# Introduction to Ookami supercomputer

Eva Siegmann, Robert Harrison

Institute for Advanced Computational Science, Stony Brook University, USA

CnC 2021: The Thirteenth Annual Concurrent Collections Workshop
27 October 2021

# Ookami - 狼

- 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

SBU's Seawulf

Stony Brook University

OOKAMI

University at Buffalo
The State University of New York

# 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

# tl;dr

*"Programmability of a CPU, performance of a GPU"*

Satoshi Matsuoka

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

# Ookami

| Node | |
|---|---|
| Processor | A64FX |
| #Cores | 48 |
| Peak DP | 2.76 TOP/s |
| Memory | 32GB@**1TB/s** |
| **System** | |
| #Nodes | 176 |
| Peak DP | 486 TOP/s |
| Peak INT8 | 3886 TOP/s |
| Memory | 5.6 TB |
| Disk | 0.8 PB Lustre |
| Comms | IB HDR-100 |

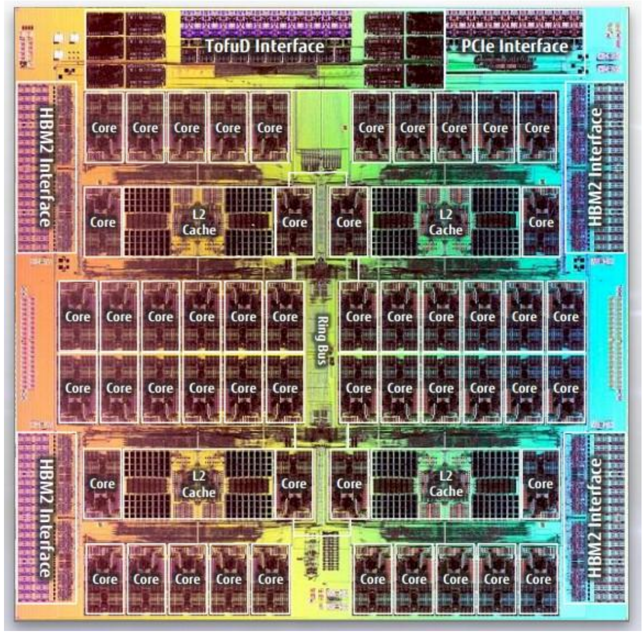
# What is Ookami

- 176 1.8Ghz **A64FX** compute nodes each with 32GB of high-bandwidth memory and a 512 GB SSD
  - Same as in currently fastest machine worldwide, Fugaku
  - First open deployment outside Japan
  - HPE/Cray Apollo 80


- Ookami also includes:
  - 1 node with dual socket AMD Milan (64 cores) with 512 GB memory and 2 NVIDIA V100 GPUs
  - 2 nodes with dual socket Thunder X2 (64 cores) each with 256 GB memory
  - 1 node with dual socket Intel Skylake (36 cores) with 192 GB memory
- Delivers ~1.5M node hours per year

# A64FX NUMA Node Architecture

- Arm V8-64bit
- Supports high calculation performance and low power consumption
- Supports Scalable Vector Extensions (SVE) with 512-bit vector length
- **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$
- 32 (4x8) GB HBM @ 1 TB/s
- PCIe 3 (+ Tofu-3) network

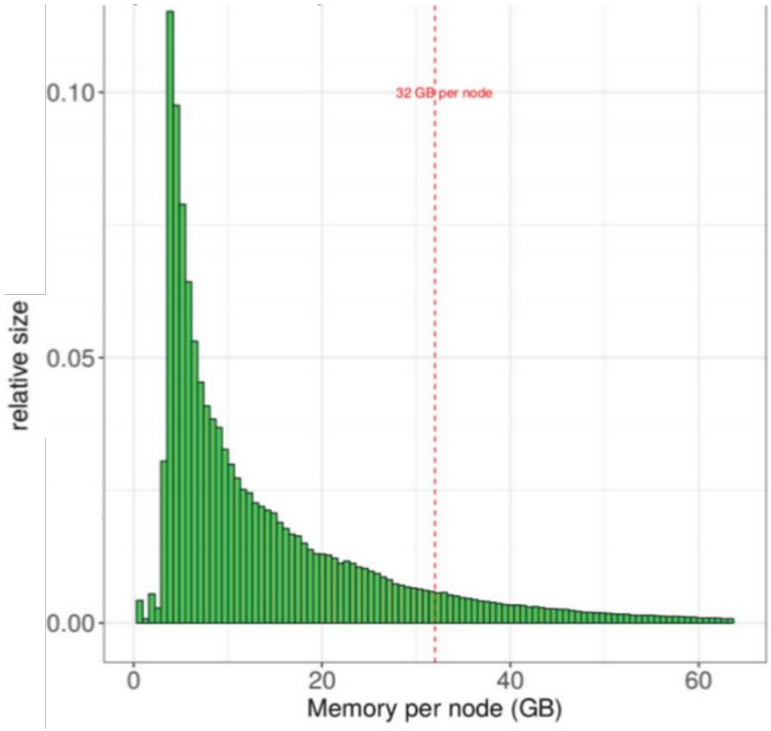


# Memory Statistics of Typical Jobs

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



# What else

- CentOS 8 operating system
- DUO Authentication
- High-performance Lustre file system (~800TB of storage)
- Slurm workload manager
- Compilers: GNU, Arm, Cray, Fujitsu, Intel, Nvidia
- Continuous growing stack of preinstalled software
    - MPI implementations
    - Math libraries
    - Performance analysis & debugging:
      (Arm Forge, Cray, GNU, TAU, ..)

# Project Phases

- Phase I (year 1)
    - Acquisition, deployment, early user operations, acceptance (technical+formal)
- Phase II (years 2-3)
    - Technology evaluation
        - Emphasis is on users exploring the technology, porting and tuning applications
    - Allocations performed by SBU
- Phase III (years 4-5)
    - Production operations
        - Emphasis shifts to using the machine for production scientific computing
    - Allocations performed by XSEDE with integration into their accounting, etc.

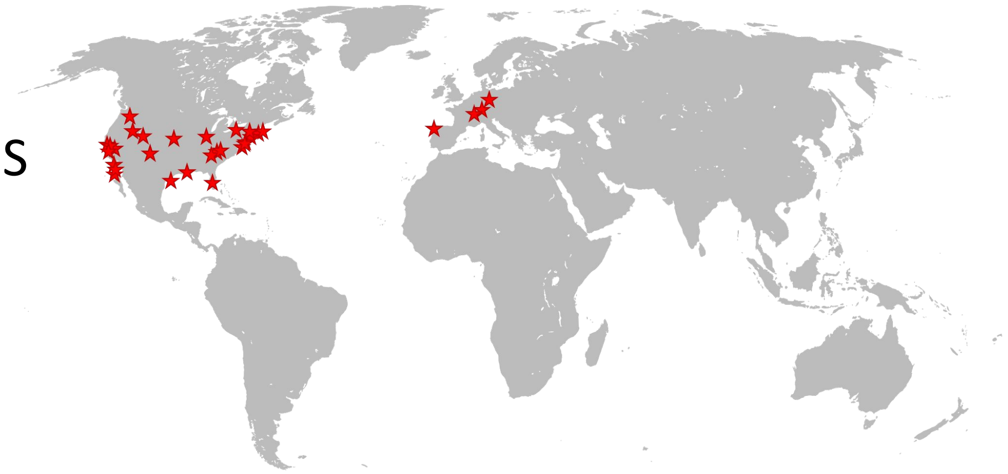
# Our Goals

- Familiarize users with this new technology
- Achieving high-performance requires detailed knowledge of
    - computer architecture
    - performance analysis and modeling
    - high-performance programming models
- Enable users to make effective use of the resource
    - E.g., switching from serial implementation to a fully-pipelined, vectorized, and threaded version → up to 100x speedup
    - E.g. switching compilers → 2 - 10x speedup

# Projects

- Total: 185 users & 60 projects
- 91.7% projects from within the US
- 8.3% from Europe
- 95% from academia
- Complete list of projects:

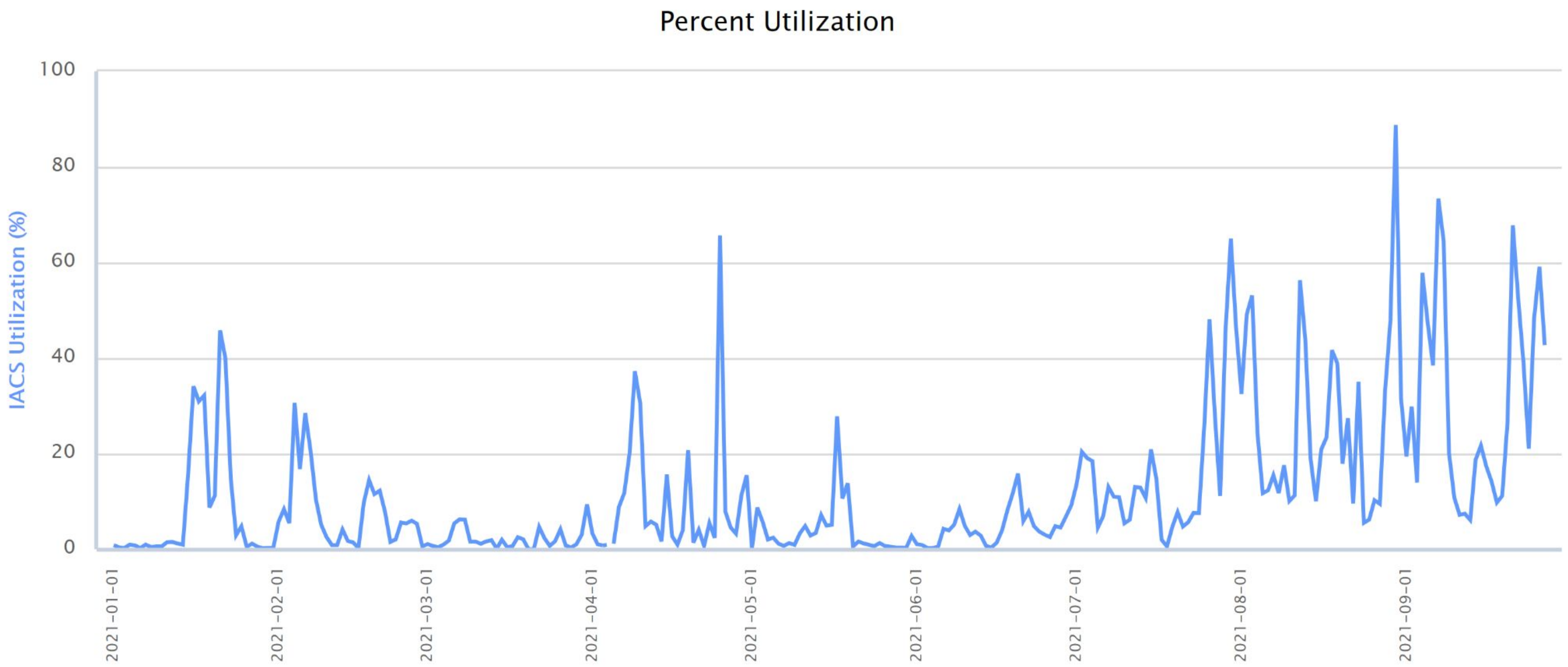
https://www.stonybrook.edu/ookami/projects/



# User Support

- Slack channel
- Ticketing system handled by the HPC support team
- Virtual office hours twice a week (Tue and Thu, each 2 hrs)
- Regular webinars
  - Vectorization hackathon, TAU, likwid, XDMoD, etc.

# Utilization 2021



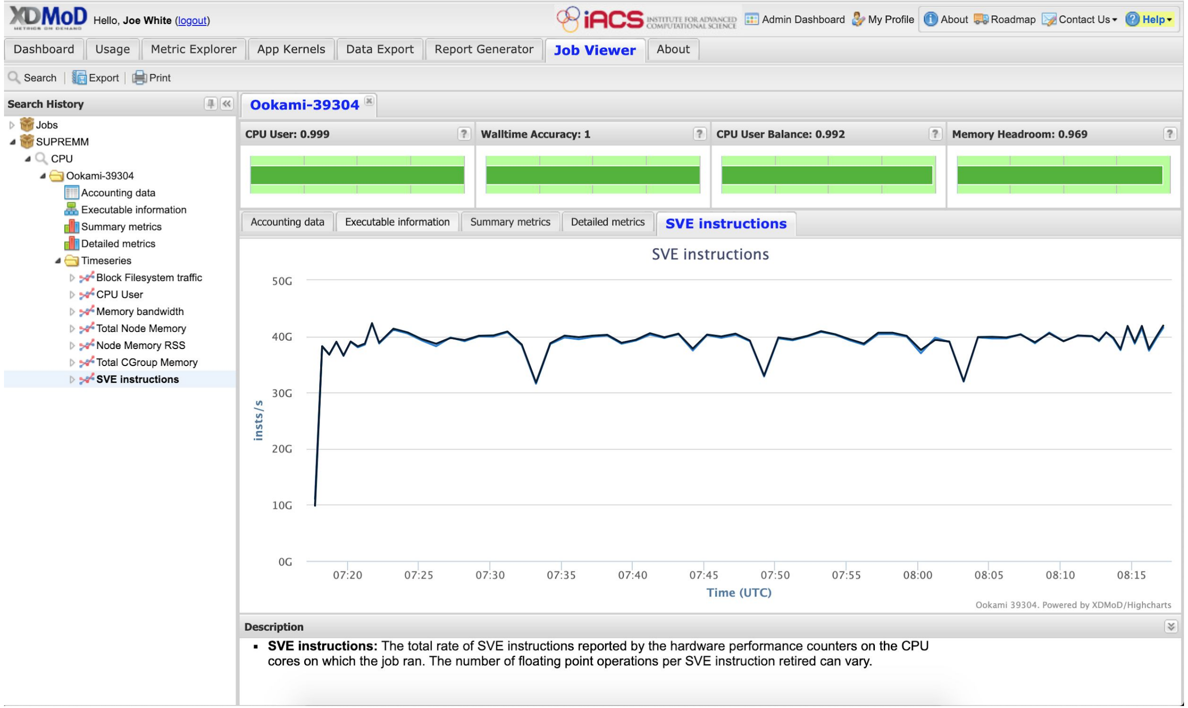
# XDMoD

- **Open XDMoD: Open Source version for Data Centers**
  - Used to measure and optimize performance of HPC centers
- **Goal: Optimize Resource Utilization and Performance**
  - Provide detailed information on utilization
  - Measure quality of service
  - Measure and improve job and system level performance
- https://ookami.ccr.xdmod.org/

# Job Viewer: Measuring Job Performance

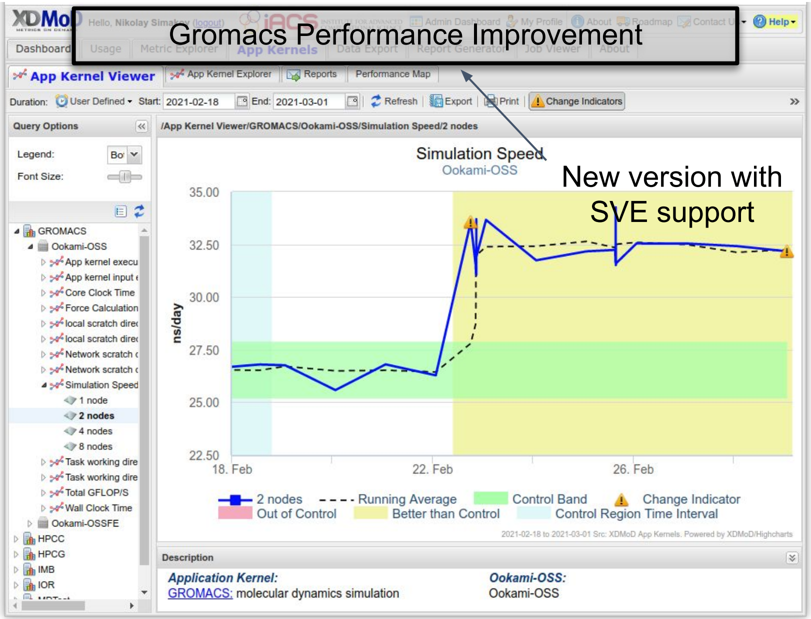
Collect and display detailed performance data collected from nodes, e.g.:

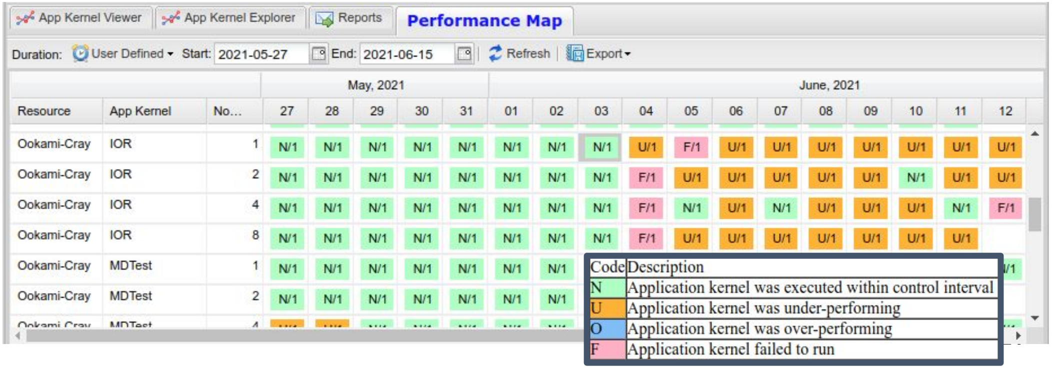
- SVE instruction count
- Node power usage
- Memory



# QoS: Application Kernels

- Computationally lightweight benchmarks or applications
  - Run periodically or on demand to actively measure performance
- Measure system performance from user's perspective
- Proactively identify underperforming hardware and software



# Case Study:
# Compiler and math library vectorization

# Compiler and math library vectorization

small test to explore the ability of toolchains to vectorize code and the resulting performance

representative of many scientific or engineering applications

- Simple: `y[i] = 2*x[i] + 3*x[i]*x[i]`
- Predicate: `if (x[i]>0) y[i]=x[i]`
- Math functions:
  - Reciprocal
  - Square root
  - Exponential
  - Sine
  - Power
- Gather: `y[i] = x[index[i]]`
- Scatter: `y[index[i]] = x[i]`

Compilers: GNU, Arm, Cray, Fujitsu, Intel

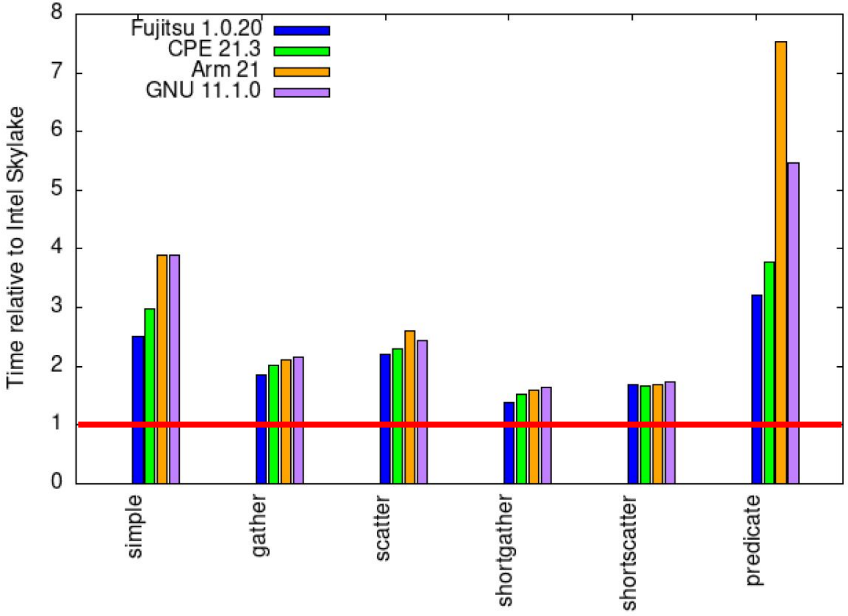
# Compiler and math library vectorization

- sizes of working vectors were adjusted to collectively fill the L1 cache
- gather/scatter: the index vector was constructed as a random permutation of the entire index space
- short gather/scatter: index vector was constructed by randomly permuting within 128 byte windows

- Results on A64FX (1.8GHz) vs Intel Skylake (Xeon Gold 6140, 2.1GHz base, 3.7GHz boost)
- The clock speed ratio leads to an expected circa 2x ratio of runtime between A64FX and Skylake

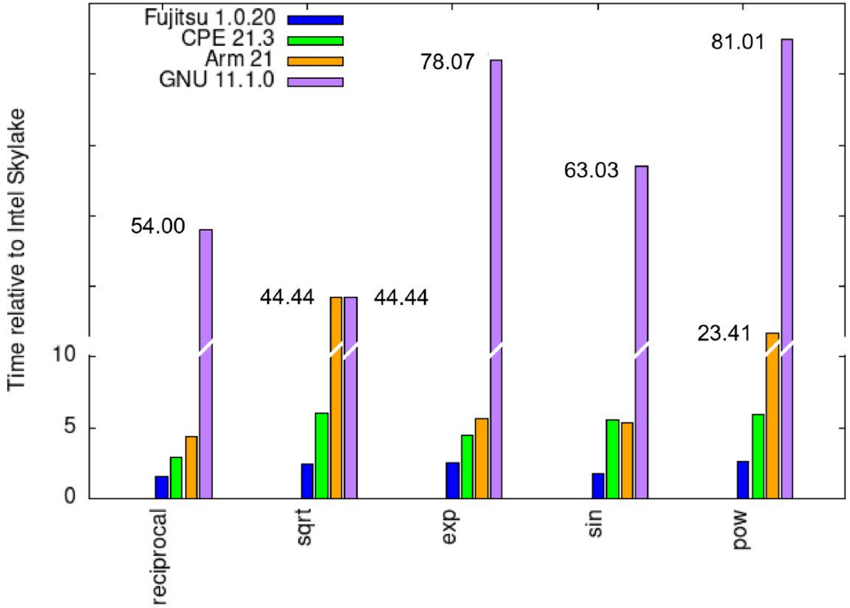
| Compiler | Version | Flags |
|---|---|---|
| Fujitsu | 1.0.20 | -Kfast -KSVE -Koptmsg=2 |
| Arm | 21 | –std=c++17 -Ofast -ffp-contract=fastv -ffast-math -Wall -Rpass=loop-vectorize -march=armv8.2-a+sve -mcpu=a64fx -armpl -fopenmp |
| Cray | 10.0.2 | -O3 -h aggress,flex_mp= tolerant,msgs,negmsgs,vector3,omp |
| GNU | 11.1.0 | -Ofast -ffast-math -Wall -mtune=a64fx -mcpu=a64fx -march=armv8.2-a+sve -fopt-info-vec -fopt-info-vec-missed -fopenmp |
| Intel | 19.1.2.254 | -xHOST -O3 -ipo -no-prec-div -fp-model fast=2 -qopt-report=5 -qopt-report-phase=vec -mkl=sequential -qopt-zmm-usage=high -qopenmp |

# Compiler and math library vectorization




- Performance: Fujitsu > Cray > Arm / GNU
- Fujitsu ~2 slower than Intel Skylake except for the predicate operation (~3) and short gather (~1.5)
- A64FX Microarchitecture Manual indicates loads of pairs of elements of a gather operation fit within an aligned 128-byte window, resulting in a 2-fold speed up
- No acceleration is indicated for scatter operations

# Compiler and math library vectorization




- Intel, Fujitsu, Cray and ARM compilers vectorized all loops
- GNU compiler did not vectorize exp, sin, and pow
- Sqrt: Arm and GNU compilers selecting the SVE FSQRT instruction (which on A64FX is blocking with a 134 cycle latency for a 512-bit vector)
- Sqrt: Cray and Fujitsu compilers employ Newton algorithm
- Accuracy not evaluated

# Key Findings

- Compiler makes a huge performance difference
- In general Cray and Fujitsu deliver best performance
- Arm delivers competitive performance and fully support current language standards
- GCC optimizes for SVE and A64FX and sometimes generates best performance, but lack of vector math library

# Ongoing work

- Investigating PyTorch and TensorFlow
- Porting commonly used science codes

  (VASP, QuantumEspresso, OpenFOAM, LAMMPS, etc.)
- Advance testbed projects to production projects
- Prepare for XSEDE allocation and ACCESS transition

# Get in Contact


**Acknowledgement:**

- The whole Ookami team
- NSF (grant OAC 1927880)


**www.stonybrook.edu/ookami**


eva.siegmann@stonybrook.edu