Impact of Resource and Parameter Scaling on HPCG Throughput

ADRIAN ABEYTA, University of New Mexico, USA
ROGER DAVIS, University of New Mexico, USA

In High Performance Computing (HPC), it is beneficial to demonstrate both the capabilities of systems computer scientists work with and understanding internal processes of these systems. A means through which this can be done is standardized benchmark testing programs. For this project, our team examined the Hopper server of the University of New Mexico's (UNM) Center for Advanced Research Computing (CARC) using the throughput of the High Performance Conjugate Gradients (HPCG) Benchmark measured by the final rating of billions of floating point operations (Gflops) over many runs. To achieve this, we executed the HPCG program while changing the problem size, time ran, and resources used independently to better understand the impact these aspects had on the performance of the system. We expected that the assistance of Graphics Processing Units (GPUs) yielded drastic increases in performance, the scale of the problem to increase throughput to a point before causing a drastic decrease, and run time length to decrease throughput rating. We also expanded our own familiarity with and grasp of bash scripts, Slurm job scheduling, and modern HPC system environments.

ACM Reference Format:
Adrian Abeyta and Roger Davis. 2024. Impact of Resource and Parameter Scaling on HPCG Throughput. 1, 1 (May 2024), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

INTRODUCTION

The primary focus of our study was to approximately demonstrate the strong and weak scaling factors described by Ahmdal and Gustafson, respectively. Gene Ahmdal argues that the serial portion of a program's code dictates the limit to which optimal performance can be achieved in a highly parallel computing environment. This inherently makes sense; as more processes are sharing the same work load, the serial portion dominates the overall performance as useful work is spread thinner amongst them. John Gustafson, on the other hand, says that the problem size should increase linearly with the number of processors in order to keep the amount of work done by each one equal. This should provide for the highest total throughput by all parallel processes.

We achieved this goal by following the constraints set out by these laws. Using CARC's Hopper cluster, we made several experimental runs for fixed problem size and varying Central Processing Unit (CPU) counts, as well as increasing problem size dimensions directly with the amount of CPUs. This gave us a clear view of not only how this cluster performs in relation to input variables, but also an idea of the proportion of serial code to parallel. As our results will show, CPU and problem size did have an effect on the throughput values using the HPCG software tool. Utilizing GPUs also had a marked effect on maximum throughput at nearly all problem sizes. This was found in our third and final experiment in which CPU-only and GPU-accelerated versions of the code were run using the same range of problem sizes. In this, we compare the performance in terms of Gustafson's scaling, which revealed to us the difference between GPU and CPU architecture when running parallel code.

We would like to thank the UNM Center for Advanced Research Computing, supported in part by the National Science Foundation, for providing the research computing resources used in this work, as well as the knowledge and guidance of both Dr. Matthew Fricke and teaching assistant Ryan Scherbarth.
Authors' addresses: Adrian Abeyta, University of New Mexico, Albuquerque, NM, 87131, USA; Roger Davis, University of New Mexico, Albuquerque, NM, 87131, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Association for Computing Machinery.
XXXX-XXXX/2024/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

METHODS

Our first set of runs tested the impact of the number of available threads on a problem of a fixed size and otherwise fixed resources in order to approximate Ahmdal's scaling. We performed these runs by using `salloc` to request a single node of the debug partition as an interactable environment. We then executed a bash script that called Open Multi-Processing (OMP) commands to change the number of allowable threads then executed the program through `srun` with otherwise unchanging parameters. We used OMP to prevent the HPCG program from scaling up the problem size through its internal process and initially provided dimensions of $16x16x16$ that was run for 60 seconds. We then used the `sed` command to collect the appropriate line from each run's output file and used that data to create our **Ahmdal scaling** 1 graph. We repeated this procedure for a problem size of $32x32x32$, and finally repeated it for the initial problem size but run for 5 minutes.

For Gustafson scaling testing, another simple bash script was used to call `srun` for each number of tasks. In Slurm, this is equivalent to number of processes to launch. Unfortunately, Slurm batch scripts could not be used for this, as "Out of Memory" errors plagued the initial test runs. To get an idea of the best number of OpenMP threads, several values were tested, with 8 being the most optimal. This was used for the remaining scaling sizes for these runs. Different HPCG run time values were also tested, but noticing that increasing time lengths did not appreciably affect the throughput values, a maximum of only 5-minutes was considered. Running on more than 1 node, and with larger amounts of allocated memory also did not provide for better results.

We used `salloc` to request resources again, this time on the condo partition with a single node and GPU, then ran a similar bash script as was used to test thread increase on fixed problem sizes. For this procedure we ran the program for 60 seconds and changed the job size from $8x8x8$ to $288x288x288$ by increasing all three dimensions by 8 for every execution. We arrived at this decision after discovering internal constraints of HPCG that require the problem size to be divisible by 8, and this approach allowed us to examine the problem on a sufficiently large scale to clearly demonstrate the general patterns of performance. At the higher end of our experiments, while only running for 60, the execution

of each iteration took considerably longer as the HPCG program generated the data and eventually resulted in exceeding the time limit of our allocated resources. From that data we constructed the **GPU Scaling** 4 graph.

RESULTS

From the data of our first three runs, we observed erratic performance threads, though not enough to result in overlap, and most importantly a lack of significant improvement as the number of threads increased. This indicated that the HPCG program likely already has some level of optimization of parallelization. Further, the trend of our larger problems executing beyond the 60 seconds of "runtime" indicates that the section of the program that must be serial may be isolated from the section used to determine performance metrics. The variance between equivalent problem sizes with a runtime of five minutes were also insignificant, contributing to our decision to use shorter runtimes for our data collection.

Ahmdal's Scaling on Debug Partition

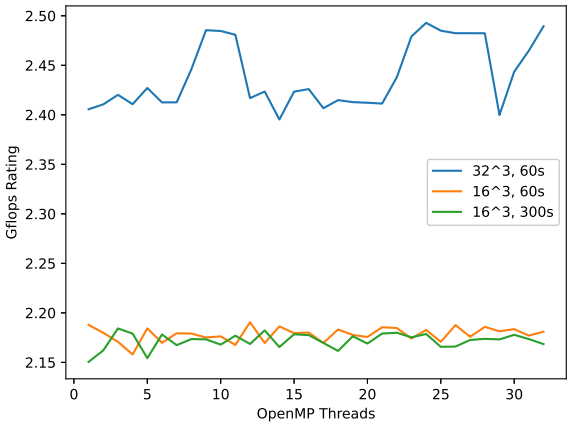
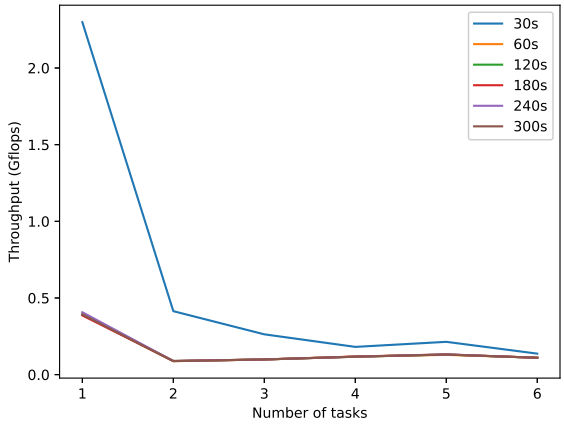


Fig. 1. HPCG run on one node of the debug partition of Hopper for 60 or 300 seconds over 1 through 32 threads. The scale of the irregularities increased with problem size, with difference of best and worst performances being 0.033 for $16^3, 60s$ and 0.098 for $32^3, 60s$

Surprisingly, while testing for scaling of problem size with a constant ratio to CPU allocation, there was actually a major decline in performance as the number of tasks was raised past 1. As mentioned, this may be due to the nature of the HPCG package, which not only tests processor speed, but memory access speeds and cluster communication performance. Also, the number of Slurm tasks is strongly scaled with the problem size by HPCG. As shown by the performance profile in 2, measuring by throughput is not necessarily indicative of Gustafson's law. However, this still shows that there is some upper-limit on performance with constant problem size to process ratio. In a future study given more time, we could thoroughly investigate this phenomenon by rather taking into account the actual run times until the problem is solved. This would give us a clearer idea of how much more work can be completed in parallel.

Gustafson's Scaling on Debug Partition



The range of throughput values with increasing number of tasks (CPUs), with problem size scaled by HPCG automatically, for a range of calculation run times. Obviously, 30 seconds is the optimal time length for this configuration: 8 OpenMP threads and size 32 local domain dimensions on the condo partition.

In contrast to processor scaling, figure 3 shows a clear linear increase in Gflops throughput by increasing the matrix problem size. These data points coincide with an expected approximation of Gustafson's law, although we are not increasing CPU allocation here. The larger problem sizes allow for more optimal utilization of memory resources over time — the CPU has a chance to maximize its throughput by performing more calculations to show its maximum processing power.

The results of our GPU-assisted runs shown in figure 4 clearly demonstrated the advantages of utilizing such hardware in appropriate calculations and at sufficiently large scale. The GPU increased the attainable throughput considerably and while it contained some irregularity, the overall trends were far clearer. During our procedure there were some runs that, while producing no errors, were declared as "INVALID" by the HPCG program. We were not able to determine the cause of that categorization, but have ruled out some potential issues such as the overall size being too large. Those runs have been excluded from our data, as they are unlikely to have deviated from the overall trend observe.

The comparison of these collections of data clearly demonstrates the benefits to overall work accomplished by incorporating GPUs into HPC systems.

While the overall trend of having a GPU being the dominating factor was expected, the details of our CPU only results demonstrate some other interesting trends, including that both procedures peaked at a problem size of $240x240x240$. Using this data we were able to execute a run of the HPCG program that attained a higher throughput than any of our previous tests.

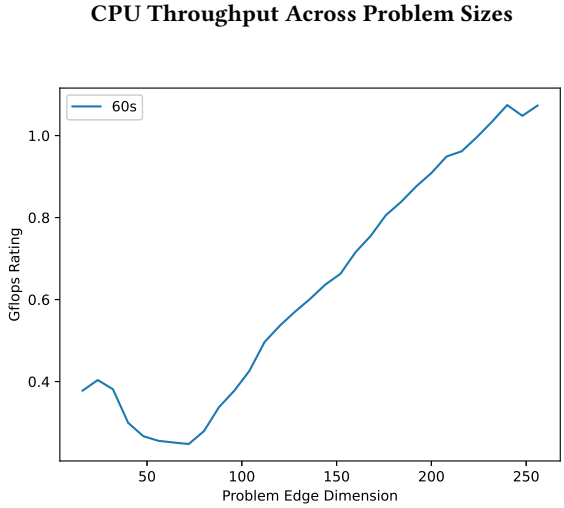


Fig. 3. CPU throughput showing consistent increase overall with a valley near 75 from until a peak at $240x240x240$ of 1.07447 Gflops. Increasing problem size with consistent memory resources had an important effect on throughput performance.

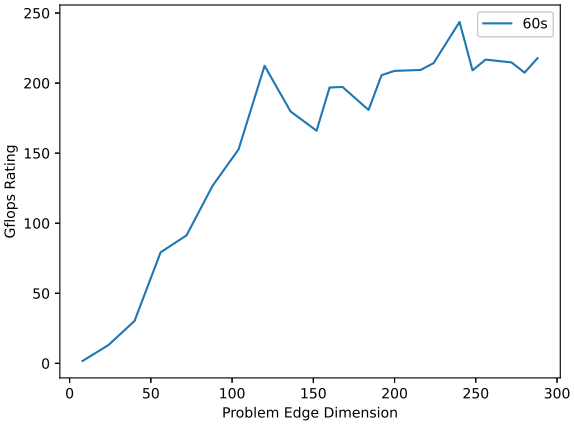


Fig. 4. HPCG run on one node with one GPU as well. Problem size was a cube with edge dimensions ranging from 16 to 288, increased by 8 at each step, excluding runs that returned "INVALID." Gradual increase initially, rapid increase above a problem size of 40^3, plateauing near 120^3, and peaks at 240^3 with 243.63 Gflops.

CONCLUSIONS

The HPCG software package has proven to be useful in estimating the throughput of UNM's Hopper cluster in a representation of real-world computations used by scientists. It was also an interesting tool in attempting to show the maximum throughput of its hardware and communication network.

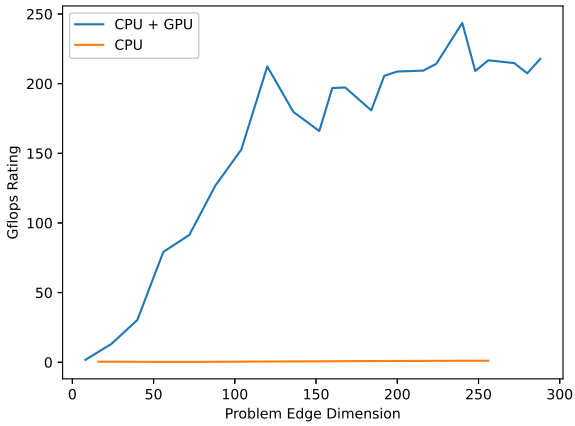


Fig. 5. Throughput of GPU assisted and unassisted executions across problem sizes

Best Throughput results

Resources	Partition	Dimensions	Runtime	Rating
CPU	Condo	$240x240x240$	60s	1.07447
CPU + GPU	Condo	$240x240x240$	60s	243.63
CPU	General	$120x120x120$	60s	6.85628

Although this HPC environment employs nodes with a high number of cores per node, increasing the number of parallel tasks did not have an effect on raw throughput. GPU-accelerated performance, however, did have an advantage which was several orders of magnitude over CPU only calculations. We feel this can be attributed to the highly-parallel environment of a single GPU, as opposed to the limited resources given on a single node. These devices are extremely optimized to perform these very calculations, which shows how indispensable they are for scientists' studies. In addition, since we used highly-customized code provided by the Nvidia corporation's GPU platform, the throughput values were as optimized as possible.

Our scaling study proves that both overall problem size and CPU allocation have varied effects on throughput performance, and it appears to confirm that Gustafson's and Ahmdal's laws are correct with our approximations of them.

AUTHOR CONTRIBUTIONS

{Adrian Abeyta} ran Gustafson scaling HPCG test runs and created plot, as well as runs for CPU-only calculations for comparison with GPU acceleration on varying problem sizes. Wrote introduction and relevant portions of methods and results and report editing and finalization.

{Roger Davis} wrote the code that produced figures 1 and 4 and used the data produced by Adrian Abeyta to generate figures 3 and 5. Additionally, he wrote the abstract and the portions of the methods and results sections concerning Ahmdal scaling and GPU assisted throughput and report editing and finalization.