ARMY RESEARCH LABORATORY

Selecting a Benchmark Suite to Profile High-Performance Computing (HPC) Machines

by Jamie Infantolino, Song Park, and Dale Shires

ARL-TR-7141 **November 2014**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-7141 **November 2014**

Selecting a Benchmark Suite to Profile High-Performance Computing (HPC) Machines

Jamie Infantolino
Secure Mission Solutions

Song Park and Dale Shires
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
November 2014	Final	November 2013–February 2014

4. TITLE AND SUBTITLE
Selecting a Benchmark Suite to Profile High-Performance Computing (HPC) Machines

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jamie Infantolino, Song Park, and Dale Shires

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-S
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7141

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
New computing architectures raise questions of technical challenges and investment risks to the high-performance computer (HPC) community. Difficulties exist in determining runtime performance ahead of time and lowering the high costs associated with code migration. Furthermore, the popularity of heterogeneous systems has added complexity in assessing resource capability. A mechanism to quickly and accurately predict performance would be instrumental in assessing the risk of HPC center procurements. A sound metric for performance prediction will help to determine applicability and feasibility of new architectures. To realize the art of anticipating performance, we surveyed the literature on benchmarking suites. This report describes the central characteristics a benchmarking suite should exhibit and derives a list of pertinent requirements for diverse computational resources at the US Army Research Laboratory. Modern benchmark suites are analyzed with respect to the derived list of requirements. This study leads to a benchmark candidate that can be leveraged for a state-of-the-art and next-generation supercomputing facility.

15. SUBJECT TERMS
computation, benchmarking, high-performance computing, heterogeneous computing, parallel computing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jamie Infantolino
Unclassified	Unclassified	Unclassified	UU	32	19b. TELEPHONE NUMBER (*Include area code*) 410-278-7121

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

Current technology is moving toward computing systems containing multiple types of processing architectures. Machines now contain central processing units (CPUs), graphics processing units (GPUs), and many integrated core (MIC) architecture all within the same computer and all available for use by a single program. To achieve optimal performance, programmers need to target their applications to use the heterogeneity of available processors; this new paradigm of heterogeneous computing provides a layer of new complexity in high-performance computing (HPC). Heterogeneous parallel programming requires a different set of optimizations and programming practices that have opened up a whole new field of study on optimal programming methods for each architecture and across architectures.

With each new architecture entering the HPC community, there is a need to determine how a particular application will perform on the new processor and how the performance compares to other architectures. One solution to both problems is to use a common benchmarking program or programs and compare the results between applications and architectures. Benchmarking various architectures can be difficult because the same programs are not always supported because of programming requirements, language constraints, hardware constraints, etc. Benchmarking offers valuable information with many possible uses for research laboratories, including predicting machine performance, determining applicability of a task, and optimizing code for a given architecture. Furthermore, benchmarking can evaluate the feasibility and applicability of a new architecture just released to the market. Researchers are often unsure how available resources will perform in Army scenarios. We can use a benchmarking suite to gain insight and lower risk to technology insertion. This capability can place us at the forefront of predicting performance of both the hardware configurations and software implementations on any emerging architecture.

There are several reasons for supercomputing centers to leverage benchmarks to evaluate the available resources. Benchmarking provides a way to obtain knowledge about architectures independently and compare those architectures using a common set of metrics. The results can distinguish optimal and subpar performance on a particular resource. Knowledge of the performance on these cutting-edge technologies can serve as feedback to drive technology in the future. For example, if a benchmarking suite indicates that the current architecture is limiting in terms of bandwidth for a highly visible application, then a bandwidth demand can be expressed to the manufactures. Moreover, benchmarking can assist in determining the optimal type of computer resources to procure or the type of machine to avoid, which in turn helps facilitate an effective cost analysis.

Another salient aspect of benchmarking is the knowledge gained by using the actual benchmark software implementations on the machine. This knowledge will provide programmers with insight on the best programming practices and procedures that should be used on each architecture and what should be used to make code the most versatile across multiple architectures. Programmers will then be able to more efficiently implement algorithms, providing quicker solutions to the field. By choosing a benchmark suite that can be run on different architectures, programmers can examine the code within the suite if it is available to determine how the various implementations are done. And if there is more than one implementation available, the different implementations can be compared. For example, if there is a CUDA version available that runs very fast on a GPU, we can apply that new knowledge to an OpenCL version. In addition, if a particular benchmark program is not performing close to theoretical peak, we can examine the implementation to determine why. By using the simple benchmark programs, programmers can determine the root cause of any performance gains or issues much quicker because of the simple nature of the kernels. These simple programs help software engineers learn implementation tricks to improve their code for the various architectures.

We began our research by identifying programs to investigate as candidates. A single program cannot cover the range of disciplines and applications to serve as a predictive mechanism. This challenge spurred researchers at the University of California, Berkeley,[1] to develop the concept of a benchmarking suite. A suite consists of multiple programs, each small in size and scope, covering the most common applications found in the science and engineering fields. Numerous suites have been developed by researchers and engineers from some of the top institutions and companies around the world. We surveyed existing suites to leverage their nontrivial development time, typically measured in years. Moreover, the suites that are presented in literature have gone through peer reviews by subject matter experts within the community. These facts manifest confidence in the suite as a good measure of performance indication.

To the best of our knowledge, this report is the first survey to comprehensively examine available benchmark suites for use in heterogeneous HPC. The information obtained by this effort should provide the foundation for benchmark research in this new heterogeneous computing era. It provided a strong starting point for research on heterogeneous benchmarking methods.

In conducting an extensive study of benchmarking suites to assess effectiveness and utility, we first formulated the characteristics of a good benchmarking suite. These characteristics led to the development of a list of properties for a good benchmarking suite. In combination, a list of requirements was generated for an ideal benchmarking package. Each benchmark suite was then evaluated for advantages and disadvantages to guide with the survey.

2. Motivation

The first motivating factor to studying available benchmark suites is to determine the particular application programs that each suite has to ascertain if the results have any relevance to Army applications. If a suite covers only a small subset of applications, then the results cannot be used Army wide. In addition, if the outputs from the suite have no meaning to the HPC industry, then the results cannot be compared against other results that may have been published.

Second, the supercomputing center at the US Army Research Laboratory (ARL) hosts multiple systems consisting of both traditional x86 systems and accelerator-enhanced heterogeneous systems. A tool that quickly predicts the performance of these diverse computing systems is currently lacking. Software development of implementing performance analysis on contrasting architectures remains a challenge. A testing method is needed as increased diversity in architecture is observed in current trends.

Finally, the last motivating factor to study available benchmark suites is to capitalize on past high-quality work. By circumventing the process of designing a benchmark that satisfies our requirements from scratch, we can instead concentrate on advancing the benchmark suite. Therefore, we conducted a comprehensive survey of various benchmarking packages.

3. Essential Characteristics of a Benchmark Suite

After evaluating a broad range of benchmark suites in the literature, we report the essential characteristics. The first characteristic pertains to extreme simplicity; programs that are considered too easy do not represent any real-world applications. The results are therefore irrelevant and do not fulfill the goals of this effort.

The second characteristic, related to the first, involves complexity, where programs that are too complex lead to difficulty drawing conclusions based on the results. Without being able to identify what operations within the program are contributing to the performance numbers, accurate runtime prediction cannot be derived. For example, if an application program covers a reduction operation and a scan operation, it would be impossible to determine which part of the final runtime numbers are influenced by which operation, making it infeasible to draw conclusions for those operations.

The third characteristic incorporates applicability, where the set of programs within the suite should cover a wide application area. A benchmark suite that covers only a select subset of applications will only provide information for that subset and cannot be used to make cost-

effective decisions on architectures. A benchmark suite covering a wide range of applications represents a superior sampling to support research now and in the future.

The fourth characteristic helps to assess the hardware side of the program; the suite should not only test software implementations but also provide data on the hardware status. One can think of programming as a 2-sided coin: the hardware side and the software side. They must work together to get the best performance for any program on any architecture. Gone are the days when a programmer can only worry about improving the software implementation or improving the hardware configuration to improve performance. Today, more than ever, both parts have to work together to achieve the best performance possible. A versatile benchmark suite will profile and provide metrics for both software and hardware performance. Moreover, testing of the machine helps diagnose hardware issues over time by determining and maintaining a good baseline metric against which to compare.

The fifth characteristic provides results in industry-accepted standards. The standard does not have to be defined but should be a unit of measure commonly used by researchers when reporting data. Results are used to draw conclusions, analyzed as a comparison tool, and evaluated to predict performance, all of which need a solid unit of measure. If the results adhere to the industry standard, then publications based on this benchmark suite will be aligned with the industry. For example, in regard to machine performance, a lot of reported metrics are floating-point operations per second (FLOPS) or GFLOPS (10 billion FLOPS), which provides for a common unit to compare data against. ARL researchers can then compare their data against published data and help outside reviewers examine ARL-generated data. In addition, the benchmark suite needs to provide relevant information about the hardware system itself, e.g., throughput numbers and bandwidth numbers. These values assist in locating hardware bottlenecks, testing different hardware configurations, diagnosing system problems, or signaling a decline in performance as the hardware ages.

The sixth characteristic—one of the most complex and usually not considered until after development has been completed—relates to supporting multiple architectures. Having a suite of programs running on different architectures, such as GPUs, MICs, and CPUs, adds complexity and technical challenges. There are many well-written benchmarking suites that do not run on different architectures because they can only run on a subset of architectures. As heterogeneous computing systems increase, the execution of a variety of architectures becomes an important factor. Capability evaluations are needed for each architecture to assist researchers in deciding on a target architecture for their application. This characteristic is the hardest one to predict and fulfill. However, history is a good predictor of the future. If a computer language that a suite is implemented in had the ability to run on new architectures in the past, chances are much higher it will be able to run on each new future architecture. So if the benchmark suite has been able to be run on previous generations of hardware, support most likely will continue for future generations with very little effort needed by researchers.

The last characteristic is related to the sixth characteristic—one implementation of the application programs that can be utilized across multiple architectures which will help with consistency across each architecture. A single implementation for different architectures allows for logical and direct comparisons, minimizing the effort needed to maintain the benchmark suite. If one set of application programs can be used across all available platforms, then only one code base will need to be maintained.

4. Requirements for Benchmark Suite Assessment

By considering the motivations for studying benchmark suites and identifying the characteristics and properties of a good benchmark suite, we derived a list of requirements. A clear, concise list of requirements is important when evaluating each benchmark suite. This list provides a common evaluation criteria in which decisions can be made. The benchmark suite must possess the following requirements:

1. Be open source.
2. Utilize heterogeneous computing.
3. Be widely accepted in the HPC community as a trusted benchmarking suite.
4. Generate publishable results.
5. Be expanded if needed.
6. Be modifiable.
7. Possess one software implementation per application program.
8. Cover a wide variety of numerical applications.
9. Utilize multiple nodes on a cluster.
10. Be designed for HPC.
11. Have the ability to test the hardware performance of the architecture.

The first requirement of the chosen benchmark suite, being open source, is needed in case ARL wants to expand or modify the code in the future. This also usually avoids any licensing fees or restrictions that may be needed for non-open-source programs. Even though a program is open source, it may not be expandable or modifiable; this limitation leads to 2 other requirements, numbers 5 and 6. These requirements are discussed in the following paragraphs.

The second requirement seems obvious, but it needs to be stated: the suite chosen has to support heterogeneous computing. The suite must run on a variety of architectures, particularly the ones available to ARL researchers.

One of the goals of the benchmarking effort is to expand community knowledge; hence, results must be publishable. Publishing is easier when the benchmarking suite is accepted within literature, as stated in requirement 3. Specifically, the suite has already been through the peer review process, thus creating trust in the suite. This process provides confidence that the results will be accurate and usable in determining the performance of the architectures across all applications, thus fulfilling one of the main goals of using the benchmarking suite. In addition, the benchmarking suite must be able to be used in publications, as stated in requirement 4.

Requirements 5 and 6 can be related. For usability and maintainability, it is necessary for the chosen benchmarking suite to be able to be expanded and modified as needed now and in the future. It will allow us the ability to customize the suite to meet its needs along with providing opportunities to experiment with various architectures. Additions or modifications may include, but are not limited to, expanding to run on any future architectures, fixing any bugs, improving the implementation if necessary, etc. These requirements are not obvious with open source software. To fully justify the time that is spent on the benchmarking effort, the suite must be able to be used for years to come on any architecture. The ability to modify and expand any benchmarking suite will ensure that goal is achievable.

As part of the maintainability efforts, it will be very beneficial to have to maintain only a single source code. This leads to requirement 7: one software implementation per application. The effort required to maintain a benchmark suite increases exponentially for each software implementation of the same application that is needed.

As stated previously, one of the main characteristics and properties of a benchmarking suite is the breadth of applications it covers. Too many application programs lead to unnecessarily long runtimes, but enough must be included to provide in-depth knowledge.

At the ARL DOD Supercomputing Resource Center (DSRC), there are numerous clusters available for researchers to improve runtime performance of their applications. Many researchers have written their codes to run on multiple nodes. A good benchmarking suite needs to have the ability to run on a multinode cluster as stated in requirement 9. This will provide data for researchers to profile different architectures in a cluster configuration.

In regard to requirement 10, the benchmarking suite is going to be supporting the HPC computers at ARL, so any benchmarking suite chosen must be designed with large-scale parallelism in mind. The future potential of HPC is of high interest behind this research.

As previously stated, the hardware configuration of any of these architectures is just as important as the software implementation when it comes to improving runtime performance of an application. Declines in hardware performance will negatively impact performance of any

system. A benchmarking suite can be used to catch these performance declines or help test different hardware configurations to see the impact each may have on performance. The suite will provide a good baseline metric to compare against as well over the lifetime of the architecture. Therefore, requirement 11 specifies a hardware test component to the suite.

5. Available Benchmarking Suites

With a reasonable list of requirements, the next step consisted of surveying the available benchmark suites. This section presents the details of the available benchmark suites that were examined during this research. Each subsection will have a list of advantages and disadvantages along with a general overview of the benchmarking suite. If there are no advantages listed, it usually invalidated the suite from further evaluation. At the end of each subsection, the requirements met by the benchmarking suite are listed. The numbering scheme used in that subsection matches the number from the previous requirements section. For further information on the benchmarks listed, please refer to the References section. A summary table is provided at the end of this section to compare the requirements met by each benchmark suite.

5.1 OpenDwarfs[2]

Background:

- Developed by Virginia Tech University.
- Dwarfs are high-level applications that cover a variety of domains.
- Each application does not cover the whole breadth of the dwarf, so long-term plans include adding applications to each dwarf so users can pick which one covers their specific application.
- Developers of the suite made efforts to avoid optimizing for any specific architecture.
- Thirteen dwarf programs.

Advantages:

- Runs on ARL machines.
- Claims it is not optimized for a specific architecture, which could lead to better results on our heterogeneous machines.
- One of the industry-approved suites.

Disadvantages:

- Only 13 dwarf programs.

- Limited documentation to help usability.
- Needs specific versions of third-party libraries.
- Does not test the hardware component of the system.

Requirements Met: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

5.2 SHOC[3]

Background:

- Developed by Oak Ridge National Laboratory and the University of Tennessee.
- Multiple levels of programs to classify performance.
- Implemented in OpenCL.
- Uses Message Passing Interface (MPI) to scale from one node to a cluster.
- Two kinds of testing available:
    - Stress testing: identify devices with bad memory, insufficient cooling, or other hardware issues
    - Performance testing: measure system performance
- Broken into 3 levels:
    - Level 0: Stress tests to measure low-level hardware characteristics
        a. Both a stress test and a performance test.
    - Level 1: Represent common parallel programming tasks in real applications
        a. Performance testing.
    - Level 2: Real-world applications
        a. Performance testing.
        b. Provides stability testing using a set of “dwarfs” within the code to test a machine over its lifetime or a new machine.
- Runs one kernel for a very long time to increase the chance of overheating or other transient effects.
- Checks results occasionally to ensure calculations are being performed correctly.
- Performs using data already on device, which leads to more accurate results since the device will not have the time to cool while data is being transferred if performed using data on the CPU.

Advantages:

- Runs on ARL machines.

- Claims it is not optimized for a specific architecture.
- Extensive documentation.
- Sixteen application programs available, broken down into 3 different levels.
- Multiple types of testing available, e.g., performance, stress, stability testing.
- One of the industry-accepted suites.

Disadvantages:

- Timing mechanisms use event timers that supposedly underestimate the time.
- Kernels appear to be optimized for GPUs.

Requirements Met: All

5.3 Rodinia[4]

Background:

- Implemented with OpenMP for CPU and CUDA/OpenCL for GPU/MIC.
- Nineteen programs.

Advantages:

- OpenCL implementation available.
- Used within literature.

Disadvantages:

- Not all programs are available to be run over all architectures.
- Different code base for each implementation (CUDA vs. OpenCL vs. OpenMP).
- Code optimized for each coding language (e.g., OpenCL will run really well on GPU but not so well on CPU).

Requirements Met: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

5.4 CompuBench[5]

Background:

- Open standards–based benchmarking suite.
- Downloadable executable.

Advantages: None

Disadvantages: Not open source

Requirements Met: 2, 3, 10

5.5 Parboil[6]

Background:

- Suite of 11 programs.
- Intended to study the throughput of various architectures.
- Intended to help programmers get the most out of the architectures available to them.
- Optimized versions for each implementation (OpenCL vs. CUDA vs. C).
- Optimized versions of OpenCL program available (Base vs. NVIDIA).

Advantages: Different versions available to test different compilers.

Disadvantages:

- Need multiple versions to optimally run on the variety of architectures.
- Small number of dwarfs available.
- Does not appear to cover the breadth of applications available in SHOC or OpenDwarfs.

Requirements Met: 1, 2, 3, 4, 5, 6, 7, 8, 10, 11

5.6 HPC Challenge[7]

Background:

- Challenge program issued to various computing centers to determine the best computers for individual applications.
- Consists of 7 basic tests:
  - HPL (Linpack)
  - DGEMM: Double precision matrix-matrix multiplication application whose result is the floating point rate of execution.
  - STREAM: Synthetic application program to measure memory bandwidth whose result is computation rate for the kernel.
  - PTRANS: Application program measures the communication between pairs of processors communicating with each other simultaneously.
  - RandomAccess: Application program performs random updates of memory and measures the rate.
  - FFT: Double-precision complex 1-dimensional discrete Fourier transform to measure floating point rate of execution.

- Communication bandwidth and latency: Application program is a set of tests to measure latency and bandwidth of simultaneous communication patterns.

Advantages: A suite of the most common benchmarking programs.

Disadvantages:

- Covered by other suites.
- Just a challenge; not really a new suite in terms of new applications.
- Does not appear to have active user base.
- Does not run on anything but CPU out of the box.

Requirements Met: 1, 3, 4, 5, 6, 9, 10, 11

5.7 Graph500[8]

Background:

- Created for high-memory-use applications.
- Not really created for HPC (GPU/MIC) architectures.
- Graph algorithms.
- Addresses:
  - Concurrent search
  - Optimization (single-source shortest path)
  - Edge-oriented applications
- Goal: develop a compact application that has multiple analysis techniques accessing a single data structure representing a weighted, undirected graph
- Input data sizes: 17 GB–1.1 PB
- Steps to benchmark algorithms:
  1. Generate edge list.
  2. Construct a graph from edge list (timed portion).
  3. Randomly sample 64 unique keys.
  4. For each key, compute parent array (timed portion) and ensure that the parent array is a correct Breadth-First Search key for given search tree.
  5. Compute and output performance information.

Advantages: Tests non-GPU computers.

Disadvantages: Not designed to run on all the HPC machines available at the ARL DSRC.

Requirements Met: 1, 3, 4, 5, 6, 7, 8, 9, 10

5.8 Phoronix[9]

Background:

- Suite consisting of 60+ test programs.
- A variety of published results to compare results against.
- Utilizes XML testing framework.
- Can easily be expanded if needed.

Advantages:

- Over 60 programs that cover a wide range of applications.
- Precompiled.
- Can be expanded.

Disadvantages:

- Too many programs to be used efficiently as a benchmark suite.
- Too generic for HPC.
- Requires PHP software to run.

Requirements Met: 1, 2, 3, 4, 5, 6, 7, 8, 9, 11

5.9 Linpack[10]

Background:

- Solves large, dense systems of linear equations using Gaussian elimination with partial pivoting.
- Dominated by matrix-matrix multiplications.
- Data access is unit stride and hidden by concurrently performing calculations on previously retrieved data.
- Performs better on machines with high-floating-point computation rates and adequate streaming memory systems.
- Calculations represent common operations in real-world applications, but Linpack does not cover a wide variety of applications.

Advantages:

- Very well-known and well-respected benchmark program.
- Been in use for years.

Disadvantages: Only covers one type of application.

Requirements Met: 1, 2, 3, 6, 8, 9, 10

5.10 High-Performance Conjugate Gradient (HPCG)[11]

Background:

- An addition to Linpack to run as a benchmark on supercomputers.
- C++ implementation of a preconditioned conjugate gradient method with a local symmetric Gauss-Seidel preconditioner.
- Designed to represent more realistic applications, as opposed to Linpack.
- Better correlated computation algorithms and data access patterns seen today in the HPC community.
- Requirements:
  - Accurately predict system for target suite of applications.
  - Drive improvements to computer systems, which will benefit real-world applications.
- Goal: Provide a benchmark that is more reflective of real-world applications than the current LINPACK.

Advantages:

- Better representative of real-world applications computations.
- Developed by the same people that developed Linpack.

Disadvantages:

- Covers only one type of application.
- Brand new, so it has not been fully reviewed in literature.

Requirements Met: 1, 2, 3, 6, 8, 9, 10

5.11 NAS Parallel Benchmarks (NPBs)[12]

Background:

- Developed by the National Aeronautics and Space Administration (NASA) to evaluate supercomputer performance.

- Derived from computational fluid dynamics.
- Consists of 3 groups of application programs.
- The original 5 kernels:
    - IS: Integer Sort, random memory access
    - EP: Embarrassingly Parallel
    - CG: Conjugate Gradient, irregular memory access and communication
    - MG: Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
    - FT: Discrete 3-dimensional fast Fourier transform, all-to-all communication
- Three pseudo applications:
    - BT: Block Tri-diagonal solver
    - SP: Scalar Penta-diagonal solver
    - LU: Lower-Upper Gauss-Seidel solver
- Multizone version:
    - BT-MZ: uneven-sized zones within a problem class, increased number of zones as problem class grows
    - SP-MZ: even-sized zones within a problem class, increased number of zones as problem class grows
    - LU-MZ: even-sized zones within a problem class, a fixed number of zones for all problem classes
        a. Benchmarks for unstructured computation, parallel input/output (I/O), and data movement
    - UA: Unstructured Adaptive mesh, dynamic and irregular memory access
    - BT-IO: test of different parallel I/O techniques
    - DC: Data Cube
    - DT: Data Traffic
- GridNPB: rate the performance of computational grids.
    - ED: Embarrassingly Distributed
    - HC: Helical Chain
    - VP: Visualization Pipeline
    - MB: Mixed Bag
- OpenMP, Java, MPI implementations available.

Advantages: From NASA.

Disadvantages: Different implementation for each architecture.

Requirements Met: 1, 3, 6, 8, 9, 10

5.12 Iterative Solver Benchmark[13]

Background:

- Solves an iterative solver for sparse matrices.
- Does not address scalable distributed memory parallelism or nested parallelism.
- Lesser-known benchmark.

Advantages:

- Developed by the author of Linpack.
- Addresses a different need than Linpack.

Disadvantages:

- Different implementation for each architecture.
- Not optimized for any HPC.
- Implements only one type of application.

Requirements Met: 1, 3, 6, 9, 10

5.13 EEMBC[14]

Background:

- General-purpose CPU benchmarking suites.
- Programs are a snapshot of scientific and engineering applications.
- Does not run on GPU.

Advantages:

- General-purpose suite.
- Been in use for many years.

Disadvantages:

- Cannot be run on the GPU.

Requirements Met: 1, 3, 4, 6, 7, 8, 9

5.14 SPEC[15]

Background:

- General-purpose benchmarking suites.
- Multiple target architectures available.
- Created to provide a realistic standard benchmarking suite.
- OpenMP and MPI versions available.
- Licensing fee is $800 and up.

Advantages:

- General-purpose suite.
- Been in use for many years (however, does not seem to have been updated recently).

Disadvantages:

- Need a different suite for each type of architecture.
- No GPU available.
- No OpenCL version.
- Have to purchase the software.
- Not open source.

Requirements Met: 1, 3, 4, 5, 6, 7, 8, 9

5.15 SPLASH-2[16]

Background:

- Early parallel application suite.
- Multithreaded applications from scientific and graphics domain.
- Does not run on GPU.

Advantages: None

Disadvantages:

- Cannot be run on the GPU.
- Not state of the art.
- Data sets are too small.

Requirements Met: 1, 3, 4, 5, 6, 7, 8, 9

5.16 Parsec[17]

Background:

- A wider range of parallelized techniques when compared to SPEC, EEMBC, and SPLASH-2.
- Optimized for multicore CPU.
- Intended to be used on chip-multiprocessors, not HPCs.
- Does not run on a GPU.

Advantages:

- General-purpose suite.
- Spans a wide range of applications.

Disadvantages:

- Cannot be run on a GPU.

Requirements Met: 1, 3, 4, 5, 6, 7, 8, 9

5.17 MineBench,[18] MediaBench,[19] ALP-Bench,[20] BioParallel[21]

Background: Application-specific benchmark suites.

Advantages: Optimized for one application.

Disadvantages:

- Cannot be run on the GPU or other accelerators.
- Covers only one application.

Requirements Met: 1, 3, 6, 8, 9

5.18 Summary Table

The following Table summarizes the requirements met by each benchmark suite.

Table Summary of available benchmark suites

Suite	Requirement										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
OpenDwarfs	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
SHOC	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Rodinia	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
CompuBench	No	Yes	Yes	No	No	No	No	No	No	Yes	No
Parboil	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No
HPC Challenge	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes
Graph500	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Phoronix	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Linpack	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No
HPCG	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	No
NPB	Yes	No	No	No	No	Yes	No	Yes	Yes	Yes	No
Iterative Solver Benchmark	Yes	No	Yes	No	No	Yes	No	No	Yes	Yes	No
EEMC	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No
SPEC	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
SPLASH-2	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Parsec	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
MineBench, MediaBench, ALP-Bench, BioParallel	Yes	No	Yes	No	No	Yes	No	Yes	Yes	No	No

6. Benchmark Suite Analysis

The summary table shows that only one benchmark suite met all of the requirements: SHOC. SHOC covers both the stated and implied requirements that have been previously discussed. It is open source, easily downloadable, and can be run on all architectures at ARL with no modifications to the application programs. This includes the heterogeneous computing resources. It was developed by the well-respected Oak Ridge National Laboratory and has been in the literature for almost 5 years. It has been thoroughly reviewed by various well-known researchers in the industry and has been widely accepted as a good metric for benchmarking various architectures. It does have the ability to be expanded and modified. The results of using this benchmark suite are publishable, and the code is provided to researchers “as is”, so modifications are allowed. It has one implementation that can be used across diverse architectures. The breadth of applications it covers is extensive. It does have the ability to go from a single node to a multinode cluster without many modifications. The code has been written and optimized for HPC machines. Finally, it has the ability to provide hardware performance numbers as well.

There are some other benefits to using SHOC. It is user friendly and can compile easily on a number of different machines. It is easy to run and interrupt the results. The code itself is easy to read and understand, which will help if modifications are needed in the future. The applications themselves are well-known algorithms that do not take very long to understand. This makes reading and understanding the code much easier. The benefit to understanding the code is that it helps researchers to better understand the performance results. More details on SHOC are provided in the next section.

7. SHOC Details

SHOC provides a good mix of applications that are relevant for HPC centers. This benchmark suite can be used by researchers and system administrators over the lifespan of any given architecture. SHOC has OpenCL implementations of each application program. By using OpenCL,[22] the programs can be run on a variety of architectures. OpenCL is one of the few computer programming languages that can be compiled and run on GPUs, MICs, CPUs, and advanced RISC machines (ARMs), and has MPI support, which covers the range of architectures at ARL. OpenCL allows the functionality of maintaining one code base across a wide range of supported architectures. SHOC does provide a separate CUDA implementation in addition to OpenCL.

SHOC is a library of kernels split into 3 different application levels. The lowest level (level 0) contains the stress tests for the system. These tests include measuring bus speed both for download and readback, device memory tests, timing of compiling kernels, maximum flop calculations, and queue delays. These application programs can be used throughout the lifetime of the hardware. They can be used to help diagnose hardware problems and test new configurations to compare to old configurations. Performance numbers for each kernel are reported to the user.

Level 1 consists of application programs that represent common parallel programming programs found in real-world applications. These are mainly used for testing a particular architecture. Some examples of applications in this level are a sort algorithm, a reduction algorithm, and a scan algorithm. Each program is a small kernel that runs in a matter of microseconds. The kernel is run a set number of iterations to get a total runtime and average runtime for that application. This setting is designed to overcome the possible stability issues faced by small kernels to ensure the most accurate average runtime possible. Then runtime performance numbers are calculated and reported to the user.

Level 2 is another level of performance testing applications, but these applications are more complex and closer to real-world problems than level 1. The same process is used here as in level 1—the kernel is executed numerous times, and performance numbers are reported to the user.

The performance numbers reported by each application program are the numbers used for benchmarking. Each is reported independently in a summary output list. The unit of measure for each program conforms to what appears to be the industry standard for each particular test, which makes comparing results from this suite to others much easier.

8. Conclusions

SHOC provides the ability to benchmark multiple architectures while using only one code base. The results are easily understood and can be traced back to a specific application. All of this combined will allow researchers to use the results to predict performance of a given application on any one of the available architectures.

The results of this research effort have proved that SHOC is a benchmarking suite to be leveraged. Each suite studied was compared against a set of requirements that were based on the properties and characteristics of a good benchmarking suite.

9. Future Work

The next step is to use the benchmark suite to profile the various architectures available at ARL and obtain a baseline number for the architectures. These steps should be taken to ensure the code will run properly on the machines and produce relevant metrics to judge performance. Then the code itself should be examined to determine if the optimal kernels are being run for each architecture. This will allow programmers to conduct direct comparisons across architectures and draw conclusions. Based on this work, SHOC may need to be tailored for heterogeneous benchmarking method research. Changes could include improving the actual kernels to try to optimize the kernel parameters across different architectures.

The benchmark suite programs can also be used outside a benchmark effort because of the in-depth understanding of the various programs gained during this effort. Researchers can use that knowledge to take the next steps in terms of research. The suite could be used as a starting point for a project to investigate how to optimize programs across each architecture. For example, a certain program may run better on certain architectures over others because of input parameters. There are automatic tuning methodologies that could be developed to improve program performance across each architecture while still maintaining one code base. The knowledge gained here will continue to help researchers improve their algorithms and implementations for the various architectures.

10. References

1. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW. The landscape of parallel computing research: a view from Berkeley. EECS Department, University of California: Berkeley (CA); 2006. Report No.: UCB/EECS-2006-183.

2. Feng W, Lin H, ScoglandT, Zhang J. OpenCL and the 13 dwarfs: a work in progress. In: ICPE 12 Conference Committee. ICPE 12. Proceedings of the Third Joint WOSP/SIPEW International Conference on Performance Engineering; 2012 Apr 22–25; Boston, MA. Washington (D): Association for Computing Machinery; c2012. p. 291–294.

3. Danalis A, Marin G, McCurdy C, Meredith JS, Roth PC, Spafford K, Tipparaju V, Vetter JS. The scalable heterogeneous computing (SHOC) benchmark suite. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units; 2010 Mar 14; Pittsburgh, PA. New York (NY): ACM; c2010; p. 63–74.

4. Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW, Lee SH, Skadron, K. Rodinia. A benchmark suite for heterogeneous computing. In: Proceedings of the 2009 IEEE International Symposium on Workload Characterization; 2009 Oct 4–6; Austin, TX. Los Alamitos (CA): IEEE Computer Society; c2009; p. 44–54.

5. CompuBench - performance benchmark for various compute APIs (OpenCL, RenderScript). [accessed 2014 Jun 2]. http://compubench.com/result.jsp.

6. Stratton JA, Rodrigues C, Sung IJ, Obeid N, Chang LW, Anssari N, Liu GD, Hwu W. Parboil: a revised benchmark suite for scientific and commercial throughput computing. Champaign (IL): University of Illinois at Urbana-Champaign; 2012. Report No.: IMPACT-12-01.7.

7. Luszczek P, Bailey D, Dongarra JJ, Kepner J, Lucas R, Rabenseifner R, Takahashi D. The HPC challenge (HPCC) benchmark suite. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06); 2006 Nov 11–17; Tampa, FL. New York (NY): ACM; Article 213.

8. Murphy RC, Wheeler KB, Barrett BW, Ang JA. Introducing the graph 500. Cray User's Group (CUG); 2010.

9. Phoronix Test Suite. Linux testing & benchmarking platform, automated testing, open-source benchmarking. c2008 [accessed 2014 Jun 2]. http://www.phoronix-test-suite.com/?k=home.

10. Dongarra JJ, Luszczeky P, Petitetz A. The LINPACK benchmark: past, present, and future. Concurrency Computat.: Pract. Exper. 2003;15(9):803–820.

11. Dongarra J, Heroux MA. Toward a new metric for ranking high performance computing systems. Albuquerque (NM): Sandia National Laboratories; 2013 Jun. Report No.: SAND2013-4744.

12. Bailey DH. NAS parallel benchmarks. In: Padua D, editor. Encyclopedia of Parallel Computing. New York (NY): Springer; 2011.

13. Dongarra J, Eijkhout V, van der Vorst H. An iterative solver benchmark. Sci Program. 2001;9(4):223–231.

14. The Embedded Microprocessor Benchmark Consortium. El Dorado Hills (CA): EEMBC; c2014 [accessed 2014 Jun 2]. http://www.eembc.org/.

15. Standard Performance Evaluation Corporation. Gainesville (VA): SPEC; c1995 [accessed 2014 Jun 2]. http://www.spec.org/.

16. Woo SC, Ohara M, Torrie E, Singh JP, Gupta A. The SPLASH-2 programs: characterization and methodological considerations. ACM SIGARCH Computer Architecture News. 1995; 23:24–36.

17. Bienia C. Benchmarking modern multiprocessors [dissertation]. [Princeton (NJ)]: Princeton University; 2011.

18. Narayanan R, Ozisikyilmaz B, Zambreno J, Memik G, Choudhary A. Minebench: a benchmark suite for data mining workloads. In: Workload Characterization; 2006 IEEE International Symposium; 2006; San Jose, CA; p. 182–188.

19. Lee C, Potkonjak M, Mangione-Smith WH. MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems. In Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture; 1997, p. 330–335.

20. Li ML, Sasanka R, Adve SV, Chen YK, Debes E. The ALPBench benchmark suite for complex multimedia applications. In Workload Characterization Symposium; 2005. Proceedings of the IEEE International; IEEE Computer Society, Washington, DC, 2005; pp. 34–45.

21. Jaleel A, Mattina M, Jacob B. Last level cache (LLC) performance of data mining workloads on a CMP-a case study of parallel bioinformatics workloads. In High-Performance Computer Architecture; 2006. The Twelfth International Symposium on, 2006; pp. 88–98.

22. OpenCL: the open standard for parallel programming of heterogeneous systems. Beaverton (OR): Khronos Group; c2014 [accessed 2014 Jul 29]. https://www.khronos.org/opencl/.

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIO LL
IMAL HRA MAIL & RECORDS MGMT

1 GOVT PRINTG OFC
(PDF) A MALHOTRA

11 DIR USARL
(PDF) RDRL CIH S
J INFANTOLINO
S PARK
D SHIRES
J ROSS
S ALLEN
D RICHIE
M BUSSE
B RAPP
R HANEY
C SLAUGHTER
RDRL CIO AM
R NAMBURU

INTENTIONALLY LEFT BLANK.