Swarming SCC17: Final Architecture Proposal for Georgia Institute of Technology

Petros Eskinder, Nicholas Fahrenkrog, Dezhi Fang, Manas George, David Meyer, Jessica Rosenfield, Alok Tripathy
Chirag Jain, Will Powell, Oded Green
{peskinder3, nick.fahrenkrog, dezhifang, xkcd, dmeyer31, jrosenfield3, atripathy8, cjain, will.powell, ogreen}@gatech.edu

Abstract - We briefly describe our finalized hardware and software architecture, justifying why it is well suited for this year's student cluster competition.

I. Hardware Configuration

Below, we describe our hardware configuration and justify why this system is well suited for the competition.

The hardware currently under consideration is heavily based on the two node IBM Minsky system (Power S822LC). Each node is outfitted with two 10-core Power8 CPUs and four NVIDIA Tesla P100 GPUs. We intend to use Infiniband EDR's as our interconnect between the two nodes, and NVIDIA's NVLink interconnect within each node for GPU-GPU and GPU-CPU communication.

A. Strength of Hardware: Why POWER8?

The POWER8 architecture is the first processor to come out of the OpenPOWER consortium, and has been designed for computationally intensive applications. We see this in part with the NVLink protocol, showing how the POWER8 system is optimized to work well with NVIDIA GPUs. A similar synergy is seen in the choice of interconnect which supports some of the fastest interconnects, like Mellanox InfiniBands.

Here is a listing of the advantages we believe the Power8 possesses:

- **Fast Memory Subsystem**: The Power8's biggest attractor's is its very fast memory subsystem. The POWER8 memory subsystem series shows consistently high memory bandwidth (∼90GB/s) on the Stream bandwidth benchmark, staying in the high nineties even as the number of threads varies from 20 to 80 [1]. These numbers were obtained using the GCC compiler, which is free and much easier for most developers to obtain and integrate into their toolchains, as opposed to vendor-specific compilers, and is therefore more representative of real world performance. The POWER8 processor also has big, high-bandwidth caches at each level of abstraction, further strengthening the memory subsystem.
- **SMT**: The per-core 8-way simultaneous multi-threading provides an incredible boost to memory-heavy applications, allowing the processor to hide memory latency over many threads. This means improved performance in applications like graph traversal. Thread scaling is further helped by support for hardware transactional memory, which removes the overhead of locking mechanisms in software.
- **Embedded NVLink**: Currently, POWER is the only architecture that supports the NVLink protocol, providing fast data transfer between GPUs. NVLink supports transfer rates 5 and 12 times faster than the more common PCIe3 protocol. Combined, the four NVLink ports on the Minsky system provide a 80GB/s full duplex link between the GPUs, minimizing the bottleneck presented by CPU to GPU transfers.
- **CAPI**: The Coherent Accelerator Processor Interface allows for tighter integration between the POWER8 processors and the P100 accelerators, allowing for better heterogeneous performance. It accomplishes this through two means: (1) providing a unified virtual memory space; meaning that the accelerators and the CPUs can use the same memory addresses, reducing device driver overhead, (2) permitting accelerators to behave like normal threads, reducing overhead with respect to cache coherency, as that is now taken care of in hardware.

Overall, these system features position our team to vastly outperform the competition with memory bound applications like HPCG. Coupled with our powerful Tesla P100 GPU's, described, we expect to be strong contenders during this year's student cluster competition.

B. Strength of Hardware: The GPU Advantage

When designing our supercomputing cluster, we made several key decisions regarding (1) the composition of each node and (2) the number of compute nodes.

Our preliminary literature survey suggested that using GPUs as computation accelerators provided significant performance improvements at a lower cost and power than adding another purely CPU-based node. For instance, Matsuoka et al. [2] showed that GPU's accounted for 66% of their LINPACK performance while only accounted for 15% of their total power consumption.

Consequently, we elected to build a cluster consisting of a small number of nodes, each based on a combined CPU/GPU platform.

Our experimental results corroborate our literature survey. Although at the time of writing, only two of our eight P100 GPU's have arrived, we have been able to achieve dramatic speedups using GPU optimized HPL and HPCG binaries. For example with HPL, each GPU, at its peak usage, provides 4-5 TFLOP/s.

Additionally, we have found that these GPU based speedups have *not* come at a significant hit on power efficiency. From our experiments with HPL, we found that each GPU consumes 240-270 Watts of power when active, and 30 Watts when idle. In contrast, we found that one 10-core Power8 CPU consumes

roughly 250-280 Watts, that is while providing a *fraction* of the computational benefit.

II. System Software

We describe and justify our selection of operating system, file system and toolchains.

A. Operating System

We are using CentOS 7. The Power8 system supports two operating systems: Red Hat Enterprise Linux (RHEL) 7.1+ and Ubuntu 16.04+ [3]. Although we are more experienced with Debian-based systems, several key libraries, such as recent versions of ESSL, only offer support for RHEL-based systems.

B. File system

For our file system, we will be using GPFS, a parallel file system provided by IBM, as it provides the performance characteristics matched by few file systems. We felt it was a good choice because our sponsor, IBM, as well as numerous Georgia Tech researchers have ample experience with tuning the performance of GPFS.

C. Toolchain

As far as our toolchain, our hardware guided our selections. In general, we elected to use vendor specific tools, as they are clearly fine-tuned for our machines. Since our machine consists of IBM Power8 CPUs with NVIDIA P100 GPUs, this translated to IBM's proprietary toolkit and NVIDIA's CUDA toolkit.

- **Compilers**: We are using standard GNU compilers as well as IBM's XL C/C++ compiler and CUDA C/C++.
- **Math Libraries**: For mathematical applications like LINPACK, we use the Engineering and Scientific Subroutine Library (ESSL). This is IBM's vendor-specific BLAS library.
- **Message Passing Interface**: We intend to use both OpenMPI and MVAPICH. Both MPI implementations are CUDA-aware, and work with InfiniBand. Initially, we had expected to use IBM Spectrum MPI. However, we found that support for Power8 systems was dropped.
- **System Monitoring**: To monitor our system for good load balancing, we use *Datadog*, a monitoring service. To monitor our GPUs' performance, we are using the NVIDIA visual profiler as well as `nvidia-smi`.
- **Profiling and Optimization**: We will use a variety of miscellaneous tools to fine tune the performance of our system. For instance, to diagnose performance bottlenecks (e.g. communication latency, thread creation) in MPI and OpenMP applications, we will use *Allinea*, an OpenMP and MPI profiler. For other tasks less suited for Allinea, for example, CPU intensive tasks executed on a single machine, we will use Linux performance tools such as `perf`. We also plan on using Flame Graphs as a way of guiding our optimization efforts.

III. JEROME: Our Reproducibility Stack

One area where we have invested considerable attention is with automating our system environment. We make use of various tools so that we can deploy our system and be full functional within a few commands. We call these set of tools, Jerome.

A. Chef

Bootstrapping and provisioning reproducible clusters.

Extensively used in industry, Chef is a provisioning tool often used to manage the software available on a cluster.

With our chef configuration, we are able to quickly set-up a machine and load it with our desired environmental state, i.e. all relevant libraries and tools installed and loaded.

As far as our workflow, we host our chef configuration in a git repository. Team members can make changes to the configuration of the cluster via submitting Pull Requests. Configurations are then converged into each of the hosts. This process is *idempotent* and *revertible*. Consequently, we are able to design reproducible experimentations of software configurations.

We are also able to make use of chef on the cloud. For example, through chef, we are able to share a set of common tools on both our Power 8 cluster and our cloud clusters.

B. Environment Module

Managing conflicting libraries and toolchains.

To ensure reproducibility, we package each library and toolchain component into an environment module.

For example, some applications can only be compiled in certain versions of GCC, whereas other applications can benefit from Power 8 specific optimizations present in IBM XLC. As its non-trivial to setup all the environment variables required to switch between conflicting versions, we use environment modules to automate this process.

C. RPM Package and Artifact Store

Version-controlling system components.

To avoid recompiling things when deploying to a new machine, we perform three major actions: (1) version control each RPM package, (2) store dependency and "file ownership" info for each RPM module, and (3) publish packages into our internal RPM repository hosted on Amazon Simple Storage Service (S3).

D. Fabric

Managing machine state

We have a service discovery system that indexes each of our machines, storing informations such as IP address and architecture. We also have a collection of tasks, written with the help of Fabric, to help manage the state of machines. (Convergence, Updating Module Config, etc.). Combining these parts, we can easily manage all of the machines in a cluster via a simple command-line tool. For example

```
fab converge -R power8
```

converges all Power8 machines to the latest configuration.

```
fab set_chef_branch:test-intel-cuda
        converge -R intel
```

converges all Intel machines to the branch "`test-intel-cuda`" for testing configuration changes made in that branch.

REFERENCES

[1] J. D. Gelas, "Assessing IBM's power8, part 1: A low level look at little endian," Tech. Rep.

[2] S. Matsuoka, T. Aoki, T. Endo, A. Nukada, T. Kato, and A. Hasegawa, "GPU accelerated computing—from hype to mainstream, the rebirth of vector computing," *Journal of Physics: Conference Series*, vol. 180, no. 1, p. 012 043,

[3] (2015). Supported linux distributions for power8 linux on power systems.