Benchmarking Application I/O in the Community

Julian M. Kunkel[1], Olga Mordvinova[2], Dennis Runz[2], Michael Kuhn[3], Thomas Ludwig[1,3]
kunkel@dkrz.de

Universität Hamburg

DKRZ DEUTSCHES KLIMARECHENZENTRUM

1: Deutsches Klimarechenzentrum
2: Ruprecht-Karls-Universität Heidelberg, Department of Informatics
3: University of Hamburg, Department of Informatics

Abstract

Benchmarking I/O performance of a high performance computer is a tough task. Many sequential and parallel benchmarks exist, however they differ in pattern, tested interfaces and internal behavior. Interpretation of results depends on one hand on the benchmark, on the other hand, projection of obtained results to particular applications is complicated or hardly possible. Porting scientific applications to an architecture is challenging, library and architecture dependencies are a burden to developers. Therefore, a few application benchmarks exist, which try to mimic application behavior in a small core. However, writing an individual application kernel as a new benchmark for each application leads to a wide diversity in benchmarks.

We propose the benchmarking tool *Parabench* which allows to mimic a rich variety of application programs. *Parabench* interprets an easy programming language during runtime to avoid portability or licensing problems. In addition, a future Trace-Replay mechanism will allow to replay application traces directly on other systems by interpretation of the traces. Standardized tests of these results ease evaluation. We will start an open community and collaborations to exchange patterns and measured results on the website. Vendors can take the patterns and provide results obtained on new architectures and systems.

Parabench

- C code with MPI support, GNU General Public License
- I/O kernels are written in *Parabench* Programming Language (PPL)
- *Parabench* parses the kernel description and prepares execution
- During runtime the program is interpreted and executed (see figure 1)

- POSIX and MPI-I/O support
- Explicit C-like file-handles and implicit (specify filenames to all calls)
- Synchronization among processes via barriers
- MPI file views (currently MPI array is supported)
- Control-structures, loops and group based execution
- Supported features: timing of commands, unified parameters, automatic return value error checking, evaluation of results

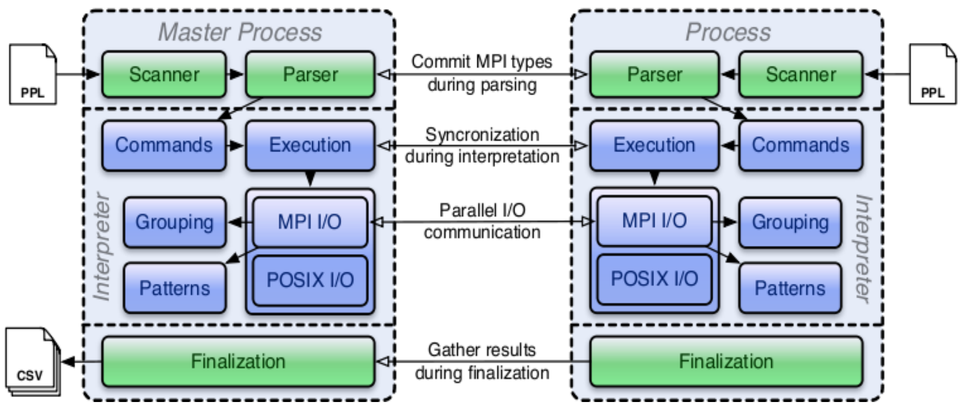

Figure 1: Parabench processing.

Advantages

- One standard I/O benchmark for most use-cases
- Library dependencies – porting library dependencies might be tough
- Avoid licensing issues – e.g. benchmark proprietary software in third-party environments. Experiments with SAP BWA application proofs the concept.
- Small interpreter – easier to understand internal behavior
- Automatic error checking in the benchmark
- Community to exchange patterns and compare results

Example Parabench Programs

MPI-I/O

```
// define the file view (right now an array is used)
// level 0 == independent, contiguous I/O
define pattern {"pattern0", 10, (100 * 1024 * 1024), 0};
// level 3 == collective, non-contiguous I/O
define pattern {"pattern3", 10, (100 * 1024 * 1024), 3};

time["MPI-IO test"] {
   barrier("world");
   time["w-lvl0"] pwrite("$env/file1-level0.dat", "pattern0");

   barrier("world");
   time["w-lvl3"] pwrite("$env/file1-level3.dat", "pattern3");

   barrier("world");
   time["r-lvl0"] pread("$env/file1-level0.dat",pattern0","world");

   barrier("world");
   time["r-lvl0"] pread("$env/file1-level3.dat","pattern3",world");
}
```

POSIX

```
define param "num" $num 1000

$env = "./env-posixio-test";
$myDir = "$env/$$rank/";
$file = "$myDir/file";

mkdir($env);

barrier;

print „filop ++“

time["POSIX stresstest"]{
        time["mkdir"] repeat $i $num mkdir("$myDir-$i");
        barrier;
        …
        time["write"] repeat $i $num write("$file-$i", 1024);
        …
}
```

Application Trace-Replay

An alternative to program I/O kernels directly is to trace application behavior i.e. communication, qualitative CPU behavior and I/O calls. Calls of the application are recorded in the trace files and can be executed by a replay program in the same order. This enables to store application patterns and use them as a gold standard to benchmark and optimize the environment under reproducible conditions. Also, licensing and portability issues are unproblematic by sharing these trace files. Without time-consuming porting of an application to a new system the application performance can be evaluated qualitatively and potential of the new system can be estimated. By versioning these trace files it is possible to evaluate performance of an older code on the same system but different environments (e.g. system kernel, I/O subsystem, configuration). Tracing and replaying behavior of non-deterministic applications (e.g. load balanced applications) is limited, but replaying results in a deterministic way is an advantage to understand and optimize the environment without dealing with non-deterministic applications.

Open Community (under development)

- Share patterns and traces in the community.
- Exchange and analyze results on the website.
- Compare results to other environments or sites.
- Determine the best suited site for an application – before porting the application to the environment!
- GUI with a similar interface for local analysis.
- Obtain common application behavior to evaluate middleware optimizations.

- We are looking for partners!
- If you are interested in this project let us know.

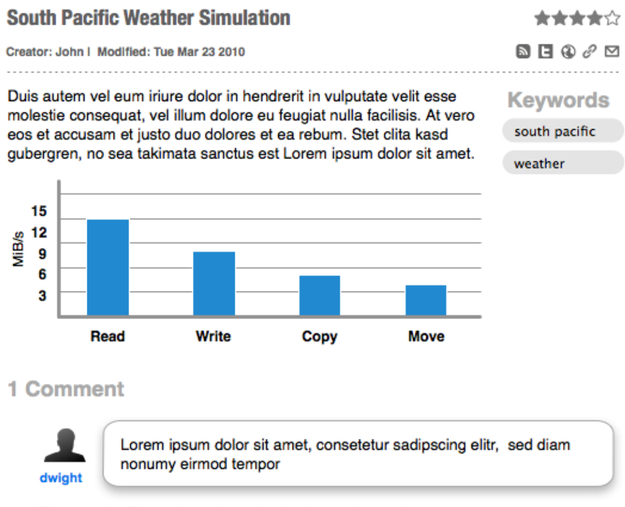

Figure 2: Results of a single benchmark run on one system.

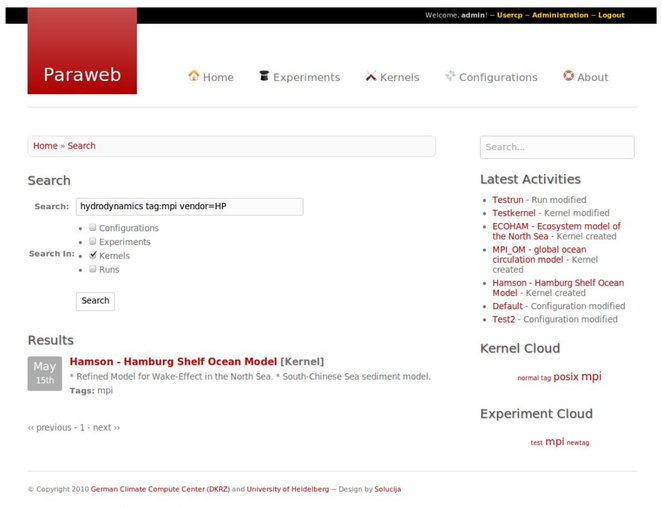

Figure 2: Searching for an application kernel.

Summary & Conclusion

Starting with *Parabench,* a tool is available to mimic application behavior by reducing the burden to programmers to recreate a full-featured application kernel. Application benchmarks allow to evaluate system behavior without porting the application directly. Our future plans are to create a trace-replay tool and to establish an open community to exchange application patterns and observed behavior. Estimating application performance before porting the application shows potential environments for the application. Vendors, middleware and operating system developers can use these benchmarks and traces to understand and to optimize the environments towards the needs of the users.

We welcome your interest to reach the goal of providing a rich set of common scientific application benchmarks.

Please see also the BenchIT project – http://www.benchit.org/