Big Orange Bramble+
BOB and ALICE

December 2, 2016
Overview

Hardware Overview
Software Stack Overview
Enclosure
Daughter Card
Monitor Node
Fan & Reset Controller
Pine64 Issues

Slurm
Accounting
Benchmarking
Facial Recognition
FDS
WRF
Gillespie Algorithm
Hardware Overview

Figure 1: Hardware Diagram
# Software Stack Overview

**Figure 2: Software Stack**

<table>
<thead>
<tr>
<th>Programming Tools</th>
<th>GNU Compilers</th>
<th>OpenCV</th>
<th>mpi4py</th>
<th>TensorFlow</th>
<th>OpenBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specialized Programing Tools</td>
<td>MPICH2</td>
<td></td>
<td></td>
<td>OpenMPI</td>
<td>Cuda</td>
</tr>
<tr>
<td>File System</td>
<td>NFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster Management</td>
<td>Slurm</td>
<td>Ansible</td>
<td>LDAP</td>
<td>Monitor GUI/Service</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Raspbian 32 bit</td>
<td>Debian 64 bit</td>
<td>Ubuntu 64 bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Layer</td>
<td>Raspberry Pi</td>
<td>Pine 64</td>
<td>Nvidia TX1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Enclosure

- Rack Enclosure on Casters
- Pull out shelf for keyboard
- One rack shelf for 32 Pine64s
- 3 rack shelves for NVIDIA TX1s
- 12 PWM controllable 12V Fans

Figure 3: Enclosure
Daughter Card Review

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

Figure 4: Current Sense Technique
Daughter Card Review

Figure 5: Daughter Card Schematic
Daughter Card Updates

- PI-2 port on Pine64 allows for compatibility with BOB DC
- BOB DC did not use pull-up or pull-down resistors on I2C SCL or SDA lines. Simple mod.

Figure 6: Mounted DC

Figure 7: Pull-up Resistors
Daughter Card Firmware

• Same Adafruit C++ library for the INA219 was used
• The Python library overlay for this Adafruit C++ library had to be completely rewritten
  – The original Python library was written by a hobbyist
  – It took advantage of several assumptions, many of which included that this library would only be used with a Raspberry Pi 1 or 2
  – The rewritten library should be more generic and allow for any Pi-like board with some flavor of Linux to utilize its functionality and is made available on our BitBucket account
• Monitor node continues to handle 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

**Figure 8: Main Monitor Page**

<table>
<thead>
<tr>
<th>Node Selection Menu</th>
<th>pine-w0</th>
<th>Go to Map</th>
<th>QUIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date &amp; Time</td>
<td>Fri Dec 2 02:56:23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU Temperature</td>
<td>26.0 °C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU Load</td>
<td>0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU Core Voltage</td>
<td>1.04 Volts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARM Frequency</td>
<td>0.48 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply Current</td>
<td>234 mAmps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>4.992 Volts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shunt Voltage</td>
<td>2.340 mVolts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max CPU Temperature</td>
<td>47.0 °C</td>
<td>@ Node</td>
<td></td>
</tr>
<tr>
<td>Min CPU Temperature</td>
<td>26.0 °C</td>
<td>@ Node</td>
<td></td>
</tr>
<tr>
<td>Max Core Voltage</td>
<td>1.3875 Volts</td>
<td>@ Node</td>
<td></td>
</tr>
<tr>
<td>Min Core Voltage</td>
<td>1.04 Volts</td>
<td>@ Node</td>
<td></td>
</tr>
<tr>
<td>Max ARM Frequency</td>
<td>1150.002 MHz</td>
<td>@ Node</td>
<td></td>
</tr>
<tr>
<td>Min ARM Frequency</td>
<td>0.48 MHz</td>
<td>@ Node</td>
<td></td>
</tr>
</tbody>
</table>

**Table:**

- **Monitor Node GUI**
- **Figure 8:** Main Monitor Page

**Table Data:**

- Node Selection Menu: pine-w0
- Date & Time: Fri Dec 2 02:56:23
- CPU Temperature: 26.0 °C
- CPU Load: 0.10
- CPU Core Voltage: 1.04 Volts
- ARM Frequency: 0.48 MHz
- Supply Current: 234 mAmps
- Supply Voltage: 4.992 Volts
- Shunt Voltage: 2.340 mVolts
- Max CPU Temperature: 47.0 °C
- Min CPU Temperature: 26.0 °C
- Max Core Voltage: 1.3875 Volts
- Min Core Voltage: 1.04 Volts
- Max ARM Frequency: 1150.002 MHz
- Min ARM Frequency: 0.48 MHz
Monitor Node GUI

Figure 9: Heat Map Page
Fan & Pine Reset Controller

- Arduino Mega 2560 can control fan PWM duty cycle, from Daughter Card feedback through Monitor Node.
- Additionally provides wiring to toggle RST on Pine64s

Figure 10: Fans

Figure 11: Arduino Shield for Controller
Fan & Pine Reset Controller

Figure 12: Controller
Pine64 Issues

- Issues with HDMI/DVI Display
- Reboot Issues
- Networking Issues
- Software Difficulties
- Cache Issues
- HPL Issues - Failed Residual Calculation
Slurm Job Pack Investigation

Job Packs sound great
Released in 16.5
Slurm Job Pack Investigation

Job Packs sound great
Released in 16.5

Not in 16.5
Slurm Job Pack Investigation

Job Packs sound great
Released in 16.5

Not in 16.5

Post to Slurm Mailing list
Slurm Job Pack Investigation

- Job Packs sound great
  Released in 16.5

- Not in 16.5

- Post to Slurm Mailing list

- Job packs should be released in 17.02
Slurm Job Pack Investigation

Post to Slurm Mailing list

Job packs should be released in 17.02

Not in 17.02 Pre-release
Slurm Job Pack Investigation

Post to Slurm Mailing list

Job packs should be released in 17.02

Git repository has a job packs branch

Not in 17.02 Pre-release
Slurm Job Pack Investigation

Post to Slurm Mailing list

Job packs should be released in 17.02

Not in 17.02 Pre-release

Git repository has a job packs branch

Can't launch jobs in same COM World
Slurm Job Pack Investigation

Job packs should be released in 17.02

- Not in 17.02 Pre-release

Git repository has a job packs branch

- Can't launch jobs in same COM World

Try different versions of MPI
Slurm Job Pack Investigation

- Job packs should be released in 17.02
- Not in 17.02 Pre-release

- Git repository has a job packs branch
- Can't launch jobs in same COM World

- Need newer GCC
- Try different versions of MPI
Slurm Job Pack Investigation

- Need newer GCC
- Try different versions of MPI
  - Try new MPICH2
Slurm Job Pack Investigation

Need newer GCC

Try different versions of MPI

Try new MPICH2

Can't launch jobs in same COM World
Slurm Job Pack Investigation

- Need newer GCC
- Try different versions of MPI
  - Try new MPICH2
  - Try new OpenMPI
- Can't launch jobs in same COM World
Slurm Job Pack Investigation

Need newer GCC

Try different versions of MPI

Try new MPICH2

Can't launch jobs in same COM World

Try new OpenMPI

Can't launch jobs in same COM World
Slurm Job Pack Investigation

Try new MPICH2

Can't launch jobs in same COM World

Determine Job Packs are not ready

Try new OpenMPI

Can't launch jobs in same COM World
Slurm Job Pack Investigation

Job Packs sound great
Released in 16.5

Not in 16.5

Post to Slurm
Mailing list

Job packs should be
released in 17.02

Git repository has a
job packs branch

Not in 17.02
Pre-release

Can't launch
jobs in same
COM World

Need newer
GCC

Try different
versions of MPI

Try new
MPICH2

Try new
OpenMPI

Can't launch
jobs in same
COM World

Can't launch
jobs in same
COM World

Determine Job Packs
are not ready
Accounting On Alice Investigation

- MySQL is used to store Slurm’s accounting information.
- MySQL apt package has missing dependencies and will not install on the pine 64s.
- Apt has mysql-server-5.6 which can install.
- Learned how to configure Slurm for accounting and how to setup the database for use with Slurm.
- Discovered that Slurm needs to be reconfigured to detect the MySQL installation.
- Slurm can’t detect MySQL without MySQL-dev tools. Main MySQL-dev has broken dependencies and 5.6 does not have dev package.
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 13: HPL Matrix
Revealed L2 cache is likely not functioning properly
HPL - Pine64

Number of nodes (4 threads per node)

Performance (GFLOP/s)

Blue = Pine64, Red = BOB, Orange = TX1
HPL - nVidia TX1

Number of Nodes

Performance (GFLOPs)

Orange = TX1 CPU, Red = BOB
HPL - CUDA notes

- Two variations utilizing CUDA were found
- Github project from David Martin
- nVidia provided code for Intel & Fermi
- GPU offers only 16 GFLOPs theoretical for double precision
- Large overhead associated with both implementations due to memory/data management
- Future goal: HPL for a zero-copy unified memory architecture
- Future goal: Single precision CUDA-enabled HPL (∼500 SP GFLOPs theoretical per GPU)
HPL - Performance Summary

• Pine64 demonstrated poor scaling due to block size and networking
• Currently only getting 47 GFLOPs at 29% scaling efficiency with 32 nodes
• nVidia TX1 demonstrated solid CPU performance
• Achieving 131 GFLOPs at 61% scaling efficiency with 12 nodes
• For reference, BOB achieved 148.8 GFLOPs at 37.7% scaling efficiency with 64 nodes
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
HPCG - Pine64

Blue = Pine64, Red = BOB, Orange = TX1
HPCG - nVidia TX1

Number of Nodes

Performance (GFLOPs)

Red = BOB, Orange = TX1
HPCG - Performance Summary

- Pine64 yet again demonstrates poor scaling
- Achieved 807 MFLOPs at 41.5% scaling efficiency
- Actually a better showing than HPL - achieved 1.7% of HPL
- nVidia TX1 performs somewhat better than BOB - but less so than in HPL
- Achieved 1.472 GFLOPs at 87.15% scaling efficiency (1.1% of HPL)
- For reference, BOB achieved 5.07949 GFLOPs at 94.5% scaling efficiency (3.4% of HPL)
nVidia TX1 CUDA Notes

- Tegra X1 SoC offers very good performance for an embedded system
- GPU is optimized for half-precision performance (1 TFLOP theoretical)
- GPU is ill-suited for double precision tasks
- CUDA should utilize zero-copy memory because the SoC shares its 4GB between the CPU and GPU already
- MPI should then be able to directly access the same memory from the CPU and have good performance
Facial Recognition- Overview

• OpenCV
  – Software package, initiated in 1999/2000
  – Objective: real-time computer vision

• Process
  – Detect faces in images.
  – Detect faces with live video input.
  – Machine learning and facial prediction.
  – Parallelization.
Facial Recognition - Overview

cascade = "cascades/haarcascades/haarcascade_frontalface_default.xml"
c = cv2.CascadeClassifier(cascade)

def open_images(im):
    ri = cv2.imread(im)
    gi = cv2.cvtColor(ri, cv2.COLOR_BGR2GRAY)
    return gi

def detect_faces(im):
    gi = open_images(im)
    flag = cv2.CASCADE_SCALE_IMAGE
    faces = c.detectMultiScale(gi, scaleFactor=1.1, minNeighbors=10, minSize=(100, 100), flags=fl)
    print "Number of detected faces: {0}".format(len(faces))
    return show_image(gi, faces)

if __name__ == "__main__":
    images = ["image.jpg"]
    im = Pool(4).map(detect_faces, images)

• Image processing internally parallelized with TBB library.
• I/O data parallelism with Pool().
Facial Prediction- Overview

• Process:
  – Read in images. (Potentially on the order of hundreds)
  – **Resize each image consistently!!!**
    • Drawback: Undefined effect on initial image resolution.
    • Solution: Minimalize resize factor.
  – Separate into two numpy arrays:
    • Samples: Image vectors.
    • Labels: Assign label to each image vector for classification.
  – Train algorithm(s) on image datasets.
  – Predict on face using trained data.

• Implemented Algorithms
  – Classification/Supervised learning:
    • KNN
    • Random Forest
    • Ada Boost
    • SVM
    • Normal Bayes
Facial Prediction- Results

• Approach

- Image file with image ranges corresponding to each label.

• Results

- Dependent on camera.
- Dependent on training data.
- Successfully predicts on high-resolution images.
- Struggles with low-resolution images.

• Onto parallelization!!!!!
Facial Prediction- Parallelization

• Approaches
  – Point-to-point communication
    • Master process responsible for distributing jobs.
    • Interconnect between nodes too slow.
    • Overhead from constant communication.
  – Dynamic process management
    • Each process gets copy of data.
    • Upon job completion, worker broadcasts to surrounding nodes.
    • Suffers from slow interconnect speeds.
  – Collective communication
    • Slices data into even junks.
    • Scatters among workers.
    • Worker performs processing on individual chunk.
    • Master process gathers all processed images.
if __name__ == "__main__":
    comm = MPI.COMM_WORLD
    size = comm.Get_size()
    rank = comm.Get_rank()
    samples = []
    labels = []
    fl = []
    s = []
    l = []
    
    if rank == 0:
        data = glob("Pics/*.jpg")
        slices = [[[] for i in range(size)] for i, slice in enumerate(data):
            slices[i % size].append(slice)
    else:
        data = None
        slices = None

    sd = comm.scatter(slices, root=0)

    fl = Pool(4).map(detect_faces, sd)

    labels, samples = break_lists(fl)

    s_data = comm.gather(list(samples), root=0)
    l_data = comm.gather(list(labels), root=0)

    kill_process(rank)
Facial Prediction - BOB Results

![Graph showing execution time vs number of nodes for different image counts: 128 Images, 256 Images, 320 Images.](image)

- **Execution Time (sec)**
- **Number of Nodes**

- **128 Images**
- **256 Images**
- **320 Images**
BOB vs ALICE - 128 Images

Number of Nodes

Execution Time (sec)

BOB
ALICE

1 2 3 4 8 12 16 20 24 28 32
BOB vs ALICE - 256 Images

The graph shows the execution time (in seconds) for BOB and ALICE as the number of nodes increases.

- BOB
- ALICE

Execution Time (sec)

Number of Nodes
BOB vs ALICE - 320 Images

![Graph showing the comparison between BOB and ALICE in terms of execution time for different numbers of nodes. The x-axis represents the number of nodes, and the y-axis represents the execution time (seconds). The graph shows that BOB has a consistently higher execution time compared to ALICE.]
Facial Prediction- Conclusions

• BOB
  – Speedup across 64 nodes:
    • 128 images: 11.5x
    • 256 images: 12x
    • 320 images: 11.2x

• ALICE
  – Speedup across 32 nodes:
    • 128 images: 5.4x
    • 256 images: 6.7x
    • 320 images: 7x
    • 384 images: 6.8x
    • 448 images: 7.3x
FDS - Overview

- 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.
Figure 14: FDS results on BOB
FDS - Scaling Results

Figure 15: FDS results on ALICE
WRF - Overview

- Atmospheric modeling system for weather prediction
- National Center for Atmospheric Research, the National Oceanic and Atmospheric Administration, Air Force Weather Agency, and many more
- Domain is divided into grid, grid cells updated concurrently by each worker node
- Primary challenges:
  - compiling WRF and dependencies on BOB
  - Network bottleneck when utilizing large number of worker nodes
- Test case - Hurricane Katrina
WRF - Scaling Results

Execution Time (s) vs. 
# of Nodes (4 Processes Per Node)
Gillespie Algorithm

- Broadly used in chemistry, biology, and economics for stochastic simulations
- Utilizes master equations and associated rates to create a time-based model of reactions in a system
- Implemented with 2 Monte Carlo steps, thus is an expansion of the previous Monte Carlo application on BOB
- Single simulation designated to single core
  - No division of computation or parallelism
  - Limited to 64 computations at any given time on BOB
- Simulations can run indefinitely
Gillespie Algorithm - Steps

1. Initialization: concentration of each species and rates of reaction equations
2. Propensities: calculate the likeliness of each reaction to occur
   - A propensity is calculated by $\mu_i = C_s \times r_i$ for each reactant species (on left side of the arrow)
   - If the sum of propensities ever reaches 0, no more reactions can occur in the system
3. Monte Carlo - Time: time step is chosen using $\Delta t = \mu_{total} \ln k$ where $k \in (0, 1)$ and $\mu_{total}$ is the sum of the propensities
4. Monte Carlo - Reaction: a reaction is chosen randomly, weighted by propensities
5. Update: concentrations are updated based on which reaction is chosen in previous step
Gillespie Example 1

• Species

<table>
<thead>
<tr>
<th></th>
<th>E</th>
<th>S</th>
<th>ES</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>5</td>
</tr>
</tbody>
</table>

• Equations

<table>
<thead>
<tr>
<th></th>
<th>E+S → ES</th>
<th>0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES → E+S</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>ES → E+P</td>
<td>0.25</td>
</tr>
</tbody>
</table>

• Initial Propensities

<table>
<thead>
<tr>
<th></th>
<th>E+S → ES</th>
<th>(10 * 0.4) + (11 * 0.4) = 8.4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ES → E+S</td>
<td>12 * 0.35 = 4.2</td>
</tr>
<tr>
<td></td>
<td>ES → E+P</td>
<td>12 * 0.25 = 3</td>
</tr>
</tbody>
</table>
Gillespie Example 1 Results

![Graph showing concentration over time for different states (E, S, ES, P)].

- **Time (ms)**: 0, 200, 400, 600, 800, 1,000, 1,200
- **Concentration**: 0, 10, 20, 30

Legend:
- **E**: Orange line
- **S**: Green line
- **ES**: Blue line
- **P**: Red line
Gillespie Example 2 Results

![Graph showing concentrations over time with labels E, S, ES, P]
Gillespie Example 3

• Species

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• Equations

<table>
<thead>
<tr>
<th></th>
<th>A → B</th>
<th>0.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → C</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>C → A</td>
<td>0.34</td>
<td></td>
</tr>
</tbody>
</table>

• Initial Propensities

<table>
<thead>
<tr>
<th></th>
<th>A → B</th>
<th>10 * 0.33 = 3.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → C</td>
<td>0 * 0.33 = 0</td>
<td></td>
</tr>
<tr>
<td>C → A</td>
<td>0 * 0.34 = 0</td>
<td></td>
</tr>
</tbody>
</table>
Gillespie Example 3 Results

![Graph showing the concentration over time for three different substances labeled A, B, and C. The x-axis represents time in milliseconds (0 to 140), and the y-axis represents concentration (0 to 10). The graph shows the fluctuations in concentration for each substance over time.]