Ookami Webinar, 03/03/2021

Ookami - 狼

- Ookami is Japanese for wolf
  - Homage to the origin of the processor and the Stony Brook mascot
- A computer technology testbed supported by NSF
- Available for researchers worldwide

  (excluding ITAR prohibited countries & restricted parties on the EAR entity list)
- Usage is free for non-commercial and limited commercial purposes

What is Ookami

- 174 **A64FX** compute nodes each with 32GB of high-bandwidth memory and a 512 Gbyte SSD
  - Same as in currently fastest machine worldwide, Fugaku
  - First deployment outside Japan
  - HPE/Cray Apollo 80
- Ookami also includes:
  - 1 node with dual socket AMD Rome (128 cores) with 512 Gbyte memory
  - 2 nodes with dual socket Thunder X2 (64 cores) each with 256 Gbyte memory and 2 NVIDIA V100 GPU
  - Intel Sky Lake Processors (32 cores) with 192 Gbyte memory
- Delivers ~ 1.5M node hours per year

Fugaku #1 Fastest computer in the world

First machine to be fastest in

all 5 major benchmarks:

- Green-500
- Top-500 – 415 PFLOP/s in double precision – nearly 3x Summit!
- HPCG
- HPL-AI
- Graph-500

- 432 racks
- 158,976 nodes
- 7,630,848 cores
- 440 PF/s dp (880 sp; 1,760 hp)
- 32 Gbyte memory per node
- 1 Tbyte/s memory bandwidth/node
- Tofu-2 interconnect

https://www.r-ccs.riken.jp/en/fugaku

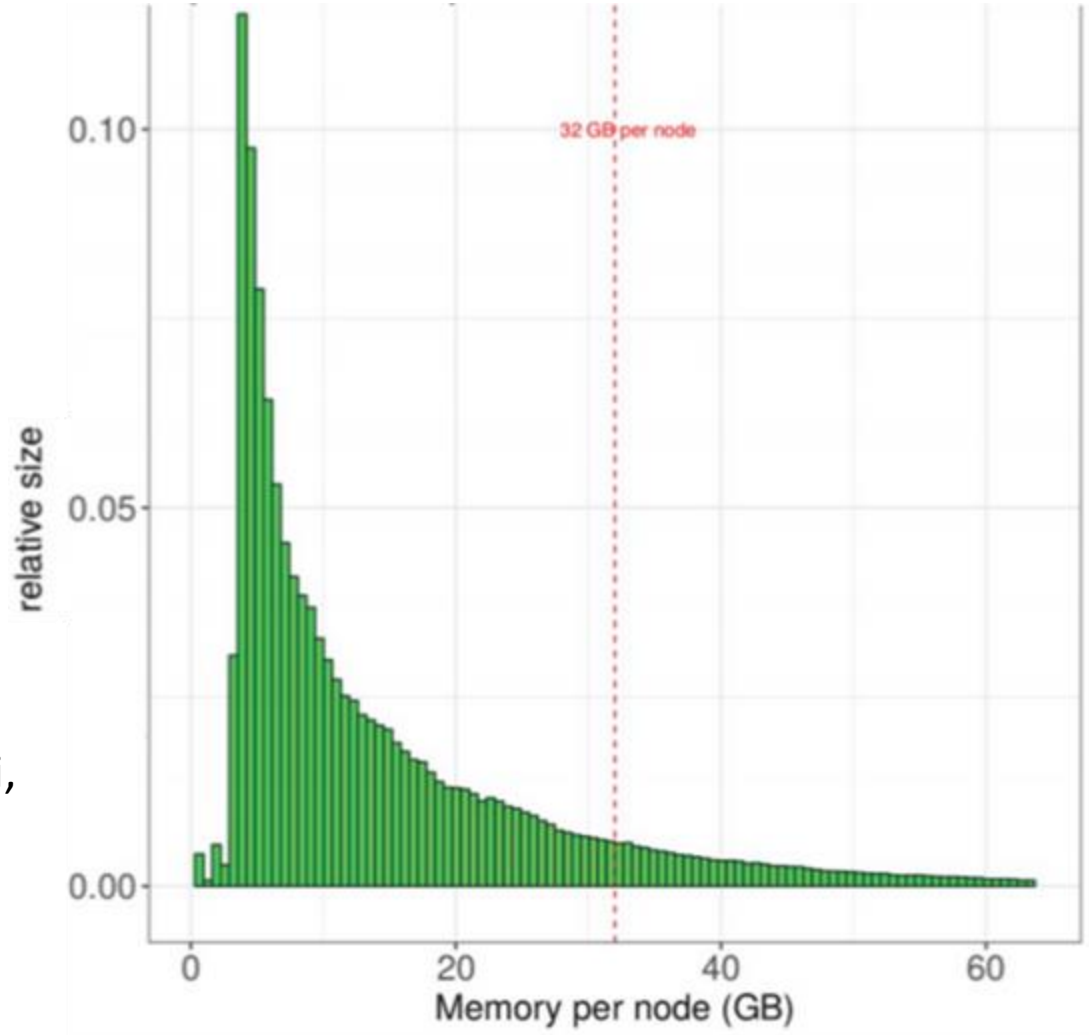
Benefits for users

- Access and evaluate state-of-the-art computing technology
- Conduct your own research on newest processors
- Port, tune, and optimize your code in preparation for a new generation of supercomputers
- Secure environment with system maintenance done by the Ookami team

Memory Statistics of Typical Jobs

2017 analysis of XSEDE workload revealed
86% of all jobs need less than 32 GB / node

These 86% of jobs correspond to 85% of the total XSEDE cpu-hour usage



Simakov, White, DeLeon, Gallo, Jones, Palmer, Plessinger,Furlani, "A Workload Analysis of NSF's Innovative HPC Resources Using XDMoD," arXiv:1801.04306v1 [cs.DC], 12 Jan 2018

A64fx at a Glance

- ARM V8 64-bit
- 512-bit SVE
- 48 compute cores
- 4 NUMA regions
- 32 (4x8) GB HBM @ 1 TB/s
- PCIe 3 (+ Tofu-3) network

“Programmability of a CPU, performance of a GPU”

Satoshi Matsuoka (Head of RIKEN, home of Fugaku)

- Blazing fast memory
- Easily accessed performance
- New technology path to exascale

A64fx NUMA Node Architecture

- Supports high calculation performance and low power consumption
- Supports Scalable Vector Extensions (SVE)
- **4 Core Memory Groups (CMGs)**
  - 12 cores (13 in the FX1000)
  - 64KB L1$ per core
    - 256b cache line
  - 8MB L2$ shared between all cores
    - 256b cache line
  - Zero L3$
  - 8 GB HBM at 256GB/s

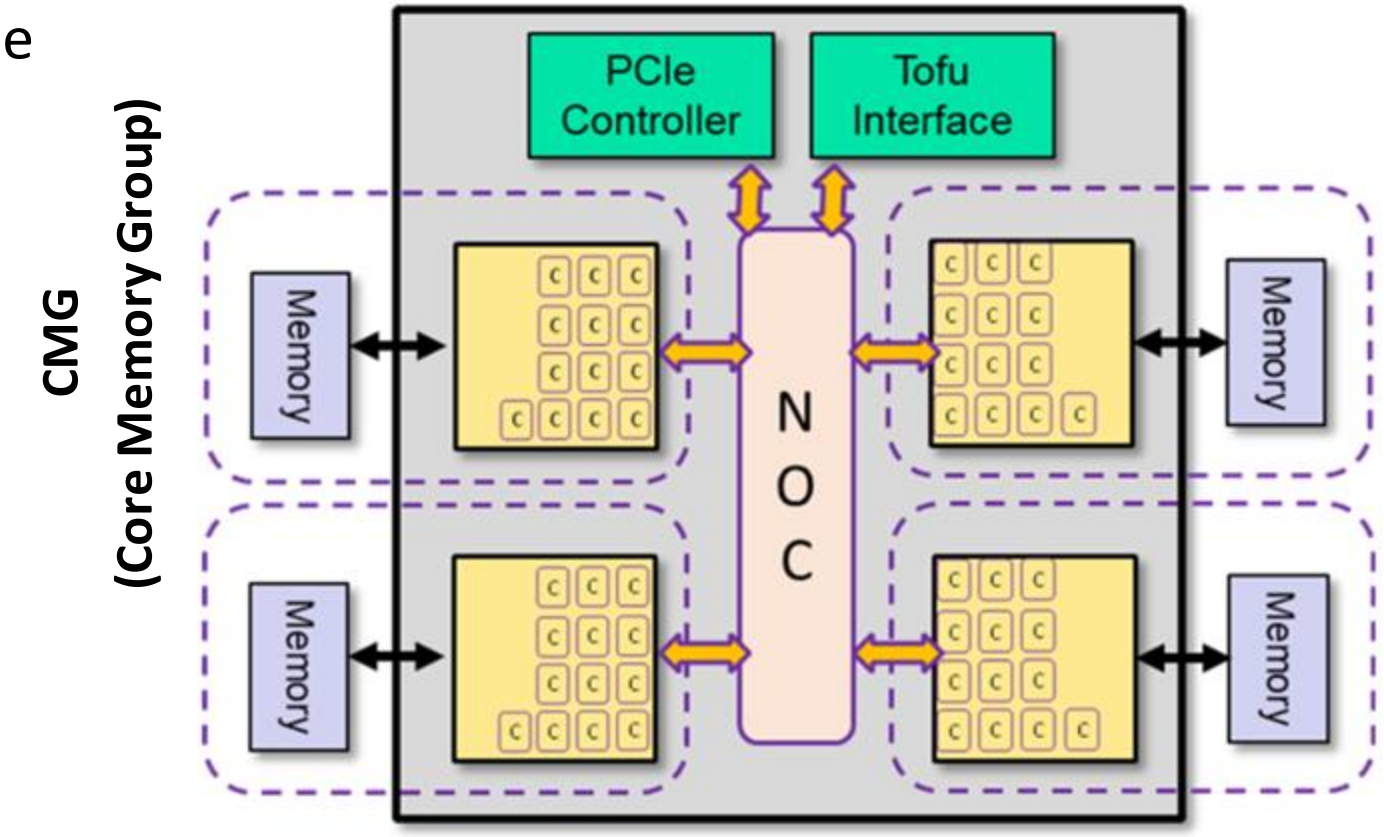
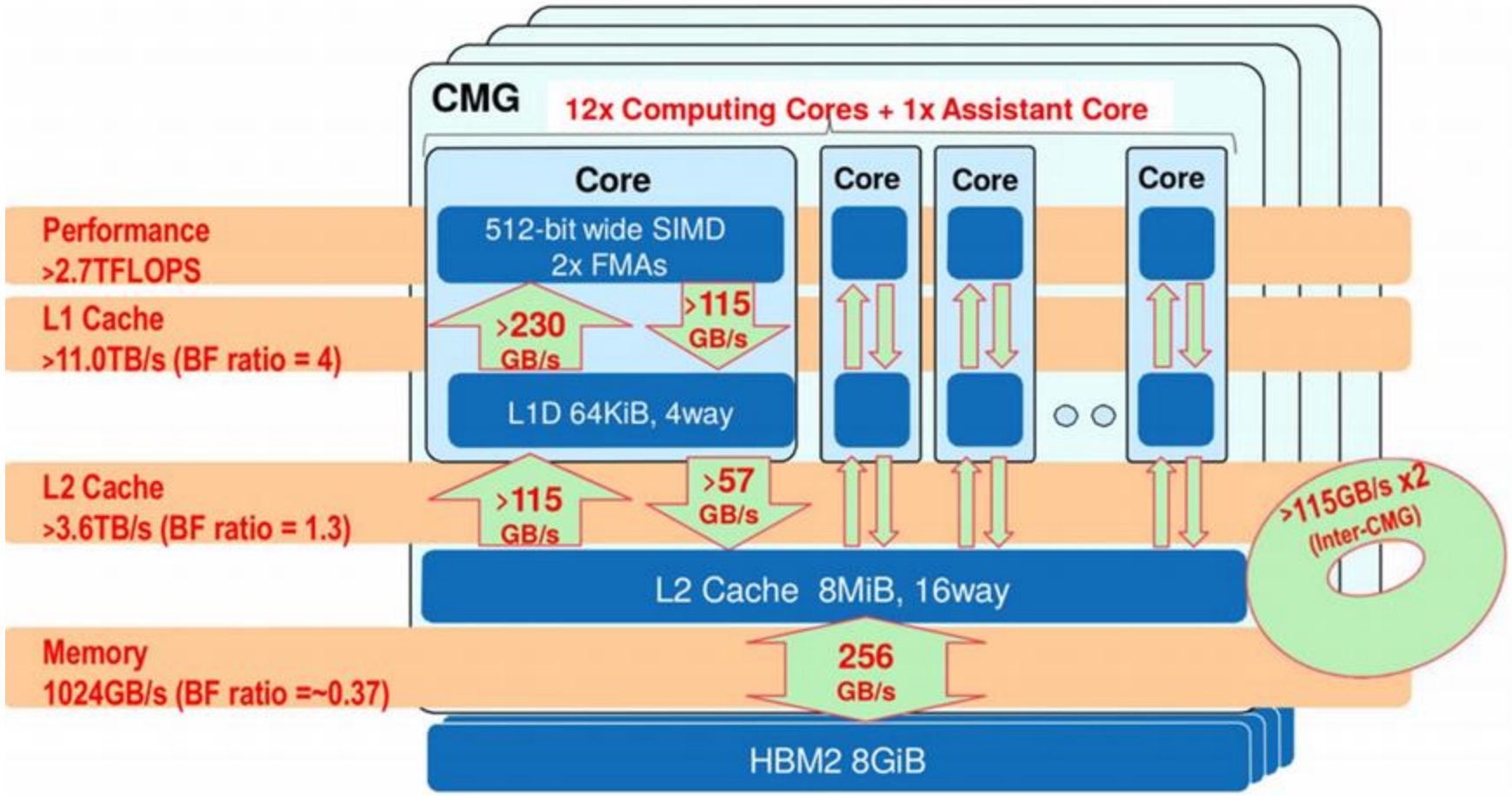


Diagram is the „1000“ chip.
We have „700“ chip, i.e. no assistant cores and no Tofu interface

http://www.jicfus.jp/jp/wp-content/uploads/2018/11/msato-190109.pdf

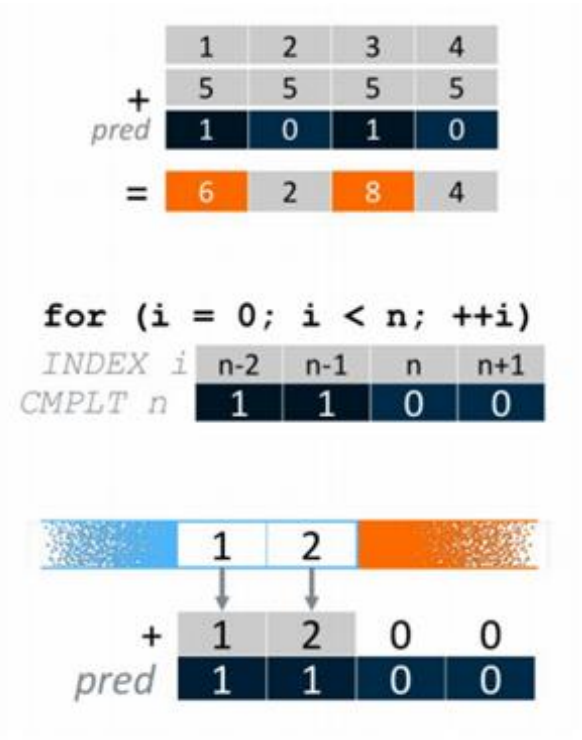
A64fx Core Memory Group



http://www.jicfus.jp/jp/wp-content/uploads/2018/11/msato-190109.pdf

SVE (Scalable Vector Extensions)

- Enables Vector Length Agnostic (VLA) programming
  - VLA enables portability, scalability, and optimization
  - The actual vector length is set by the CPU architect
    - Any multiple of 128 bits up to 2048 bits
    - May be dynamically reduced by the OS or hypervisor
- Predicate-centric architecture
  - Predicates are central, not an afterthought
  - Support complex nested conditions and loops
  - Predicate generation also sets condition flags
  - Reduces vector loop management overhead
- SVE was designed for HPC and can vectorize complex structures
  - Gather-load and scatter-store; horizontal reductions
  - SVE begins to tackle traditional barriers to auto-vectorization
    - Software-managed speculative vectorization allows uncounted loops to be vectorized.
    - In-vector serialized inner loop permits outer loop vectorization in spite of dependencies.
- Support from open source and commercial tools



SVE vs Traditional ISA

How do we compute data which has ten chunks of 4-bytes?

Aarch64 (scalar)

- Ten iterations over a 4-byte register

NEON (128-bit vector engine)

- Two iterations over a 16-byte register + two iterations of a drain loop over a 4-byte register

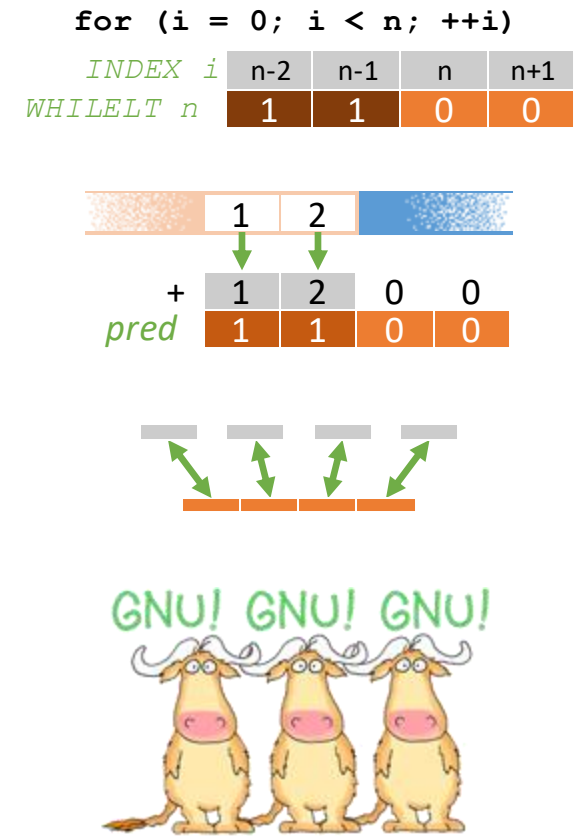
SVE (128-bit VLA vector engine)

- Three iterations over a 16-byte **VLA register** with an adjustable **predicate**

arm

Summarized

- SVE enables Vector Length Agnostic (VLA) programming
- VLA enables portability, scalability, and optimization
- Predicates control which operations affect which vector lanes
  - Predicates are not bitmasks
  - You can think of them as dynamically resizing the vector registers
- The actual vector length is set by the CPU architect
  - Any multiple of 128 bits up to 2048 bits
  - May be dynamically reduced by the OS or hypervisor
- SVE was designed for HPC and can vectorize complex structures
- Many open source and commercial tools currently support SVE



arm

What else

- Operating system is CentOS 8
- Module environment
- High-performance Lustre file system (~800TB of storage)
- Slurm workload manager
- Compilers: GNU, Arm, Cray, Nvidia, Fujitsu (soon)
- Continuous growing stack of preinstalled software
- Ticketing system for any kind of issues / questions /requests

Modules

Continuously growing software stack managed through modules

```
[esiegmann@login1 ~]$ module avail
------------------------------------------------ /cm/local/modulefiles ------------------------------------------------
cluster-tools/9.0  dot             ipmitool/1.8.18  module-git   openldap                      python37
cmd                freeipmi/1.6.4  lua/5.3.5        module-info  openmpi/mlnx/gcc/64/4.0.3rc4  shared
cmjob              gcc/9.2.0       luajit           null         python3

------------------------------------------------ /cm/shared/modulefiles ------------------------------------------------
blacs/openmpi/gcc/64/1.1patch03  hdf5/1.10.1                          lapack/gcc/64/3.8.0          slurm/19.05.7
blas/gcc/64/3.8.0                hdf5_18/1.8.21                       mpich/ge/gcc/64/3.3.2        ucx/1.6.1
bonnie++/1.98                    hwloc/1.11.11                        mvapich2/gcc/64/2.3.2
cm-pmix3/3.1.4                   intel-cluster-runtime/ia32/2017.8    netcdf/gcc/64/gcc/64/4.7.3
default-environment              intel-cluster-runtime/intel64/(default)  netperf/2.7.0
fftw3/openmpi/gcc/64/3.3.8       intel-cluster-runtime/intel64/2017.8 openblas/dynamic/(default)
gdb/8.3.1                        intel-cluster-runtime/mic/2017.8     openblas/dynamic/0.3.7
globalarrays/openmpi/gcc/64/5.7  iozone/3_487                         openmpi/gcc/64/1.10.7

------------------------------------------------ /lustre/shared/modulefiles ------------------------------------------------
anaconda/3                          arm/gcc/9.3.0             CPE-nosve       htop/3.0.2          openmpi/gcc/4.0.5
archiconda/3                        arm/gcc/armpl/20.3.0      curl/7.73.0     internal/template   openssl/1.1.1h
arm/arm-linux-compiler/20.3         arm/licenceserver/20.1.3  doxygen/1.8.20  lapack/3.9.0        test
arm/arm-linux-compiler/armpl/20.3.0 cmake/3.19.0              git/2.29        ncurses/6.2         xpmem/2.6.3
arm/forge/20.1.3                    CPE                       gnuplot/5.4.0   openblas/0.3.10     zsh/5.8
```

Slurm

- Job scheduling system
- Queues jobs based on available resources, fair share policy
- Run jobs with
  - `srun` interactively
  - `sbatch` for batch processing

```
#SBATCH --job-name = my_job
#SBATCH --output=my_output.log
#SBATCH --nodes=1
#SBATCH --time=00:05:00
#SBATCH --partition=short

Module purge
module load moduleA
srun ./myprogram
```

Queue	Time Limit	Min Nodes	Max Nodes
short	4h	1	32
large	8h	24	80
long	2d	1	8
extended	7d	1	2

Getting Started

- LINUX and MACOS:

  SSH to Ookami: `ssh -X NetID@login.ookami.stonybrook.edu`

- Windows :

  Use MobaXterm

- DUO Authentication is used

- Directories:

  `/lustre/home/<my NetID>`

  `/lustre/scratch/<my NetID>`

  } personal directories

  `/lustre/projects/<projectname>` shared directory

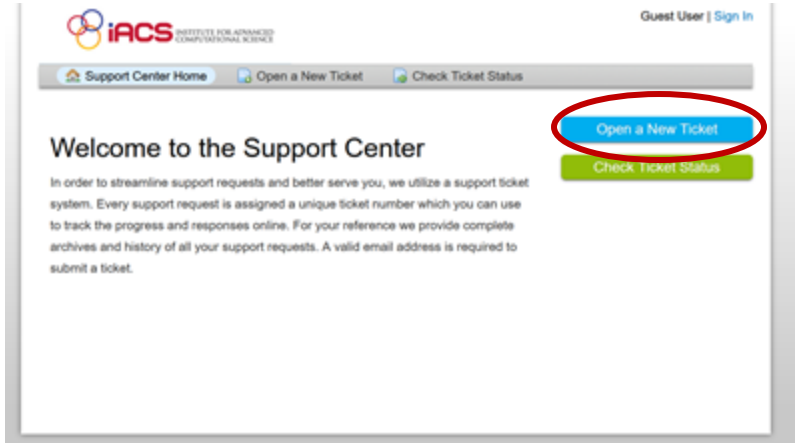
Use Case

- SWIM

The swim code is a finite-difference approximation of the shallow-water equations and is known to be memory bandwidth limited.

Getting Accounts

- Getting access:
  - Create a project request and submit it through ticketing system: https://www.stonybrook.edu/commcms/ookami/support/faq/getting-a-project-on-ookami.php
  - If you are not affiliated to SBU: Fill a volunteer demographic form
  - All members of a project will get accounts
  - New members to an existing project can get accounts anytime



Getting Accounts

- Submit a project request
  - Testbed:
    - Porting and tuning software
    - Benchmarking
    - Limited production calculations to demonstrate capability
    - Significantly less than 15,000 node hours per year
    - First two project years
  - Production:
    - Less than 150K node hours per year
    - Lower priority during the first two project years

Get in Contact

- https://www.stonybrook.edu/ookami/
- Ticketing system: https://iacs.supportsystem.com/
  - Technical questions / issues
  - Project / account requests
- Ookami_computer@stonybrook.edu
  - For general questions
- Bi-weekly Hackathon (Tue 10am – noon, Thu 2 – 4pm)
- Slack Channel #OOKAMI