FINAL REPORT: San Diego State University Student Cluster Competition Team: Experiences and Observations

G. Williams*, G. Behm*, A. Esparza*, V. G. Haka†, T. Nguyen*, A. Ramos*, B. Wright*, J. Otto*, C. Paolini†, M. P. Thom
*Department of Computer Science, San Diego State University
†Department of Computer Engineering, San Diego State University

***Abstract*—The San Diego State University Student Cluster Competition (SCC) team participated in the SCC16 for the first time [1]. The cluster is based on 8 Kennedy Pass processors, for a total of 320 cores. In this competition, the team learned to assemble the cluster from the rack up; perform all administration tasks; MPI parallel programming; and how to profile 5 different HPC applications. The outcome of these activities have enhanced the students experiences of using HPC and motivated them to solve complex problems using advanced HPC skills and facilities. Finally, the team has been invited to contribute to a special issue of Parallel Computing, and have been encouraged to return next year. FROM PROPOSAL: San Diego State University proposes to host a Student Cluster Competition (SCC) team to participate in the SCC program for SC16. The SDSU SCC team will conduct research experiments to examine how well the OpenHPC software stack performs on a cluster based on the Intel Kennedy Pass chipset. Based on 16 Kennedy Pass E5-2699 v4 processors, a cluster with 22 cores each, for a total of 353 cores, 10 TB of SSD storage, and an InfiniBand network was provided by Intel, which is our project sponsor. By applying the OpenHPC software on the cluster, the outcome of our research activities will enhance students experiences of using HPC, motivate them to solve complex problems using advanced HPC skills and facilities, and provide feedback to improve the OpenHPC software stack, which will ultimately benefit the broader HPC community.**

***Index Terms*—HPC, parallel computing.**

I. Introduction

San Diego State University has participated at Supercomputing for many years through papers, posters, workshops, and the Computational Sciences Research Center (CSRC) has hosted a research booth since 2011 [1]. Participating in the Supercomputing Cluster Challenge [2] gives us the opportunity to participate at SC at a much deeper level, to reach out to our undergraduates and provide them with a richer undergraduate experience, to enhance our curricula by incorporating HPC concepts, and to contribute to the field of HPC. The SDSU team has strong backgrounds in computer science, computational science, parallel and HPC computing, and cluster management. Our team is multidisciplinary and multi-institutional, and consists of SDSU undergraduate students from multiple departments (computer science, mathematics, statistics); faculty mentors from SDSU CS, Computational Science, Math, and Biology departments; and external partnerships with the San Diego Supercomputing Center (SDSC), and several vendors, many of whom are members of the Open HPC Community.

II. About the Bit Benders Team

The SDSU SCC team has depth, strength, diversity, and a sense of fun. The name of the team arose from a common love of Futurama, and the connection that we could bend computer bits to our will, hence the name "Bit Benders." Our tag line is "Byte our shiny metal hash" (See http://www.csrc.sdsu.edu/sc/sc16/scc/).

The SCC Team consists of volunteers who have shown the ability to be successful undergraduates students at SDSU. Most have done some type of HPC/parallel computing on their own, or through internships or online courses. Student Team members include: Ali Esparza, Senior (CS); Tuan Nguyen, Senior (CS); Vaughn Ganem Haka, Senior (Eng); Armando Ramos, Sophomore (CS); Gary Williams, Senior (CS), HPL; Briana Wright, Senior (CS).

Faculty mentors: include several SDSU faculty who supported the project through teaching, working with the students on the cluster, or mentoring applications, including: Dr. Mary Thomas (Project PI); Dr. Jose Castillo and Dr. James Otto (CSRC); Dr. Chris Paolini (Engineering); Dr. Robert Edwards, Peter Bartolli, and Steve Price (CS); and Dr. Peter Blomgren (Math).

Vendor Partners: our partners have been committed to working with the team, which helped ensure that the software and hardware used is dependable and useful for the SCC challenge. In particular, Reese Baird, Hercules Randolph, and Karl Schulz (Intel and OpenHPC); and Dr. Pietro Cicotti (San Diego Supercomputer Center).

III. Team Preparation

The SDSU Team will use several methods for preparing for the competition including courses, lectures customized to the SCC activity, hands on training in the lab using the cluster, and application studies.

Student Preparation

The SDSU team will spend time together this Summer working with the students to help them learn about the cluster (building, installing and administering software, including the

OpenHPC software stack); learning about the SCC applications; participating in a series of lectures about parallel programming models needed for the applications. Each faculty mentor has committed to being available for at least one week this Summer to work with the students.

The student work done in Summer and Fall will be done as part of a research course we will set up in the Fall whereby students can get credit and write-up their experiences as an undergraduate thesis. We will also promote this effort by having students present a poster at the CSRC Applied Computational Science and Engineering Student Support (ACSESS)annual meeting, to be held in May, 2017 [4].

Students have the option to take HPC courses at SDSU. Dr. Thomas teaches two graduate level parallel computing classes, and Drs. Shen and Whitney are starting new undergraduate classes in the Fall: HPC Mathematics and Big Data. We intend to add/expand undergraduate courses to include more HPC and parallel computing topics. Dr. Cicotti will bring the team to the San Diego Supercomputing Center for a day of tours and lectures. Both the CSRC and SDSC have invited the team to participate in booth activities at the meeting. Based on our experiences at the meeting, we plan to apply for an NSF REU grant to help fund future student activities and for an XSEDE Campus Champion Fellowship for curriculum development.

Cluster Preparation

Dr. Otto leads the cluster support work at the CSRC, where the machine will be hosted, and he will be available work with Intel and students on the cluster operation. The CSRC has offered to provide a few nodes for the students to work with while waiting for the Intel cluster to arrive, and a lab space where they can have hands-on access to the cluster. Once the Intel cluster arrives, the team will work with an Intel engineer who will travel to SDSU to help build out the cluster.

System Software Preparation

The system software preparation will focus on the OpenHPC software suite. The OpenHPC community has offered to assist our students as they learn to install and deploy the software stack; in addition, the community will provide with updates as needed. Our team mentors will be involved analyzing the SCC applications and teaching the students about them.

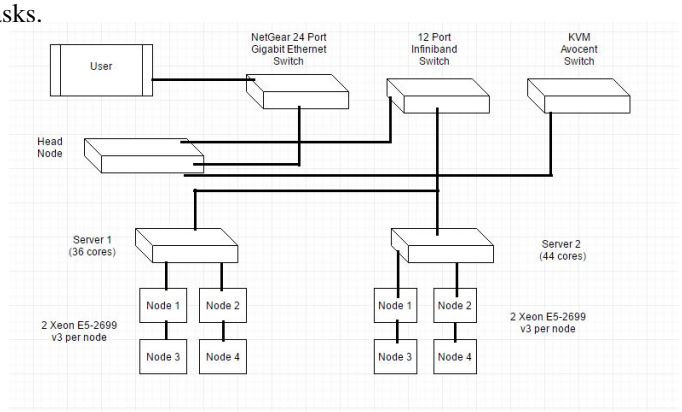
Applications Preparation

Team mentors will work with the team to analyze the SCC applications, teaching them about their design and their applications in real case studies, and working with them as they learn to run the applications on the cluster.

The students will learn about dense linear systems, how HPL is designed, and a little bit of history on top500 and how HPL is used to rank supercomputers. Then, following on the latter theme, they will learn how HPCG came about as a complement to HPL that exercises different components of the architecture; they will also learn how HPCG represents a different set of workloads of great importance for the DoE and the scientific community.

Shifting gears, we will then look at non-FLOPS oriented applications and two very different types of parallelism. On one end, we have the distributed password recovery that is embarrassingly parallel, and on the other end, we have ParConnect, which includes a graph computation and irregular decomposition.

Finally, the students will learn about the importance of visualization in presenting scientific data. In addition, the students will be exposed to the challenges of analyzing and visualizing large data sets, and the importance of designing systems that can accommodate both computing and analysis tasks.



IV. The Nibbler Cluster Architecture

- **Head node:** Intel Xeon DP Broadwell-EP E5-2680 28 core server used for login access, system management, queuing, systems, display monitor, etc.
- **Compute nodes:** Two Kennedy Pass Intel Quad board Servers [2], for a combined total of 320 cores and nominal peak performance of approximately 7-8 TFlops.
- **Networking and Connectivity:** The system uses both GigE and Infiniband networks. The Ethernet network is managed by a 24 port router, and is used as a LAN, for managing the cluster and allowing users to access the system via laptops. The IB interconnection network connects the compute nodes together and is used for computational work.
- **Rack:** all the hardware, including the interconnect fabric, fits into an open sided 42U rack.
- **Performance:** Theoretical peak performance is appx. 11.5 TFlops for 320 cores.
- **Power Usage:** Power benchmark tests have shown that the cluster, when running the HPL benchmark at full capacity, is above the 3.1 KW limit of the competition. We plan to run the benchmark and turn off 1 or 2 nodes as required during the qualification process.

V. Application Strategies

A. Team Preparation

To prepare for the SCC competition, the team met throughout the Summer focusing on key HPC topics, including:

- Introduction to MPI
- Studied one application each week.
- Profiling, scaling, performance monitoring and run-time code profiling (using tools such as gprof, MPI timers, and Allinea DDT).
- OpenHPC software stack - learning which components will work best for each app and why.

In addition to learning about the software, the team helped to build the cluster:

- cluster design and and network interconnection;
- building the rack and installing servers and switches;
- installing memory, SSD drives, IB cards, etc.;
- connecting ethernet and infiniband networks.

We have developed strategies for running and optimizing the applications, including:

- Participate in Saturday lab meetings to promote team discussion and interactions.
- Assign a team lead for each application.
- Working with faculty mentors to choose right compilers and environments;
- Profiling applications for optimization.
- Create an application profile/prediction summaries.
- Develop scripts and analysis tools for run time control and data display including a power usage monitoring script and a script to turn cores on/off for each node.

B. High Performance Linpack (HPL)

HPL is the most widely used benchmark for measuring a system's floating point computing power. It is the measure by which the TOP500 list, a list of the fastest super-computing clusters in the world, is bi-annually composed. HPL works by solving a dense N by N matrix of linear equations of the form $Ax = b$. The result is the total speed of computation given in GFLOPS, or floating point operations per second. The theoretical speed of a system can be computed on a processor by processor basis by multiplying the speed by the specified FLOP/cycle for the processor. The theoretical performance of Nibbler based off of maximum performance test graphs is around 11.5 TFLOPS.

The HPL benchmark requires a fair amount of optimization for any given architecture. The main parameters of interests involve the problem size, the size of partitions within the matrix, and how those partitions are divided up among resources for computation. These parameters are configured in a .dat file. Compilation of the C++ code required additional pointers to the BLAS and MPI libraries. Jobs were submitted through the PBS/Torque scheduler.

Optimization on Nibbler was an iterative process. One of the main limitations discovered early on was the amount of RAM available per node. Some additional RAM was acquired, but the problem size was still a primary limitation due to the amount of data per node. Through testing, the optimal partition size was found to be around the suggested maximum. Also the optimal arrangement for that partition disbursement was found to be the most square as possible given the amount of resources

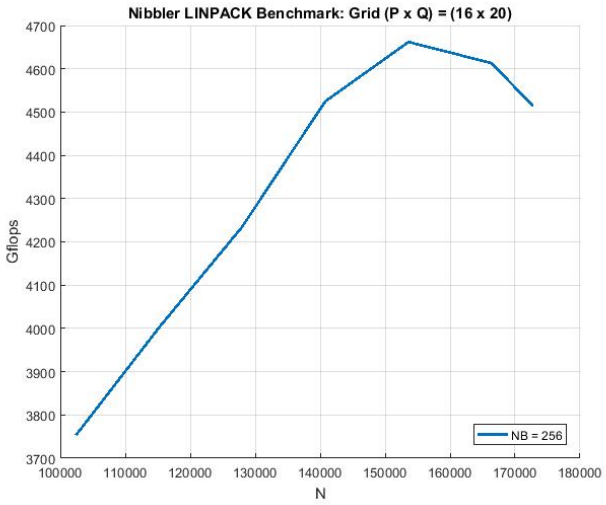


Fig. 1. Speed as a function of problem size across all 320 cores of Nibbler

or processors. These two conditions supported the theory that the best configuration was one that allowed the most local computations before requiring communication to neighboring partitions. Therefore, besides problem size, communication turned out to be a limitation even with the Mellanox Infiniband connections supporting 40Gbps transfer speeds. The graph in Figure 1 is an example of the performance curve seen on Nibbler.

Power also became a main interests of limitation. After Nibbler was first connected we quickly learned the power of HPL, as the first test across all nodes caused a power failure within our laboratory. Power distribution was more carefully considered before testing proceeded. The competition had a limit of 3.12kW, which was further divided to 1560W per monitored Geist PDU. With all 320 nodes running, we easily exceeded our max by more than 1kW. Many ideas for power conservation were theorized and most were tested. However it became apparent that to meet the power requirements, at least two nodes were going to have to be turned off completely. With two nodes off, test were closer to the requirement and it was assumed further power savings would be available from other methods on the cluster as a whole.

Arrival at the competition revealed that the power requirements were not going to be easily met and further changes needed to be made. The unfortunate decision to get rid of yet another node had to be made. RAM was stripped and dispersed among the remaining 5 compute nodes. Our theoretical peak speed with only the 5 nodes was closer to 6.5 TFLOPS. The official test result ended up being around 4.5 TFLOPS. Considering the amount of tinkering that was done to meet power requirements while still trying to take advantage of as many resources as possible, we were satisfied with our final result.

Nearly all of our competitors at SCC used accelerators. With that additional boosts, their HPL results were several orders of magnitude greater than ours. Even if we had achieved

our maximum theoretical peak speed of the complete 320 core system, we still wouldn't have compared to some of our competition. As such, and since scoring was done as a percentage of the highest reported speed, we scored very low on our final scoring for this benchmark.

Going forward it is clear that to be able to compete with the benchmark scores at the Cluster Competition of SCC, the use of accelerators is almost required. A change of architecture is recommended to allow modern accelerators. The additional power consumption would certainly lead to minimizing the compute cores, however, the benefit from fewer connections and the boosts from the accelerators may be the best combination to remain competitive. A fair amount of time was spent trying different parameter combinations for slight improvement of the percentage of theoretical peak speed and much was learned by the team in the process. Considering the competition scoring was not based on the efficiency of or even the best economical system, maximizing speed at any cost should be the concern of future teams.

C. High Performance Conjugate Gradient (HPCG)

HPCG is a benchmarking tool used to gauge the performance of a computer cluster. It works by solving a system of linear equations expressed in the form of a 3-D matrix. These linear equations can express the solution to a differential equation or the solution to a minimization problem. A good example of a minimization problem using conjugate gradient would be trying to find the right geometrical configuration for a molecule which has the minimum amount of inter-nuclear forces between atoms of that molecule.

Once the program finishes running it outputs a set of measurements which measure parameters like maximum GFLOPs, total memory bandwidth, total memory used, run time, and many others. These parameters are helpful for ranking the performance of your machine. So ideally one has to give the program the right inputs in the form of a linear system, in order to push the machine to its limit to truly know its potential. The inputs given to the program are the size of the 3-D dimensions and the runtime of the application. In order to converge onto a solution more quickly the program will run a Gaus Preconditioner. This pre-conditioner will run some pre-analysis on the linear system to reduce the problem size ahead of time.

For the competition, we were required to find the right inputs to get the most accurate ranking of our machine. This involved running a lot of tests on varying numbers of nodes and on many different problem sizes. These problem sizes ranged from 16 by 16 by 16 sized cubes all the way up to 256 by 256 by 256. We would start out by running the program on one processing element then scale it up until we were running it on all 320 processing elements. Since most of the data was in a basic raw format, we also wrote python scripts to visually plot this data and make it easier to analyze.

There was also a big learning curve in learning how all the mathematics worked and diving into the C++ source code to see how it was implemented. Much of the difficulties were in understanding how memory was being allocated within nodes and how it was being communicated to other nodes during execution. After many tests and at times waiting on a non-halting program, our team learned what the machines bottleneck was. Since large cubes were being allocated on nodes which shared a common memory pool, which required tons of inter node communication, the memory bandwidth turned out to be the bottleneck. So we figured out the size that works best is a cube of 104 by 104 by 104. Using this size during the competition, our team was able to get the fairest evaluation of our machine without overloading the machines memory bandwidth.

D. ParaView

ParaView is an open source visualization tool and can be used to analyze very large datasets in parallel. Data can be analyzed in 3D or through the built in Python batch system. ParaView accepts a wide range of data formats and can be used to create animations and take images of manipulated data. The team prepared for ParaView by reading the user guide and practicing the tutorials provided on the ParaView website (www.paraview.org).

The team used mpirun and ParaView's batch system, pvbatch to take screenshots of large datasets over various time steps. At the competition, the task given for ParaView was to take screenshots of 100,000 stars over 2,000 time steps. The data consisted of 2000 .dat files, each with the mass and coordinates of 100,000 stars. First, the .dat files had to be converted to .csv files to be compatible with ParaView. This was accomplished with a Python script that iterated through each of the 2000 files, paired with mpirun and pbsbatch.

Next, these .csv files were loaded into ParaView and converted to .vtk files in order to allow filters to be applied to the data and to generate a 3D representation. The Python code for this conversion was traced and generated by ParaView using the GUI, and that generated script was edited by the team to iterate through the 2000 .csv files using pbsbatch and mpirun. Then, several data filters were applied to the .vtk files, and this Python script was generated again using ParaView, and the team edited that script so it would iterate through the 2000 files.

Finally, the team had to take screen shots of the 100,000 stars at each of the 2000 timesteps, and this was again done in ParaView and with mpirun/pbsbatch. These 2000 .png files were then converted to a .gif using the convert command in ImageMagick. Since each student had their own application, only up to 3 students could work on ParaView at one time, and the data conversions took up most of the time.

Once the data conversions were complete, there was very little time to debug the scripts and figure out how to correctly apply filters, so not all of the data was included. The bug in the script was a warning generated by ParaView that would clog up the batch system after iterating about 12 times and cause it to freeze. Because of this, only 12 files could be converted at once, and due to the 48 hour time constraint, this limited how many files could be converted. Also, only one student was trained on how to use ParaView filters or how to run

ParaView jobs, so the other students could not help debug this issue. The team did not expect the ParaView application to be a data parsing/conversion challenge. Next year, the team should prepare by practicing parsing and converting data, and more than one person should be fully trained per application.

E. ParConnect

In bioinformatics, DNA sequencing is the process of finding the precise order of nucleotides in a DNA molecule. Current generation of sequencing machine cannot identify the structure of a whole genome at once due to the high error probability nature. Instead, randomly small strand of the genome called *k-mers* (about 100 base pairs) are sequenced separately. The whole genome can be obtained by assembling those strands together through the use of graph data structures and graph algorithms. If there is a reference genome to be compared with then the assembly process is actually a very easy problem. Unfortunately, most assembly problems fall into the category of de novo (from the beginning) assembly problem with no reference genome to be compared with. In this case, overlapping each read is necessary to assemble the original genome. A vivid analogy is it's easier to assemble small pieces of a puzzle if we were given the complete puzzle picture rather than without it. Metagenomics is the study of the genetic material from environmental sample. Whereas single genome sequencing as introduced earlier only deals with data from a single organism, metagenomics sequencing has to process data comes from many species such as an environmental sample, which makes it inherently much more difficult.

In order to address the complexity of sequencing metagenomics genomes, a group of professionals from Georgia Institute of Technology came up with a new method for sequencing metagenomics dataset. ParConnect is a bioinformatics application that implements a parallel and distributed algorithm presented in a paper at SC16. That paper also won the Best Student Paper Award at SC15 Conference. The algorithm computes the connected components in the undirected graphs or weakly connected components in the directed graphs.

For the Student Cluster Competition at SC16, the students were asked to replicate the same graphs and data like in the paper. In fact, this is the first time that a reproducibility problem were present in the competition. The goal according to the SC committee is to acquaint students with the idea of reproducibilty in science since it is the key component behind the scientific method.

Before the competitions, we were given a sample dataset which is in FASTQ format. The file takes up around 6 GB in size. At first, we came across so many problems running the application with the file due to the limited memory and computing capabilities of our cluster. Fortunately, 2 datasets (small.fastq and large.fastq) that were given during the competition are much smaller than that. In addition, we didn't take much time to parse the data from the output text file in order to generate all the graphs since the Python scripts for parsing and visualization were already written before the competition with little or no modifications at all.

The overarching requirement for the competition report is to restate the central claims in the paper then substantiate the claims through data and graphs. The main claim in the paper is the parallel algorithm devised in the paper could be improved by excluding completed partition and using load balancing. 3 variants of the algorithm (called Naive, AP, and AP_LB in the paper) are then compared for runtime performance in order to support the claim.

First, we had to tabulate the number of edges and connected components for both datasets. The reason is to confirm that if others try to reproduce the results, they should know that they are working on the same datasets. Second, we were required to compare the Allinea Perf Report profiler's timings to ParConnect's output timings and explain any differences. From the table we can see that the timing computed from the output file and that from Allinea Performance Report are not the same. The difference is due to the fact that the instruction to calculate the communication time does not take into account all MPI subroutine call, whereas the profiler included those.

Next, we have to generate 3 graphs that show the communication times and computation times over each iteration corresponding to each algorithmic variants. There are some aspects of these graph that are similar to the paper. For instance, in the first and second iteration in the paper, the total time for each starts out decently, but then for some iterations later, those times skyrocket significantly then decreasing at a somewhat linear speed. In our experiments, there are no cusps like that at all. Nevertheless, the general trend that the time will decrease as you move to the larger iteration still holds here. Like in the graph, the communication time dominated over the computation mostly for all iterations beside some of the very last iterations (this is more noticeable for the AP and AP_LB graph). The reason for all these problems are probably because the data set that was given is not big enough.

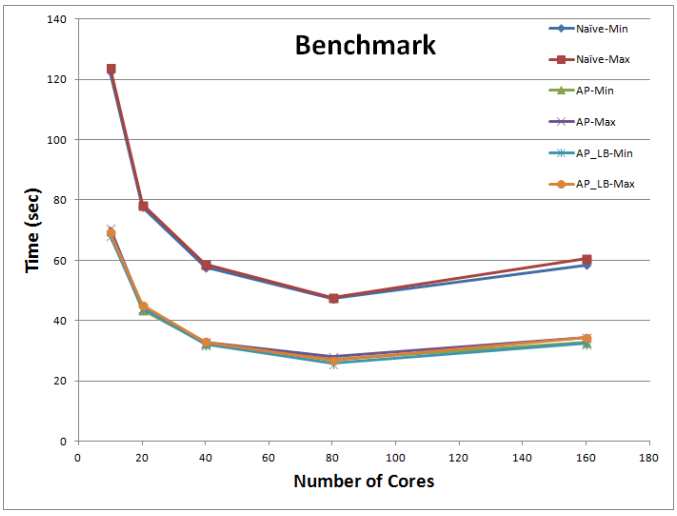


Fig. 2. Performance of the three algorithm variants for data set small.fastq, using increasing number of processor cores.

Finally, we have to perform a strong scaling experiment on the small dataset. The result is summarized in Fig. 2. Similar

to the paper, we can for the most part reproduce the scaling property of the application. Nevertheless, due to many limiting factors such as inter networking communication limitation, we only see strong scaling from 20 cores to around 80 cores. From around probably 90 or 100 cores, the communication time will be larger than computation, so theres no more advantage from parallelism anymore. One thing that we cant reproduce is the timing improvement between the AP and AP_LB. In the paper, the AP_LB consistently performed much better than the AP variants which is not the case in our graph here.

Since we are one of the top teams from the Reproducibilty Challenge at SC16, we were invited to submit a paper to a special issue of the Parallel Computing journal. As a result, our long-term goal is to work with our mentors for the next couple of months in order to refine our writing further. Hopefully, we would be able to get our paper accepted for publication.

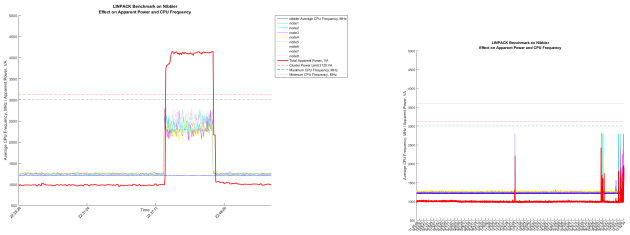
F. Password Cracking

In modern web security, authentication platforms often store passwords in a hashed representation in order to mitigate credential loss in the occurrence of a compromised password database. Traditionally, these hashes are the result of a password combined with a known chunk of data, a salt, used with a cryptographic hashing algorithm. Salts prevent attackers from using a precomputed lookup table of password hashes, a rainbow table, to immediately decipher the original passwords.

The goal of this application in the competition was to decipher as many of the original passwords from a list of password hashes and their salts. As an added challenge, each password used one of several hashing algorithms with different computation parameters.

To cope with this diverse password set, John the Ripper was used with a custom Celery executor and AMQP task queue to brute-force the correct passwords. John the Ripper is a popular and expansive password cracking application that enabled deciphering any of the given hashes with a single executable. Celery formed a flexible meta-architecture that allowed dynamic execution, profiling, and termination of John at run-time. This allowed the prioritization of the easiest computed cryptographic hashing algorithms and the creation of a probabilistic password-cracking model, preventing the application from getting stuck on complex passwords. Celery also afforded the benefit of innate performance visualization, system-wide data persistence, and dynamic resource scaling.

In the competition, the team was given a dataset of 2.6 million password hashes each with a unique salt. The hashes were resultant from two common hashing algorithms, MD5 and Blowfish. The initial dataset was subsequently split by hash type and filtered into the AMQP task queue for later consumption. Celery executors, once launched, would autonomously consume each task from the queue and execute John for the specific job. At each endpoint, metadata, including the results of John, was stored in a Redis database with high-persistence journaling. This setup enabled data metrics to be analyzed from multiple workers and stages in aggregate, creating a probabilistic model for password cracking duration



that altered the maximum runtime of John tasks. This allowed us to adjust the platform to aim for the most quickly cracked passwords, at the expense of discarding some computation on hashes proven more difficult to brute-force.

To prevent Celery executors from compromising the effectiveness of other applications via unintentional resource sharing, execution under PBS Torque was implemented at the competition. However, this and several other last-minute changes led to considerable difficulty in proper termination of Celery Executors under certain scenarios, which proved problematic on the effectiveness of the task allocation pipeline and resulted in the node hanging and have to be restarted. It is not clear why these issues occurred, and there was no time during the actual competition to diagnose the problems. Possible sources could include last minute changes in the node memory and the infiniband network configurations, which were done to accommodate other applications.

G. Open HPC Software Integration

Open HPC Software Integration writeup here.

H. Cluster Management

The power monitoring scripts show the outputs of the apparent power usage for each node, using SNMP queries to the Geist PDU. The plots also show the CPU clock speed for every core, obtained from /proc/cpuinfo. The figures above show the the power consumption when running the HPL benchmark on the full system of 320 cores (left), and on a smaller system of 220 cores (right). For the case of 320 cores, the power usage exceeds the power limit of 3.1 KW, but stays under the limit when the number of active cores is reduced to 220.

I. Mic. Figures from example paper

Fig. 4 shows the measured CW RF power characteristics at 10.24 GHz of a transistor biased at V_{ds}=20V, I_{ds}=150mA. It delivers, on the optimum load impedance $Z_{load} = 20 + j.18\Omega$ and exhibit 4.8W/mm output power at 3.5dB of compression with a PAE of 46% and an associated gain of 11.5dB.

Figure 6 shows a comparison between RF pulsed measurements and simulations with this model. The transistor is measured at Vds0=20V, Ids0=115mA (190mA/mm), on a load impedance $Z_{load} = (20 + j18)\,\Omega$. The bias voltages are continuous, and the RF signal, at a frequency of 10GHz, is pulsed. Its duration is $5\mu s$, and its period $100\mu s$. On this example, the RF input power equals 20dBm, corresponding

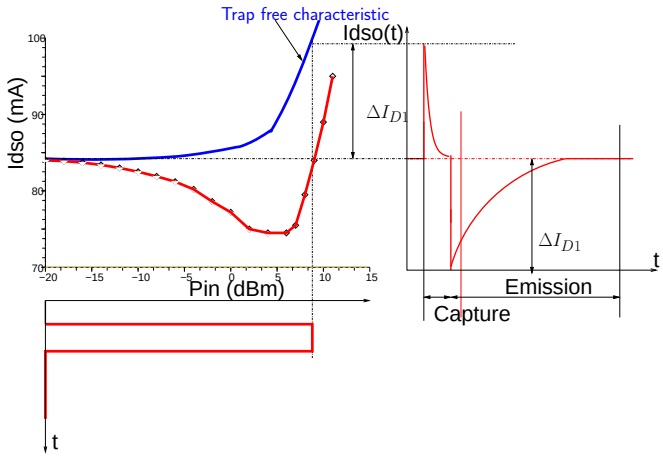


Fig. 3. Representation of the mechanism induced by traps on the average drain current.

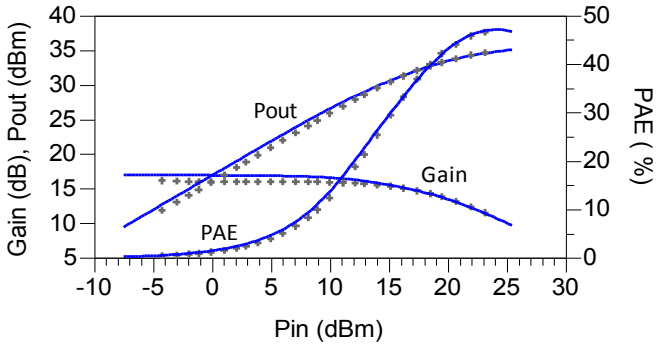


Fig. 4. Measurements (grey), modeling (blue) of the CW RF performances of a 8x75μm AlInN/GaN HEMT at 10.24GHz in class AB. $Z_{loadOPT} = (20 + j.18)\Omega$.

approximately to 1dB of gain compression. The very large amplitude of the current discontinuity at the moment when RF is switched off leads to a transient current from 97mA to 115mA (i.e. the nominal bias point).

Real-time performance monitoring and visualization was implemented using Grafana and InfluxDB, populated via the aforementioned power monitoring scripts.

VI. Conclusion and Observations

what went right, what went wrong. we learned a lot, we'll do it again. Please add any thoughts here (pros/cons/what to do differently next year).

We will certainly shift our focus from custom-tailored solutions towards refining and optimizing preexisting solutions. Additionally, any supplemental utilities and scripts that may be necessary should be created before the competition.

- Next year, the team should be prepared to parse and convert data to their applications' desired format and understand the esoteric features of each application.
- Multiple teammates should be trained for each application to easily divide labor and assist in problem-solving.

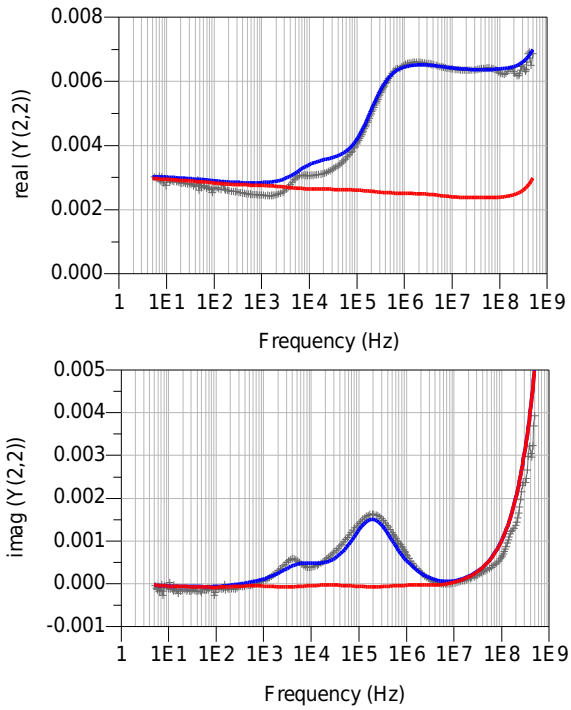


Fig. 5. $\Re\{Y_{22}\}$ and $\Im\{Y_{22}\}$. AlInN/GaN 8x75 μm HEMT measurements at $V_{ds} = 20V$, $I_{ds} = 120mA$ (grey) are compared to a simple electro-thermal model (red) and an electro-thermal model including activated drain-lag effects (blue)

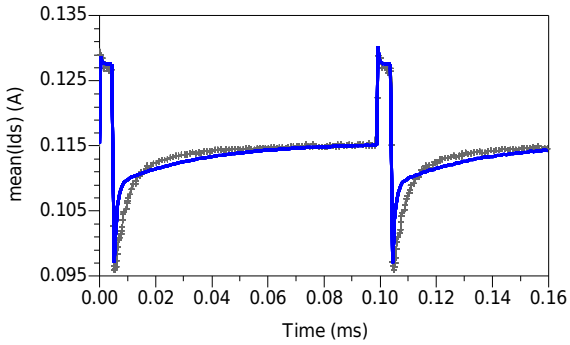


Fig. 6. Measurements (grey) and modeling (blue) of the average output current in pulsed RF signal operation at 10 GHz (5μs-100μs) with DC bias (20V, 115 mA). The amplitude of the transient induced by detrapping when RF is switched off is accurately modeled.

- Teammate scheduling and resource allocation should be planned before the competition.
- Scripting ability should be emphasized, for its usefulness in automation and problem-solving in constrained time frames.
- Any supplemental software, such as visualization or logging platforms, should be operational before the competition with intuitive integration endpoints.
- Promotional displays should be prepared prior to the competition and contain information such as:
  - Student biographies
  - SDSU, CSRC (Organizational information)

- Application monitoring (Performance visualizations)
- Sponsorship information

ACKNOWLEDGMENTS

This work was made possible by a grant from Intel, who provided the cluster, engineering support and a generous travel grant; the OpenHPC Community; and the San Diego State University Computational Sciences Research Center. The authors would like to also thank members of the SDSU Cyber Defense Team (Marcus Butler, Alexander Kirk) and the Edwards Lab (Gheni Guerios and Daniel Cuevas) for their help with the applications.

REFS TO BE INTEGRATED INTO BIB:

[1] Computational Science Research Center (CSRC). [Online]. http://www.csrc.sdsu.edu/
[2] SC16 Student Cluster Competition. [Online]. http://www.studentclustercompetition.us/
[3] Open HPC Community. [Online]. http://www.openHPC.community
[4] Applied Computational Science and Engineering Student Support (ACSESS). [Online]. http://w
[5] OpenHPC Git Hub Repository - List of Components. [Online]. https://github.com/openhpc/ohpc
[6] P. Smith, D. Smith, T. Hoefler, A. Labutina and T. Overmeyer S. Harrell, "Methods of Creat

REFERENCES

1 *Supercomputing '16 Student Cluster Competition*. Last Accessed on 11/1/16 at http://www.studentclustercompetition.us/.

2 *Detailed Specifications of the Intel Xeon E5-2600v4 Broadwell-EP Processors*. Last Accessed on 11/01/16 at https://www.microway.com/knowledge-center-articles/detailed-specifications-of-the-intel-xeon-e5-2600v4-broadwell-ep-processors/.