Big Orange Bramble

August 09, 2016

Overview

- Raspberry Pi
- Enclosure
- Daughter Card
- Monitor Node
- Power Supply
- LDAP
- Slurm
- NFS
- OrangeFS
- HPL
- SPH
- PiBrot
- Numeric Integration
- Parallel Pi
- Monte Carlo
- FDS
- DANNA

Hardware Overview

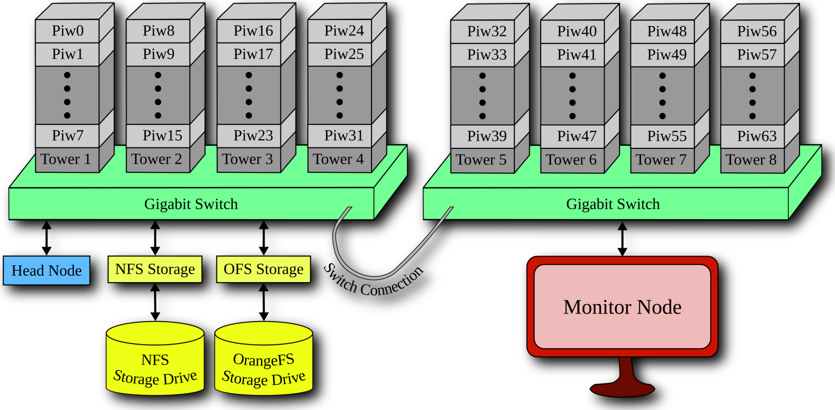


Figure 1: Hardware Diagram

Raspberry Pi 3

- Broadcom BCM28337 SOC
  - Quad Core Cortex A53 (AArch64, ARMv8 compatible)
  - VideoCore IV GPU
- 1GB LPDDR2 RAM
- 10/100 Ethernet, 4 USB 2.0 Ports

Figure 2: Raspberry Pi 3

Enclosure

- Rack Enclosure on Casters
- 2 custom Plexiglass boxes with 3D printed corner brackets house the worker nodes (dimensions: $17in \times 9.25in \times 8.75in$).
- Multiple rack shelves to hold switches and power supplies.
- Pull-out shelf for computer keyboard.

Enclosure

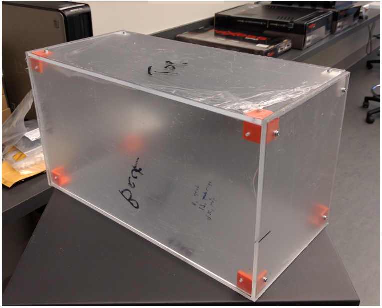
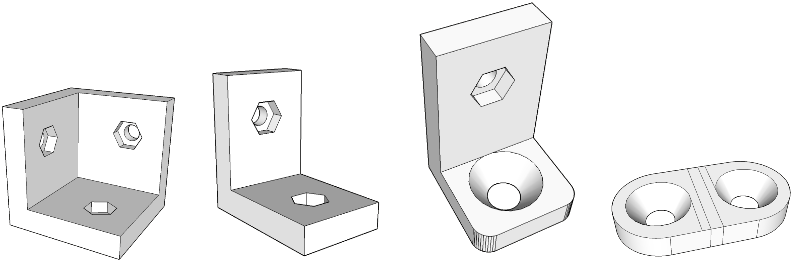


Figure 3: Custom Box

Figure 4: Rack with Casters

Enclosure



(a) Corner Brace (b) L Brace (c) Fan Brace (d) Fan Connector

Figure 5: 3D Printed Parts

Enclosure

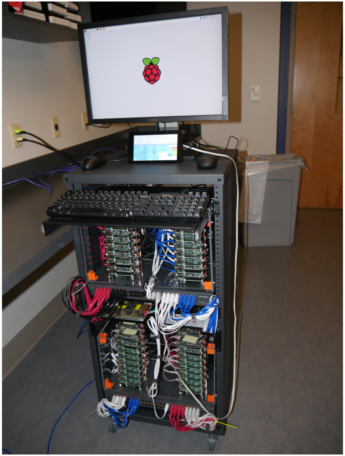


Figure 6: Filled Enclosure Front

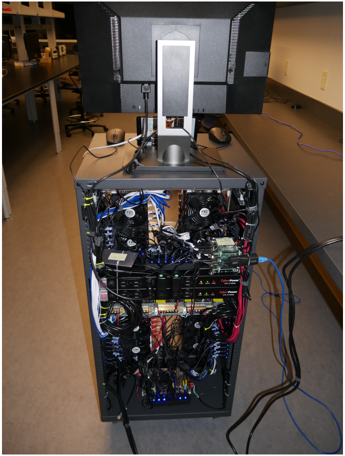


Figure 7: Filled Enclosure Rear

Daughter Card

- Needed a way to measure power input to nodes.
- Convert analog measurements to digital packets.
- Send information to a Monitor Node.

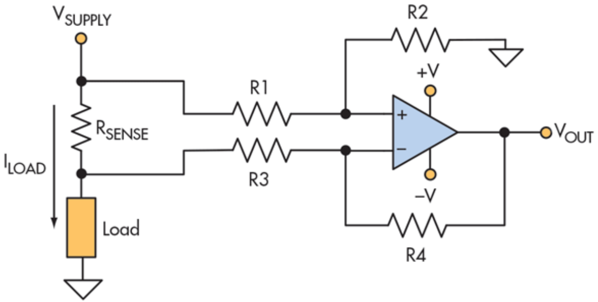


Figure 8: Current Sense Technique

Daughter Card

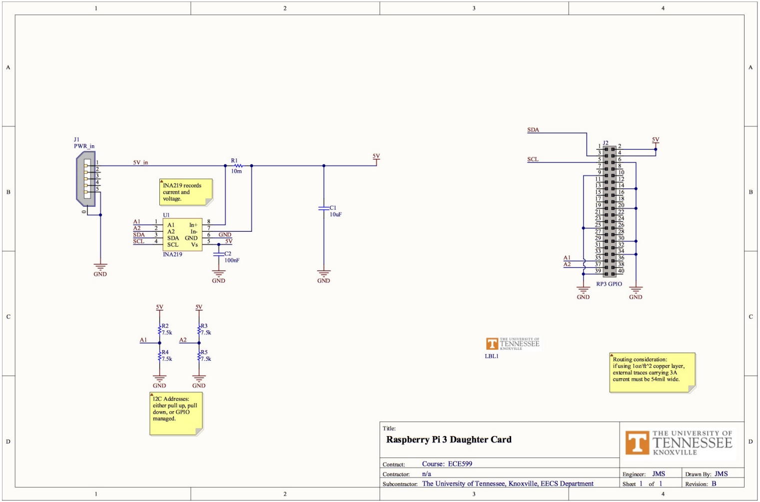


Figure 9: Daughter Card Schematic

Daughter Card

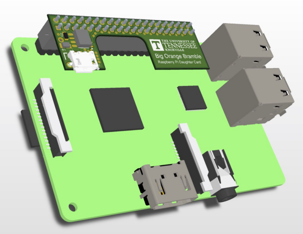


Figure 10: Daughter Card 3D Render

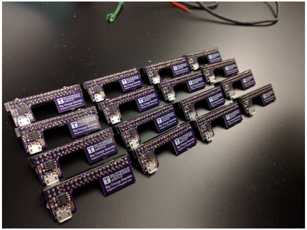


Figure 11: Sixteen Populated Daughter Cards

- Altium Designer
- OSH Park
- Tweezers and Patience

Daughter Card

- Adafruit has a C++ library for the INA219 that was originally used
- Someone had created a Python port that was much easier to integrate into our usage scenario
  - getShuntVoltage_mV()
  - getBusVoltage_V()
  - getCurrent_mA()
- Originally daughter cards connected through Python Paramiko library ssh connections for ease of access
  - This was slow and not the most easily expandable solution
- Now the monitor node handles communication with all daughter cards

Monitor Node Backend

- Separate Raspberry Pi with Touchscreen
- Python monitoring script runs as a service on each node
- Each node sends:
  - CPU temperature
  - CPU load
  - CPU frequency
  - SoC core voltage
- Nodes with daughter cards also send:
  - Supply current
  - Supply voltage
- Information is sent via UDP packets
- Information sent from nodes every 2 seconds

Monitor Node GUI

- Monitoring GUI implemented in Python and GTK and Glade
- GUI can show a map of any of the monitored metrics
- Shows min/max of the measurement metrics

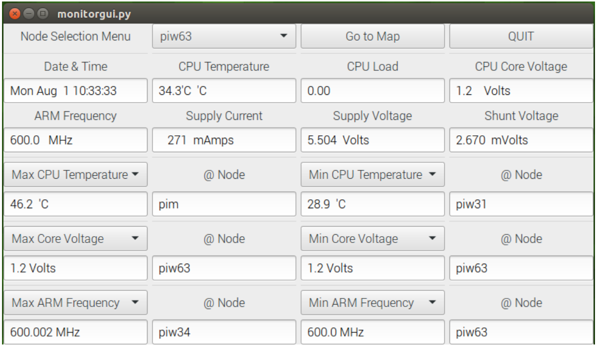


Figure 12: Monitor Gui

Monitor Node GUI

- Monitoring GUI implemented in Python and GTK and Glade
- GUI can show a map of any of the monitored metrics
- Shows min/max of the measurement metrics

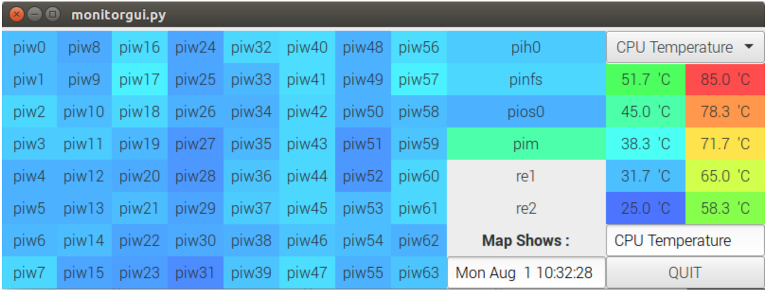


Figure 13: Monitor Map

Power Supply

- Discovered power delivery issue when HPL tests showed CPU frequency throttling
- Voltage dropout was primarily due to cable loss from cables and board-level power management devices (up to $1.5\,\Omega$)
- Power distribution system has 10 Switch Mode Power Supply (SMPS) units rated at 20 A current capacity at 5 V (100W)
- Each SMPS drives a single 7-port USB hub, modified to accommodate approximately 4A per port.

USB Hub Mods

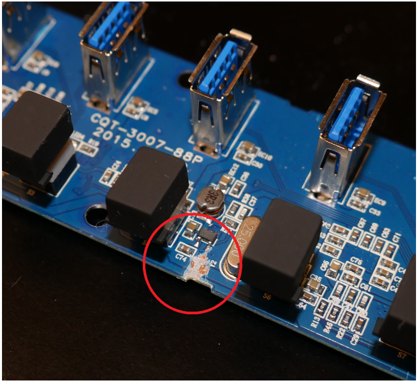


Figure 14: Sliced Power Bus

- 100mil trace only sufficient for up to 4A current load.
- Find optimum location to split trace in order to fan out current load.
- Slice trace with repeated scoring of Exacto-knife.

USB Hub Mods

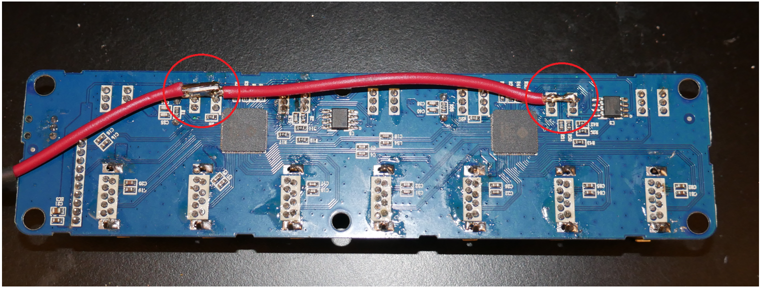


Figure 15: Power Rail Connections

- Choose fanning locations and solder 16AWG stranded wire to points.

USB Hub Mods

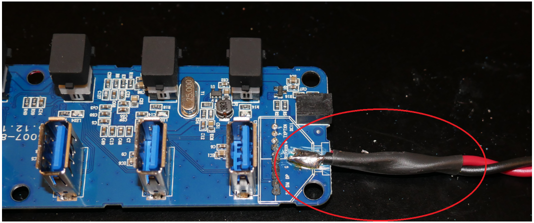


Figure 16: Ground connection and Heat-shrink

- Remove solder mask by scoring from Printed Circuit Board (PCB) to reveal Ground (GND) Plane.
- Solder 16AWG stranded wire to exposed GND plane.
- Twist pairs to establish electromagnetic coupling between transmission and return path.

USB Hub Mods

Success!

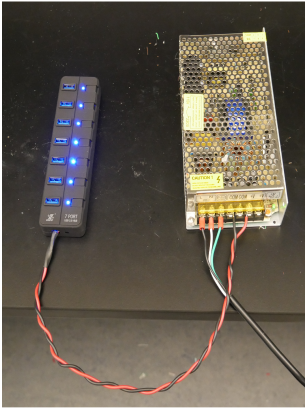


Figure 17: Power Supply

Software Overview

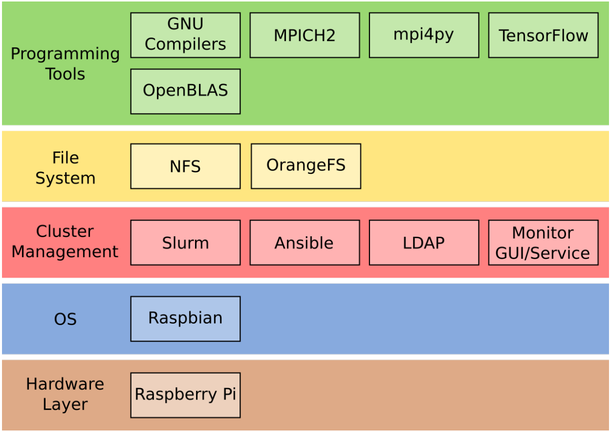


Figure 18: Software Stack

LDAP

- Used to store Users and Groups
- Made easier to use with LDAP Scripts

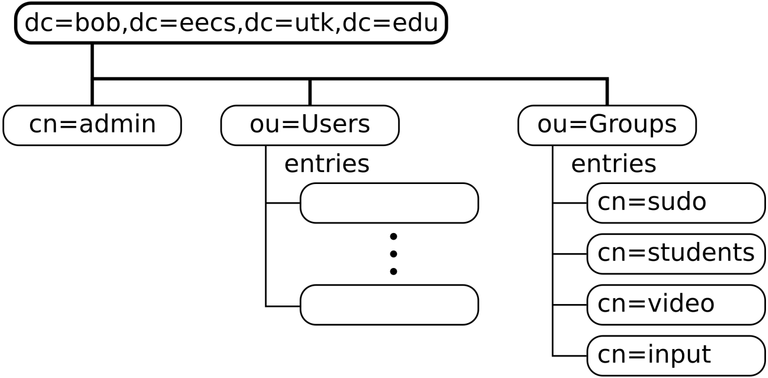


Figure 19: LDAP Structure

Slurm

- Used for resource management

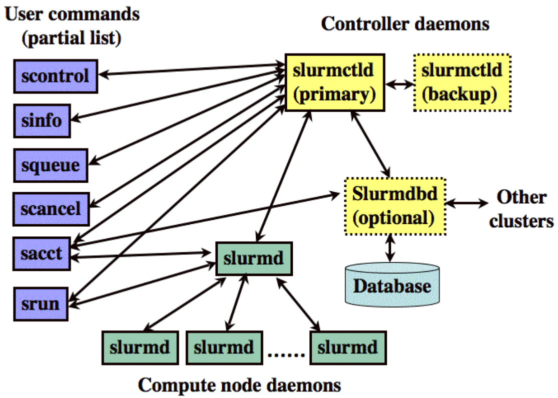


Figure 20: Slurm Architecture

NFS

- Network File System
  - Distributed file system Protocol
- Native component of the Linux Kernel
  - Current version NFSv4
- MPI and NFS
  - Message Passing Interface only supported under no attribute caching disabled
- NFS on BOB
  - Only used for storing users home directories
  - does not support MPI

What is OrangeFS?

- Parallel Virtual File System
- Object based design
- Client/Server Architecture
- Metadata and data services
- BMI - TCP/IP network communication

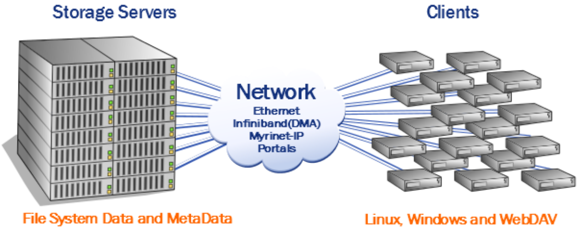


Figure 21: File Distribution

How OrangeFS Works

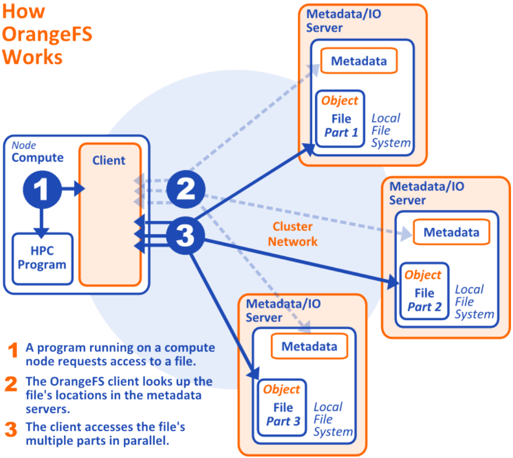


Figure 22: File Distribution

OrangeFS on B.O.B.

- 1 Server
  - Mounted to 4 TB external hard drive
  - Provides highest throughput given 100Mbit/s Ethernet
  - 3 more servers available to integrate
- 64 clients
  - FUSE
  - MPICH2
  - 262,144 Byte transfer buffers with 8 buffers per bulk transfer
  - Metadata and data files synchronized with every write operation
  - Uses a thread based implementation of Asynchronous IO
  - Additional 64 GB of unused storage space available on SD card of each client
- Provides no methods of security

OrangeFS vs NFS

5.6 GB file, 1 MB block, 8 Nodes/32 Tasks, POSIX API

Xfer Size	NFS	OFS
10 KB	10.41	10.78
100 KB	9.16	11.46
1 MB	9.65	11.12

Table 1: Read Rate (MB/s)

Xfer Size	NFS	OFS
10 KB	11.42	9.73
100 KB	11.25	11.31
1 MB	11.34	11.44

Table 2: Write Rate (MB/s)

5.6 GB file, 1 MB block, 64 nodes/256 tasks, POSIX API

Xfer Size	NFS	OFS
10 KB	8.4	5.64
100 KB	8.7	6.96
1 MB	8.5	7.85

Table 3: Read Rate (MB/s)

Xfer Size	NFS	OFS
10 KB	11.59	10.05
100 KB	11.57	11.51
1 MB	11.64	11.54

Table 4: Write Rate (MB/s)

Big Orange Bramble

August 09, 2016

HPL

- High Performance Linpack is a benchmark for clusters
- Created here at the University of Tennessee
- Solves a dense linear algebra system (highly parallel process)
- Used to determine the Top 500 supercomputers in the world
- Requires ARM optimized BLAS library
  - ATLAS looks good on paper, not effective in practice on ARMv7 due to the compiler’s inability to automatically vectorize
  - OpenBLAS offered a hand tooled ARMv7 VFP implementation which avoids the compiler issue

HPL Algorithm

- $Ax = b$ solved by LU Decomposition
- Matrix is of order N, divided into submatrices of order NB
- Process grid of P rows by Q columns

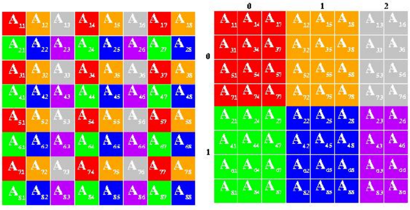
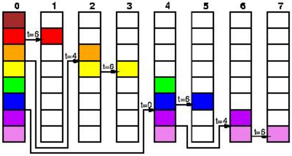


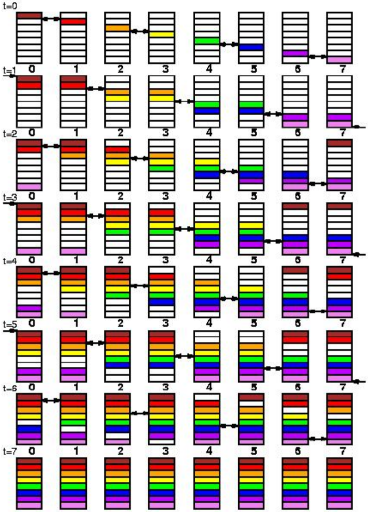
Figure 23: HPL Matrix

HPL Parameters

- Matrix Size
  - $N = \sqrt{10500^2 \times \text{nodes}}$
  - Maximize matrix size while still fitting into RAM
- Block Size
  - $NB = 100$
  - Interesting performance implications
- Broadcast Algorithm
  - Bandwidth Reducing variant (Lng)
- Process Grid $P \times Q$
  - Mostly aimed for square grids
- Lookahead Depth
  - Used $DEPTH = 0$
  - Enabling caused performance regression

HPL Broadcast

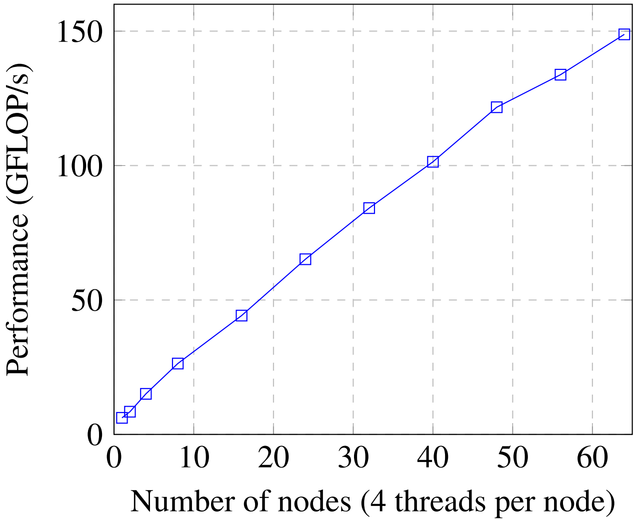




HPL Results

- R_{peak} for the cluster is 441.6 GFLOP/s
- R_{max} for the cluster is 148.8 GFLOP/s
- Scaling Efficiency at 64 nodes is approximately 37%
- Generally poor scaling as the node count increases due to low interconnect bandwidth (100 Mbit Ethernet)
- Fastest in the world in June of 1994
- Top 500 in the world in June of 2002

HPL Results



Performance (GFLOP/s)
0
50
100
150
0
10
20
30
40
50
60
Number of nodes (4 threads per node)

HPCG

- High Performance Conjugate Gradient
- Complementary to HPL to evaluate performance
- Greater emphasis on memory access speed
- Intends to model more realistic workloads, not peak performance
- Provides a “lower bound” to go with HPL’s “upper bound” on performance
- Solves $Ax = b$ with an sparse matrix conjugate gradient method
- BOB achieved 5.08 GFLOP/s on reference implementation

HPCG Algorithm

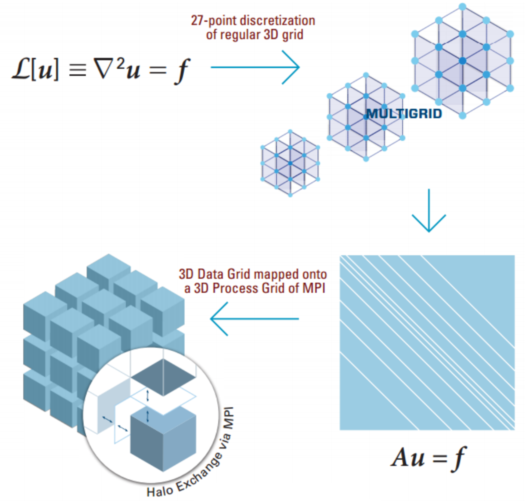
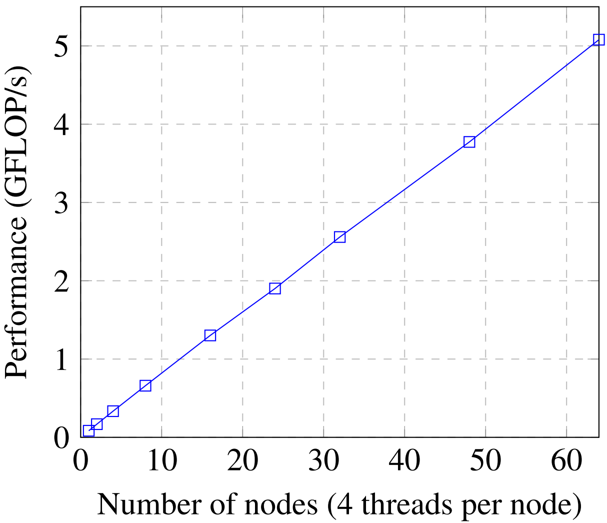


Figure 24: HPCG Algorithm Overview

HPCG Results



SPH

- In multi-particle simulations, interactions can be approximated with local only interactions.
- By distributing the particles over many nodes, an entire scene can be calculated with low latency.
- One master node manages the particles, while the remaining $N - 1$ nodes perform interaction calculations.

PiBrot

- Given $c \in \mathbb{C}$, c can be verified to be in a given Mandelbrot Set.
- For a given display area, each pixel is treated as a complex number c.
- Distributing the rows across multiple nodes allows for parallel Mandelbrot testing.
- PiBrot uses 1 node on the left render and $N - 2$ nodes on the right render.

Numeric Integration

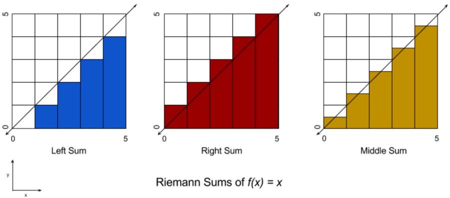


Figure 25: $f(x) = x$ from $[0, 5]$

- Area under curve can be approximated using Riemann Sums
- This application uses right Riemann Sums
- Adapted from Tiny Tiny Numeric Integration program

Numeric Integration: How It Works

1. User inputs function, domain, and # samples
2. MPI gets # available cores
3. Width of each rectangle calculated
4. Domain start and end points calculated for current core
5. Areas are calculated
6. MPI sums and reduces all areas into one variable
7. Error is calculated using scipy.integrate.quad()

Numeric Integration: Example

```
#DEFINE YOUR FUNCTION HERE!
def func(x):
        y = numpy.cos(x)
        return y

def main():
        #SET THE DOMAIN BEGINNING AND END
        xStart = 0.0
        xEnd = 1.0 * scipy.pi

        #SET NUMBER OF SAMPLES (rectangles & accuracy)
        samplesPerRank = 10
```

Figure 26: Parameters defined by user

```
On 128 nodes/cores: actualArea = 0.00000 riemannArea = 0.00245  error = -0.0024543693 time = 0.28306
```

Figure 27: Output printed to command line

Parallel Pi

- What?
  - Approximate the value of pi in parallel.
- Why?
  - Demonstrate BOB’s ability to efficiently scale embarrassingly parallel tasks.
  - Discover any remaining issues with mpi4py, Slurm.

Parallel Pi: How It Works

- First approach: Leibniz Formula

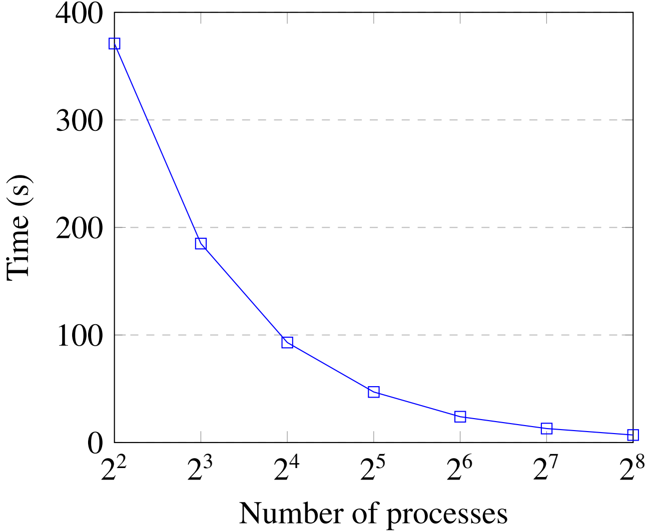
$$\pi = 4 \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1}$$

- Second approach: Bailey-Borwein-Plouffe Formula

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

- BBP written in Python with mpi4py, managed using Slurm batch script

Parallel Pi: Scaling Results



Parallel Pi: Scaling Explained

- Amdahl's Law

$$S = \frac{1}{(1-P) + P/N}$$

- P $\approx$ 99.871%
- Solving S for 256 processes (64 nodes) gives us S = 192.64
- Therefore, we can expect a speedup of approximately 193 for 256 processes

Monte Carlo

- Monte Carlo simulations utilize random sampling to create output for analysis
  - Example: Rolling a dice 100,000 times
- Driver program that parallelizes multiple runs of user created executable
  - mc_batch.py
- Simple, user friendly GUI
  - mc_gui.py

Monte Carlo Cont'd

- Currently only runs C/C++ and Python executables
- User parameters

  Required

  1. Input executable file path
  2. Output file name
  3. Number of runs

  Optional

  – Maximum run time
  – Deadline
  – Number of CPUs
- A batch file is created and submitted for SLURM to manage

Monte Carlo Example

```
#!/bin/bash

#SBATCH -N 8
#SBATCH --job-name=montecarlo_dice_roll
#SBATCH --output=d.txt
#SBATCH --open-mode=append
#SBATCH --cpus-per-task=1
#SBATCH -e montecarlo_dice_roll_err.txt
srun -n 4 python dice_roll.py
srun -n 32 python dice_roll.py
srun -n 32 python dice_roll.py
srun -n 32 python dice_roll.py
```

Figure 28: Batch file produced with Monte Carlo application

FDS

- Fire Dynamic Simulator is a large-eddy simulation (LES) code for low-speed flows, with an emphasis on smoke and heat transport from fires.
- Takes advantage of parallel processing by dividing models up into a series of meshes
- Each mesh interacts only with the meshes immediately spatially adjacent to it, monitoring things like air flow and heat transfer
- It is best if one mesh is assigned to one processor, although it is possible to assign multiple meshes to a processor

FDS Previous Work

- The work of Donald Collins of The University of Tennessee served as the basis for our work

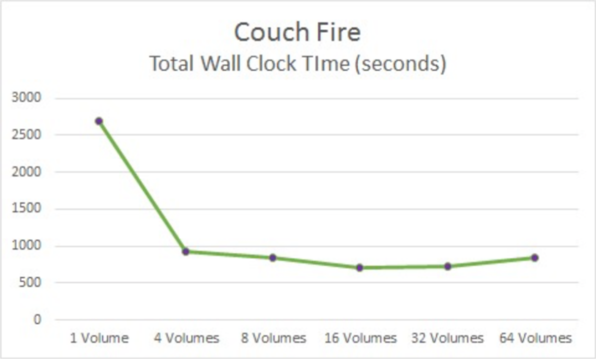


Figure 29: Source: ”Dividing and Conquering Meshes within the NIST Fire Dynamics Simulator (FDS) on Multicore Computing Systems”

FDS Results

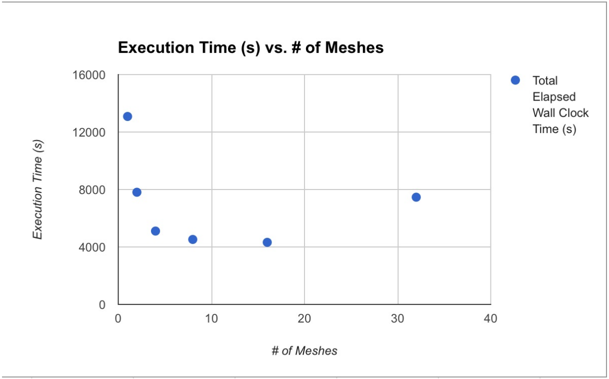


Figure 30: Timing results of tests run on BOB in which a room fire was modeled using a varying number of meshes. It becomes clear that shortly after 16 meshes the overhead begins to outweigh the performance gains.

DANNA

- Dynamic Adaptive Neural Network Arrays
- Neuromorphic Architecture
- Configurable grid of elements (neurons or synapses) with varying parameters
- Fast software based simulator available in addition to FPGA implementation
- Networks are generated by Evolutionary Optimization and tested with the simulator for fitness
- Generated a pole balancing network in just over 3 minutes which can survive for 5 minutes of simulation

DANNA

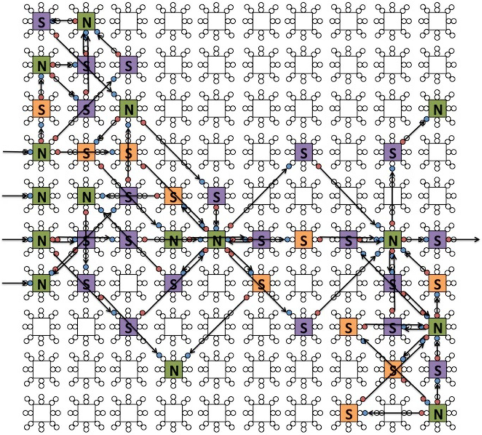
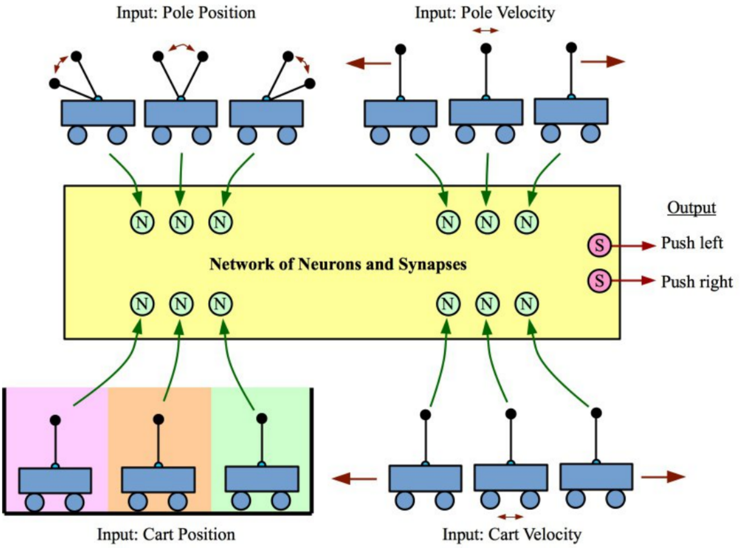


Figure 31: Visualization of DANNA

Danna Pole Balencer



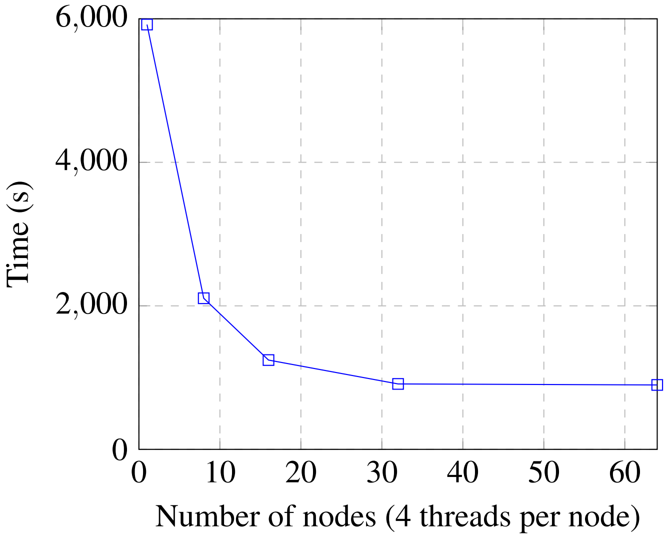
DANNA EO Results

- Compared performance from 1 to 64 nodes
- Not a great benchmark due to random nature of EO
- Uses a Master-Slave work distribution scheme

# Nodes	Distribution Algorithm	Time (s)	Scaling Factor
1	None	5930	1.00
8	Master-Slave	2105.8	2.82
16	Master-Slave	1244.6	4.76
32	Master-Slave	911.8	6.50
64	Master-Slave	897.8	6.61

Table 5: DANNA EO Performance for Pole Balancing

DANNA EO Results



Distributed TensorFlow

- Modern deep neural networks: using large datasets and large amounts of computation to push boundaries of what is possible in perception and language understanding (Large Datasets + powerful models)
- Large-scale parallelism using distributed systems is the only way
- An open source software library for numerical computation using data flow graphs. Developed in Google Brain Team
- Distributed model? Distributed datasets (data parallelism)?

Distributed TensorFlow

- In graph replication vs. between-graph replication
- Synchronized training vs. asynchronized training (Figure 32)

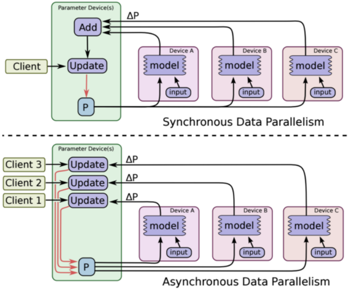


Figure 32: Synchronized training vs. asynchronized training

Distributed TensorFlow: LeNet

- A convolutional neural network based on LeNet is deployed on BOB
- Goal: recognize handwritten numbers (MNIST)

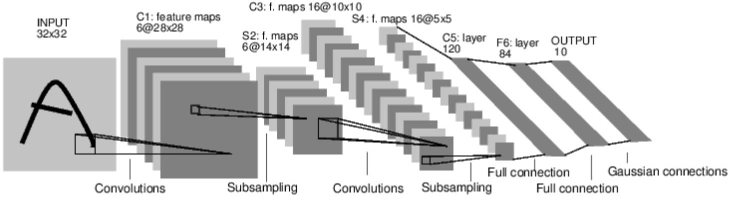


Figure 33: LeNet structure

Work Load

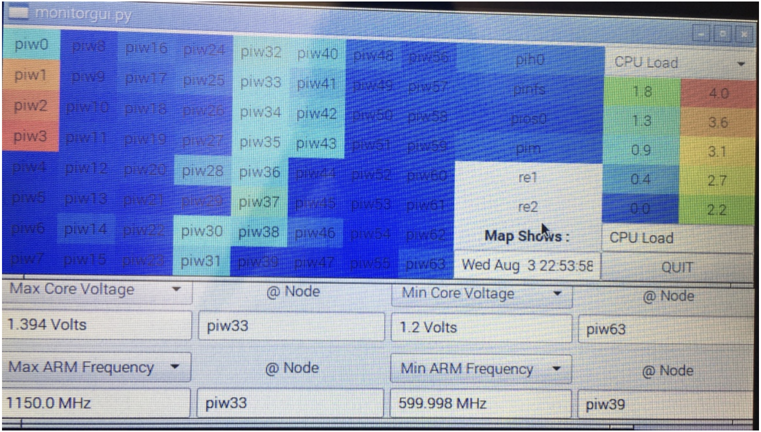


Figure 34: 4 nodes vs 16 nodes

Big Orange Bramble

August 09, 2016

IOR File System Benchmark

- Interleaved or Random (IOR) is a benchmark used for I/O operations on parallel file systems
- POSIX, MPIIO, and HDF5
- Developed by Lawerence Livermore National Laboratory
- Used by National Energy Research Scientific Computing Center (NERSC)
- NERSC8/Trinity Benchmark for sequential reads and writes
  - Fixed 1 MB block sizes
  - 10KB, 100KB, and 1MB transfer sizes
  - Fixed user defined segment count
  - One file per process
  - One shared file
  - POSIX API
  - MPIIO API

MPIIO on NFS and OFS

- 5.6 MB file
- 8 Nodes/32 tasks
- One file per process

File System	Reads (MB/s)	Writes (MB/s)
NFS(noac)	4.21	11.40
OFS	11.67	9.79

Table 6: 10 KB Transfers

Transfer Size	Reads (MB/s)	Writes (MB/s)
10 KB	11.67	9.79
100 KB	11.10	11.30
1 MB	10.44	11.65

Table 7: OrangeFS MPI-IO Read and Write Throughput

8 Client POSIX Test

- 5.6 GB file, 1 MB blocksize

Transfer Size	NFS	OFS
10 KB	10.41	10.78
100 KB	9.16	11.46
1 MB	9.65	11.12

Table 8: Read Rate (MB/s)

Transfer Size	NFS	OFS
10 KB	11.42	9.73
100 KB	11.25	11.31
1 MB	11.34	11.44

Table 9: Write Rate (MB/s)

Overall OrangeFS outperforms NFS on 8 clients and 1 server

64 Client POSIX Test

One OrangeFS Server

Xfer Size	NFS	OFS
10 KB	8.4	5.64
100 KB	8.7	6.96
1 MB	8.5	7.85

Table 10: Read Rate (MB/s)

Xfer Size	NFS	OFS
10 KB	11.59	10.05
100 KB	11.57	11.51
1 MB	11.64	11.54

Table 11: Write Rate (MB/s)

Two OrangeFS Servers

Xfer Size	NFS	OFS
10 KB	8.4	11.79
100 KB	8.7	8.93
1 MB	8.5	10.18

Table 12: Read Rates (MB/s)

Xfer Size	NFS	OFS
10 KB	11.59	15.85
100 KB	11.57	21.73
1 MB	11.64	20.81

Table 13: Write Rates (MB/s)

Big Orange Bramble

August 09, 2016