Automatic NUMA Characterization using Cbench*

Ryan Braithwaite
Los Alamos National Laboratory
Los Alamos, NM, USA
rkbrait@lanl.gov

Wu-chun Feng
Dept. of Computer Science
Virginia Tech
Blacksburg, VA, USA
feng@cs.vt.edu

Patrick McCormick
Los Alamos National Laboratory
Los Alamos, NM, USA
pat@lanl.gov

ABSTRACT

Clusters of seemingly homogeneous compute nodes are increasingly heterogeneous within each node due to replication and distribution of node-level subsystems. This intra-node heterogeneity can adversely affect program execution performance by inflicting additional data-access costs when accessing non-local data. In this work-in-progress paper, we present extensions to the Cbench Scalable Testing Framework for analyzing main memory and PCIe data-access performance in modern NUMA architectures. The information provided by this tool will be of use for task scheduling, performance modeling, and evaluation of NUMA systems.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)

Keywords

NUMA, Benchmarking, System Analysis

1. INTRODUCTION

Non-uniform memory access (NUMA) architectures are the de facto standard for servers today. The wholesale shift toward the replication and distribution of system resources is driven by the need to increase memory and I/O bandwidth to satisfy more cores simultaneously accessing data. In NUMA systems, resources local to a processor exhibit the uniform access bandwidth and latency to which programs that are designed for uniform memory access (UMA) systems are accustomed. However, resources that are remote to a processor may be subject to significantly worse bandwidth and latency data-transfer characteristics. Differences in the designs of NUMA architectures may result in substantially different data-access performance for each architecture.

When non-local resources are accessed frequently, data-access latency and bandwidth in modern NUMA architectures can significantly affect program performance. For applications that are sensitive to data-access performance, it may be critical for a system designer or application developer to understand the NUMA characteristics of the system in which the program is executed so as to avoid remote data accesses as much as possible.

We propose the use of empirical micro-benchmark data to provide data-access performance information to assist with NUMA system analysis. To run these tests in a deterministic and automated fashion, we present the addition of NUMA testing to the Cbench Scalable Testing Framework [1] [8].

The purpose of this paper is to present the characterization of data-access performance in modern NUMA server architectures using Cbench. The report generated by our new tool provides a straightforward method for comparing data-access attributes in NUMA systems. This knowledge may be used to improve application behavior in NUMA systems, improve system configurations, model the performance of applications in specific NUMA systems, and be used by run-time schedulers to improve program execution efficiency.

Our contributions are two-fold: *NUMA system characterization* and the initial implementation of *NUMA data-access performance extensions to the Cbench Scalable Testing Framework*. For the former, we present methods for data-access *bandwidth* characterization in modern NUMA architectures using open-source micro-benchmarks. This characterization analyzes CPU interconnect links and PCIe interconnect lanes to describe the bandwidth capabilities of a NUMA system. For the latter, we present the addition of our NUMA system characterization tests to the Cbench cluster testing framework in order to assist with the analysis of NUMA architectures.

This tool provides a straightforward method for analyzing and comparing data-access performance across thousands of nodes in a cluster or single stand-alone nodes, as necessary. While other NUMA tools analyze only the hierarchy of NUMA memory nodes in a system, our Cbench tool allows for analysis of the hierarchy, performance, and hardware design of a NUMA system.

2. RELATED WORK & BACKGROUND

The characterization of a NUMA system is typically thought of in terms of the hierarchy of memory nodes and the *distance* between nodes. The `libnuma` [4] and `hwloc` [2] projects describe NUMA systems in this manner, including the distance to additional hardware resources such as PCIe-based interfaces for networking and graphics.

Both tools rely on the Linux `/proc` and `/sys` kernel filesystems to determine NUMA hierarchy information for a system. The NUMA distance data reported by `libnuma` is derived by the kernel from the BIOS ACPI tables and may be wrong or incomplete. The `hwloc` project interrogates these kernel filesystems to provide the programmer with other architectural details of a system in addi-

* This work was supported in part by NSF I/UCRC IIP-0804155 via the NSF Center for High-Performance Reconfigurable Computing and Los Alamos National Laboratory via the Accelerated Strategic Computing program of the Department of Energy. Los Alamos National Laboratory is operated by Los Alamos National Security LLC for the US Department of Energy under contract DE-AC52-06NA25396.

ICPE'12, April 22-25, 2012, Boston, Massachusetts, USA

tion to the ACPI-based NUMA distance information presented by `libnuma`. The notion of NUMA distance only takes into account which nodes are connected to each other. Modern NUMA designs, such as AMD's Magny-Cours and Interlagos architectures, have variable-width CPU interconnect links and therefore exhibit different bandwidth for links of the same NUMA distance.

To provide the *performance* details necessary to describe these newer systems, we build upon existing tools and libraries by measuring the performance for data-access operations between memory nodes and between memory nodes and PCIe devices, specifically GPUs. Our primary goal in characterizing NUMA systems is to quantify the data-access performance of each type of data access that is possible in a system so that the impact of the hierarchy on a system's performance is better understood.

As a case study, we examine the AMD Magny-Cours two-socket (2P) architecture shown in Figure 1. This system has two sources of

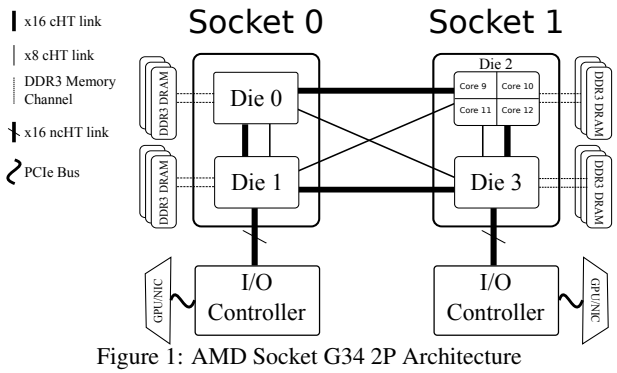

Figure 1: AMD Socket G34 2P Architecture

NUMA performance complexity: the CPU package, consisting of two CPU dies, three levels of cache, and wires in a socket; and the HyperTransport (HT) interconnect links between CPUs and other components. This increased complexity results in substantial data-access heterogeneity despite the fact that it is only a 2P design.

3. CBENCH AS A NUMA ANALYSIS TOOL

Cbench is a benchmark utility framework developed at Sandia National Laboratories and designed for HPC system integration engineers and analysts. The Cbench framework consists of a toolkit of Perl and Bash scripts that manage the building of supported open-source benchmarks from source code and the creation, submission, and analysis of benchmark jobs on Linux HPC clusters. In addition, Cbench provides basic statistical analysis to compare benchmark results for nodes in a cluster so that machines with substandard performance may be automatically identified. Running benchmark programs in Cbench provides a robust and straightforward method for executing identical tests on evaluation machines in order to compare performance.

Cbench is also used to evaluate new systems and architectures. The Cbench Single-Node-Benchmark (`SNB`) script runs node-level tests and produces a LaTeX report showing the baseline performance of a given system.

3.1 Previous Cbench Capabilities

The main focus of the Cbench framework is to run benchmark jobs across a large number of nodes in a Linux cluster to test various attributes of the system. Benchmark tests run within Cbench are generally divided into cluster-level MPI-based tests (e.g. `Linpack`, OSU Bandwidth Benchmarks, NAS Parallel Benchmarks) and node-level serial or parallel tests (e.g. `STREAM`, `memtester`, node-level `Linpack`). The NUMA data-access characterization effort presented in this paper is focused on node-level data-access bandwidth, so we focus on the node-level portion of Cbench.

3.2 Integrating NUMA Testing and Analysis

Our goal is to enable comprehensive NUMA performance analysis using Cbench. As a starting point, we extended the Cbench framework to characterize the single-threaded data-access bandwidth of NUMA systems. This characterization is achieved by running benchmark programs using the possible data-access scenarios for a given system and partition the results into *bandwidth classes*, or equivalence classes of data-access bandwidth, based on the empirical results.

We automate these data-access characterization tests by adding the capability to run benchmarks in a NUMA-aware fashion (i.e., running benchmarks with CPU scheduler and memory allocation policies explicitly set) to the `SNB` script in Cbench. This included adding the ability to detect bandwidth equivalence classes from the data produced by these benchmark tests to the Cbench `SNB` scripts. Table 1 summarizes our additions to Cbench.

Table 1: Capabilities of Cbench Versions

	OLD CBENCH	NUMA CBENCH
MPI-level	✓	✓
Node-level	✓	✓
NUMA Memory		✓
NUMA PCIe+GPU BW		✓
NUMA BW Classes		✓

3.2.1 Data-Access Bandwidth Benchmarks

We use the `STREAM` [6] [7] benchmark to determine memory-access bandwidth between CPU cores and memory nodes, interconnect links, memory controller overhead, and memory technology (e.g., DDR3). `STREAM` executes four types of memory-access operations on a large data array: `Add`, `Copy`, `Scale`, and `Triad`. `Triad` is similar to operations found in many scientific applications, so we report only `Triad` results.

We use components of the Scalable HeterOgeneous Computing (`SHOC`) benchmark suite [3] to characterize PCIe data-access bandwidth. The data-access performance of a device connected by PCIe I/O links in a system such as that shown in Figure 1 is a function of traversing the PCIe links, the HT CPU interconnect links from CPUs to PCIe devices, controllers and chipsets along the CPU-to-PCIe path, and the specific PCIe device that contains the requested data. We present the analysis of high-performance GPUs transferring data across the PCIe interconnect to requesting processors in the system shown in Figure 1 and described in Table 3. The CPU-to-I/O controller affinity shown by these tests also applies to other PCIe devices, though the reported absolute bandwidth for other devices may differ.

Analysis of data transfers for a GPU connected by PCIe involves measuring the download rate (writing to GPU memory) and the readback rate (reading from GPU memory) when transferring data between the host CPU core and the GPU device. To measure these rates, we use the `BusSpeedDownload` and `BusSpeedReadback` tests in the `SHOC` benchmark suite. These tests measure the bandwidth of the link(s) between host processor and GPU device by transferring data payloads of varying size to (download) and from (readback) the GPU device. Systems configured with NVIDIA GPUs can run both CUDA and OpenCL code, and in some cases, the PCIe bandwidth reported by the CUDA and OpenCL versions of `SHOC` vary. Cbench is configured to run both versions, though we only report CUDA results in this paper.

3.2.2 Testing Data-Access Bandwidth

Benchmarking every data-access scenario in a NUMA system entails mapping benchmark processes and their memory in every possible configuration that a program might encounter during execution. For modern many-core systems, mapping processes to individual cores greatly expands the set of core-to-node combinations that must be tested. To test all processor-core-to-memory-node combinations, $p \times m$ tests must be executed, where p is the number of CPU cores and m is the number of memory nodes in a system and where $p >> n$. However, cores that are attached to the same memory node show virtually identical memory bandwidth when accessing data on a given memory node (see Table 2), so core-to-node testing is unnecessary. We therefore allow the process scheduler to choose to schedule a process on any core attached to a memory node (i.e. any core on the same die). Such a process-to-node mapping reduces the number of tests to $m \times m$ and is done at run-time using the `numactl` tool for both `STREAM` and `SHOC` benchmarks. This ensures that the host CPU process is executed on the appropriate cores and that its memory is bound to the appropriate memory node.

For PCIe tests, it is technically only necessary to characterize one set of core/memory node combinations because GPU memory transfers use pinned memory. This means that the location of the pinned memory, not the location of the host CPU process, is the determining factor in the data-access performance for GPU programs. Nevertheless, for completeness the download and readback rates for each of the node-to-device combinations in a system are gathered by Cbench and organized into bandwidth equivalence classes, similar to the results of the `STREAM` benchmarks.

3.2.3 Addition of NUMA Tests to Cbench

We extend the `SNB` script to characterize data accesses in NUMA systems by running a given program from all cores to all memory nodes or from all memory nodes to all memory nodes and binding CPU and memory affinity as described in Section 3.2.2. This solution has the benefit of providing NUMA characterization capability to any of the node-level tests already supported by Cbench. Cbench builds many varieties of the `STREAM` benchmark; using versions from different compilers only requires instructing Cbench to build the benchmarks with the appropriate compiler.

GPU benchmark support in Cbench is in the early stages of development, and the addition of the `SHOC` PCIe bandwidth tests to the Single-Node-Benchmark script provides the first complete GPU testing capability for Cbench. These `SHOC` tests facilitate PCIe data-access characterization similar to the `STREAM` tests discussed previously.

3.2.4 Automatic Bandwidth Class Detection

A critical component of our data-access bandwidth analysis is the synthesis of *bandwidth classes* from a system's benchmark results. We employ the `Algorithm::KMeans` Perl module implementation of K-Means clustering [5] to provide automatic bandwidth classification of the benchmark results gathered by Cbench. To prepare the data for K-Means analysis, we first process the results for all iterations of each `STREAM` or `SHOC` benchmark version that was executed by Cbench and use only the best result for each data-access scenario. Pruning the data in this manner improves the accuracy of the K-Means algorithm by preventing cluster detection for results that are produced by poorly-optimized benchmark executables and not actual bandwidth classes. Once the data are pruned, the K-Means algorithm is used to determine the best clusters for the benchmark dataset.

There are two elements of the K-Means cluster algorithm that are of paramount concern: the number of clusters in the data (K) and the initial cluster center values. This algorithm automatically tries all values of K where $2 \leq K \leq \sqrt{\frac{N}{2}}$ and N is the number of data points for a given data set. The value of K with the best ratio of $\frac{avg.\ cluster\ radius}{avg.\ dist.\ btw.\ cluster\ centers}$ is returned as the number of clusters in the data. The upper bound for K was set by the module developer and has been sufficient for our purposes. The choice of the initial cluster center values for each K is the most critical aspect of proper bandwidth class detection using K-Means. The initial cluster centers are chosen using random data points, which means that the quality of the final classification is somewhat random. For a first effort at automating the process of data-access classification this randomness is acceptable, but we intend to improve the accuracy and determinism of the classification process.

3.3 Cbench NUMA Characterization Results

Cbench `SNB` NUMA Report. Tables 4 and 5 show the LaTeX report for memory and PCIe data-access bandwidth results produced by the Cbench `SNB` script for the system described in Table 3. As noted previously, the best results for each type of data access are used to determine the bandwidth classes shown in Table 4. The tables generated by Cbench are useful for comparing versions of a benchmark, as well as for checking the configuration of the machine. For example, the PCIe results in Table 5 show that both GPU devices are local to memory nodes 0 and 1 (i.e. both GPUs are local to socket 0), meaning that both are accessed through the same I/O controller and that the system may be misconfigured.

4. CONCLUSION

Next Steps. The NUMA data-access performance characterization work in this paper focused on a two-socket AMD system. Systems with much more heterogeneous architectures are available today, and analyzing these systems is one of the primary focuses of our next development effort. Furthermore, other aspects of NUMA-related data-access performance such as data-access latency, network I/O, and disk I/O will be incorporated into Cbench to provide a more comprehensive analysis of data-access performance.

The work presented in this paper focused exclusively on single-threaded analysis of data-access bandwidth. We have the initial implementation of multi-threaded memory bandwidth tests using an MPI version of `STREAM` already in Cbench. The analysis of sharing data-access resources among multiple threads is key to predictive performance modeling in NUMA systems. As we further develop our multi-threaded testing ability, we are also working to develop data-access performance models using both single-threaded and multi-threaded system characterization data from Cbench to model the performance of applications in NUMA systems by taking into account a program's data-access profile and the performance

Table 2: Core-to-Memory-Node `STREAM TRIAD` Results
Test run 30 times from each core 0-3 to each memory node

To Mem. Node	Mean BW	Std. Dev.
0	8.1 $^{GB}/_{s}$	0.020
1	5.1 $^{GB}/_{s}$	0.013
2/3	3.0 $^{GB}/_{s}$	0.023

Table 3: Configuration of the AMD 2P Test System

CPU Model	Magny-Cours 6134
CPU Cores/Mem. Nodes	16/4
Motherboard	Supermicro H8DGG
GPUs (#)	Tesla C2050 (2)
Linux Kernel	2.6.18-194.17.4.el5

Table 4: **Data-Access Bandwidth Classes as determined by Cbench for AMD 2P NUMA System**

BW Class	STREAM Triad (GB/s)	BusSpeedDownload (GB/s)	BusSpeedReadback (GB/s)
0	$3.92 < BW \leq 6.44$	$5.32 < BW \leq 5.85$	$5.33 < BW \leq 6.58$
1	$2.18 < BW \leq 3.92$	$2.35 < BW \leq 5.32$	$2.32 < BW \leq 5.33$
2	≤ 2.18	≤ 2.35 GB/s	< 2.32

Table 5: **NUMA `STREAM` and `SHOC` Bandwidth Test Results**

STREAM Triad (GB/s) – best value in each column is highlighted									
	CPU: Node 0 Mem: Node 0	CPU: Node 0 Mem: Node 1	CPU: Node 0 Mem: Node 2	CPU: Node 0 Mem: Node 3	··· ···	CPU: Node 3 Mem: Node 0	CPU: Node 3 Mem: Node 1	CPU: Node 3 Mem: Node 2	CPU: Node 3 Mem: Node 3
stream-big-c	6.40	3.91	2.16	2.15	···	2.13	2.14	3.92	6.32
stream-big-f	6.11	3.78	2.14	2.11	···	2.11	2.10	3.79	6.11
stream-c	5.32	3.35	2.18	2.15	···	2.15	2.13	3.88	5.34
stream-f	5.35	3.35	2.17	2.13	···	2.15	2.12	3.90	5.37
stream-gcc-c	6.12	3.88	2.15	2.14	···	2.14	2.15	3.89	6.28
stream-gcc2-c	6.38	3.91	2.15	2.12	···	2.12	2.12	3.91	6.14
SHOC CUDA PCIe Bandwidth Tests (GB/s) – values colored according to BW class									
Device 0									
BusSpeedDownload	5.85	5.32	2.35	2.32	···	5.81	5.32	2.35	2.30
BusSpeedReadback	6.58	5.32	2.31	2.28	···	6.58	5.28	2.31	2.28
Device 1									
BusSpeedDownload	5.81	5.32	2.35	2.32	···	5.81	5.32	2.35	2.30
BusSpeedReadback	6.58	5.30	2.31	2.29	···	6.58	5.27	2.31	2.28

penalties associated with remote data accesses in a given system. Task scheduling may also be improved by maximizing data-access performance using the data we generated by Cbench.

Summary. By adding NUMA data-access bandwidth characterization, the Cbench Single-Node-Benchmark script is now a useful tool for analyzing NUMA data-access performance. We presented this promising and extensible tool using single-threaded memory and GPU benchmarks to characterize data-access bandwidth in modern NUMA systems and we will continue to develop additional characterization tests as we look at other factors affecting data-access performance.

5. REFERENCES

[1] Cbench. `http://cbench.sourceforge.net`, Aug. 2011.

[2] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 180 –186, 2010.

[3] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pages 63–74, New York, NY, USA, 2010. ACM.

[4] A. Kleen. A NUMA API for Linux. Technical report, SUSE Labs, April 2005.

[5] S. Lloyd. Least Squares Quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, March 1982.

[6] J. D. McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http://www.cs.virginia.edu/stream/.

[7] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, pages 19–25, Dec. 1995.

[8] J. Ogden. Cbench: A Software Toolkit for Testing, Benchmarking, and Qualifying HPTC Linux Clusters. Technical report, Sandia National Laboratories, Accessed August 2011.