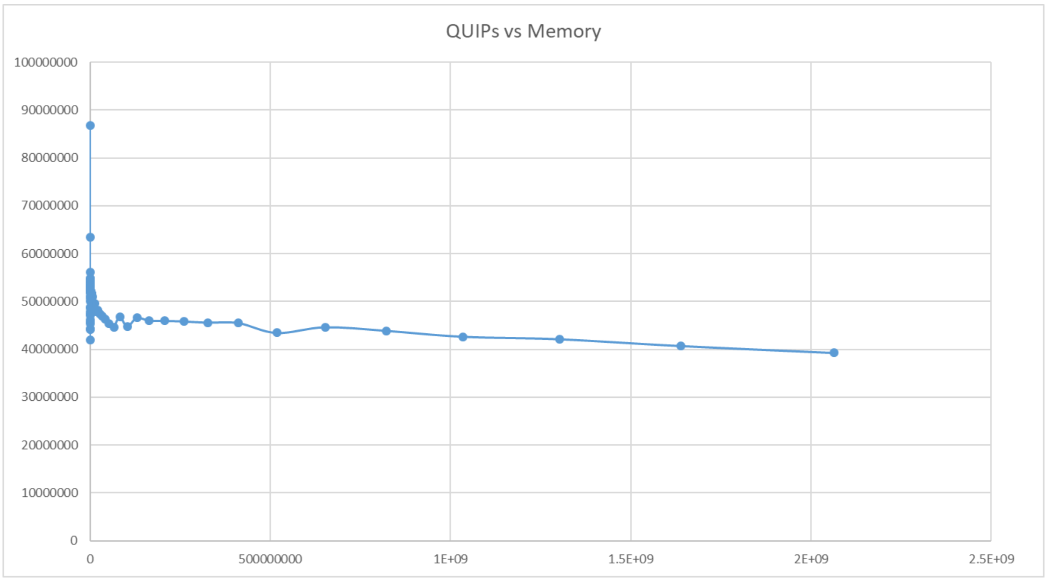
The results of running HINT for <u>system 1</u> report a net QUIPs of **900707006.3.** The following graph shows QUIPS vs. memory that is useful in reveal memory regime sizes:



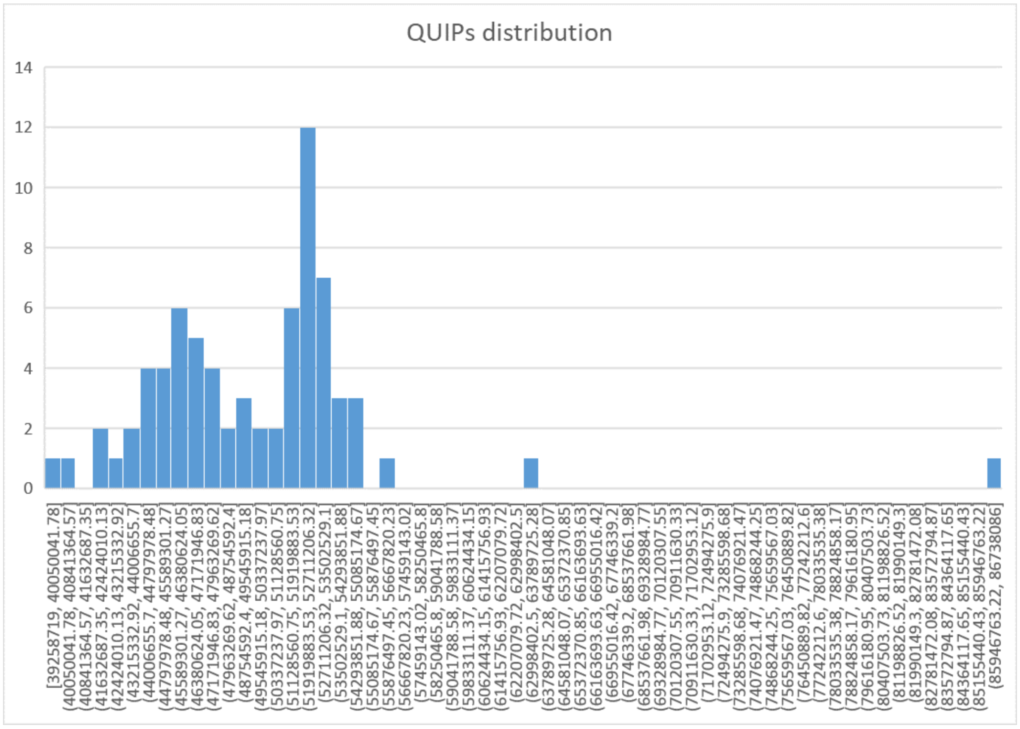
QUIPs distribution is presenter in the following figure:



The results of running HINT for <u>system 2</u> report a net QUIPs of **775425801.3.** The following graph shows QUIPS vs. memory that is useful in reveal memory regime sizes:



QUIPs distribution is presenter in the following figure:



The difference in the net QUIPs of the two systems is vastly wide. System 1 has **1.16 times** higher net QUIPs than system 2. Which means, according to HINT testbench, that system 1 has about **1.16 times** better performance than system 2.

The superiority of the performance in system 1 is mainly contributed to the faster max clock speed at 4.1GHz. Which is 1.025 times faster than the base clock speed for system 2 at 4.00GHz. The max clock speed was compared because we the testbench runs, it locks at max speed of the CPU's core. The CPU clock speed is not the only factor that makes system 1 has almost twice as much better performance than system 2, the memory has some conurbation to performance as the testbench need to store in memory of each upper and lower bounding rectangles. System 1 has larger L1, L2, and L3 caches than system 1 by 12 times, 6 times, and 1.125 times respectively.

Using 95% confidence interval for system 1 and system 2, we find:

<u>System 1:</u>

$$mean = 49834934$$

$$standard\ deviation = 6064497$$

$$95\%\ confidence\ interval\ rang = x_{mean} \pm z_{0.975} * \frac{S}{\sqrt{n}} = [48432017, 51237851]$$

<u>System 1:</u>

$$mean = 58559997$$

$$standard\ deviation = 7909776$$

$$95\%\ confidence\ interval\ rang = x_{mean} \pm z_{0.975} * \frac{S}{\sqrt{n}} = [56730206, 60389788]$$

<u>Comparing system 1 to system 2:</u>

$$mean\ of\ the\ diffreence = 8725062$$

$$standard\ deviation = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} = 1158649$$

$$95\%\ confidence\ interval = d_{mean} \pm z_{0.975} * S_x = [6419350, 11030775]$$

We can say with 95% confidant that system 1 is statistically significant than system 2.

As described in the paper "Producing Wrong Data Without Doing Anything Obviously Wrong!", This section is meant to test any external or orthogonal change to the program. For instance, the paper demonstrates how changing the size of an unused environment variable could lead to 30% to 300% change in the performance.

In our case, the order of compiling HINT testbench files were tested to see if this external change has any real effect on the overall performance of HINT. The three variations of compilations are:

Original compilation code:

```
hint: hint.o hkernel.o Makefile
        $(CC) hint.o hkernel.o -o hint $(LFLAGS)

hint.o: $(DRIV_SRC) $(INCLUDES) Makefile
        $(CC) $(DRIV_CFLAGS) -DIINT $(DRIV_SRC) -o hint.o

hkernel.o: $(KERN_SRC) $(INCLUDES) Makefile
        $(CC) $(KERN_CFLAGS) -DIINT $(KERN_SRC) -o hkernel.o
```

Tested compilation code:

```
hint: hkernel.o hint.o Makefile
        $(CC) hint.o hkernel.o -o hint $(LFLAGS)

hint.o: $(DRIV_SRC) $(INCLUDES) Makefile
        $(CC) $(DRIV_CFLAGS) -DIINT $(DRIV_SRC) -o hint.o

hkernel.o: $(KERN_SRC) $(INCLUDES) Makefile
        $(CC) $(KERN_CFLAGS) -DIINT $(KERN_SRC) -o hkernel.o
```

After changing the order of compiling the code that makes HINT library, another HINT test was conducted to observe any changes. The result shows no significant changes in the performance. The change in performance before and after the rearrange compilation file were less than 0.07%. Meaning the only changes in performance were due to the expected system errors.