Deskilling HPL

Using an Evolutionary Algorithm to Automate Cluster Benchmarking

Dominic Dunlop, Sébastien Varrette and Pascal Bouvry

CSC research unit, University of Luxembourg, Luxembourg
{Firstname.Lastname}@uni.lu

Abstract. The High-Performance Linpack (HPL) benchmark is the accepted standard for measuring the capacity of the world's most powerful computers, which are ranked twice yearly in the Top 500 List. Since just a small deficit in performance can cost a computer several places, it is important to tune the benchmark to obtain the best possible result. However, the adjustment of HPL's seventeen configuration parameters to obtain maximum performance is a time-consuming task that must be performed by hand. In a previous paper, we provided a preliminary study that proposed the tuning of HPL parameters by means of an Evolutionary Algorithm. The approach was validated on a small cluster hosted at the University of Luxembourg. In this article, we extend this initial work by describing Acbea, a fully-automatic benchmark tuning tool that performs both the configuration and installation of HPL followed by an automatic search for optimized parameters that will lead to the best benchmark results. Experiments have been conducted to validate this tool on several clusters, exploiting in particular the Grid'5000 infrastructure.

1 Introduction

Statistics on high-performance computers are of major interest to manufacturers, users, and potential users. The Top500 project [4] operates at a worldwide level as a reference contest to evaluate the 500 most powerful computer systems. The list is updated twice a year and the computers are ranked by their performance on the long-established High-Performance LINPACK (HPL) [28] benchmark, despite the existence of newer alternative benchmarks [9]. HPL is a software package that solves a (random) dense linear system using double-precision (64 bit) floating-point arithmetic on distributed-memory computers. Seventeen configuration parameters should be tuned and adapted to the computing platform to obtain maximum performance. Even though some guidelines exist to guide the search of the parameter space (firstly from the authors of HPL themselves, and secondly in articles that discuss HPL tuning such as [7,30]), this is generally a tedious task that is performed by hand. In a previous paper [8], we showed how an evolutionary algorithm (EA) may be exploited to determine the best possible parameters in a nearly automatic way, in order to maximize the results of the benchmark. The approach was validated on a small cluster hosted at the University of Luxembourg.

In this article, we describe the extension of this approach into a framework called Acbea (*Automatic Cluster Benchmark with Evolutionary Algorithm*), which provides a fully-automatic benchmark tuning tool based on an EA that explores the parameter space with many small benchmark runs, delivering parameter combinations that are likely to produce outstanding results in larger runs. The approach may be used iteratively if necessary, progressively reducing the proportion of the parameter space explored. This paper is organized as follows: §2 defines the problem

statement and recalls the approach proposed in our initial work. §3 describes the various software elements that comprise the ACBEA framework. §4 discusses scalability issues while §5 describes the experiments conducted to validate the tool on several clusters, exploiting in particular the Grid'5000 infrastructure [3]. Finally, §6 concludes this article and provides some perspectives.

2 Context & Problem Statement

HPL [28] solves a dense N by N system of linear equations $A \times x = b$ (divided into blocks of size $P \times Q$) by Gaussian elimination with partial pivoting. As well as N, P and Q, fourteen further parameters control HPL's execution, and any system administrator who has tried to evaluate the computing power of a cluster with HPL can testify to the difficulty of manually tuning these parameters to maximize the benchmark result. The problem is due to the size of the search space and the fact that a single run can take more than half a day.

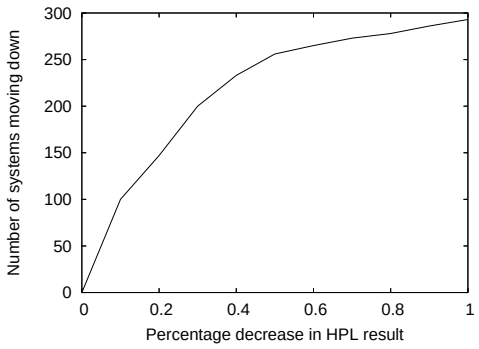

Fig. 1. Impact of HPL result reduction on the Top500 rank

Yet this tuning is of crucial importance as illustrated in figure 1, based on statistics from the latest Top500 list [4]. It shows how many systems would lose one place or more if their HPL result were slightly decreased.

Our previous work [8] promoted the idea that HPL can be seen as an *objective function* for an Evolutionary Algorithm (EA) in such a way that it faciliates and automates the tuning process. EA refers to a class of problem-solving techniques based on the Darwinian theory of evolution. A possible and acceptable solution *i.e.* a member of the *population* is called an *individual*. Each iteration (or *generation*) of an EA involves a set of genetic operators randomly applied to the individuals together with a competitive selection that weeds out poor individuals through the evaluation of a *fitness value* that indicates their quality as a solution to the problem. More details on EAs may be found in [15]. The EA in [8] is configured as follows. An individual corresponds to a set of eligible parameters for HPL. Its fitness value is the benchmark result when running HPL on the cluster with those parameters. Our initial study delegated the details of the evolutionary computation to ACOVEA, a framework initially designed to investigate the optimum combination of command-line flags for a compiler (ACOVEA stands for *Analysis of Compiler Options via Evolutionary Algorithm*). Using an adapter to match HPL to the ACOVEA interface when benchmarking a small cluster, spectacular results were obtained with little effort compared to classical hand-tuning.

This paper extends our initial proposal in two directions. Firstly, the details of the evaluation process to find the most suitable library are set out (see §3.1). Secondly, we describe an all-in-one framework called ACBEA (see §3.2) for benchmarking the computing power of a cluster through HPL. This tool is designed to download, build and launch HPL in such a way that the process of parameter tuning is handled internally and sequentially, starting from a small problem size and moving to the largest possible. Hopefully, this last configuration will produce the best benchmark result for the computing platform. ACBEA makes use of an EA to automate nearly all tedious processes such that the interaction of the user is limited to an initial setup, some manual tuning for the last step of the evaluation and finally the collection of the ultimate result (see §3.3). As before, our approach is based on

the assumption that individuals that produce good results in small, short benchmarks are likely to produce good results in larger, longer tests. This hypothesis follows from practical observations and is discussed in §4.

3 Acbea Software Components

The software harness used in [8] was assembled quickly using scripting tools. As such, it was difficult to run and to maintain, and suffered from a number of inefficiencies. For example, the evaluation of each set of HPL parameters required a batch job to be submitted to start a new instance of HPL on the cluster's compute nodes. Thus each evaluation incurred both batch submission and HPL start-up overhead. For the follow-up work documented here, a more flexible, efficient and maintainable package was developed from the ground up. The package was designed with the following constraints in mind:

- *Maximal portability.* The software package should build and run in as many UNIX-like environments as possible, and be able to utilize a choice of components for the following elements:
  - C and C++ compilation systems. GCC [31] and HP's compilers [1] were used in development, but commercial products such as Intel's [2] can also be used.
  - Batch job submission system. Development has been carried out exclusively with OAR [6], but hooks are provided to allow alternative schedulers.
  - BLAS (Basic Linear Algebra Subroutines) library implementation. We use ATLAS [33] as a default, but alternative implementations such as Intel's Math kernel Library [2] or GotoBLAS[16] can be used.
  - Message Passing Interface. OpenMPI [11] was mainly used, but alternatives such as MPICH2 [22] or Intel's MPI Library [2] are supported.
- *Minimal prerequisites.*
- *Liberal licence terms.*
- *No modification in HPL source code.* The source code of the program used to run the final benchmark must used exactly as it appears in the HPL distribution package, so as to make it clear that the result of the test is legitimate.

3.1 Choice of Evolutionary Algorithm Library

Thirteen evolutionary algorithm library packages, all written either in C or C++, were evaluated against five criteria:

1. *Portability.* The packages were built in four environments: FreeBSD, HP/UX, Linux and Mac OS X. Packages that built and passed their own test suites in all environments were given a higher score. Points were deducted if the build process was difficult and/or required additional packages.
2. *MPI support.* Two points were given to packages that included support for MPI. This consideration ultimately turned out to be unimportant, as Acbea runs the evolutionary algorithm in a single process on a cluster's head node, and so does not require MPI.
3. *Currency.* Packages having a recently-released revision were marked higher than those that had not been updated for some time. The thinking behind this was that a recent revision suggested the existence of an active development community that would be able to provide support if necessary.
4. *Maturity.* The initial release date and the revision history of each package were examined to judge its maturity. Packages that had been available for several years and which had been regularly updated were marked higher than new packages, or old packages that had seen few revisions.

Package	*Ver.*	*Date*	*Good builds*	*Bundle size*	Port-ability	MPI support	Curr-ency	Mat-urity	Size	*Total*
Evocosm[24]	3.3.1	2008	5	532kB	4	0	5	5	4	**18**
GAlib[32]	2.4.7	2007	5	368kB	4	0	4	4	4	**16**
Open BEAGLE[14]	3.0.3	2007	5	4.8MB	4	0	4	5	3	**16**
PGAPack[26]	1.1	2008	5	548kB	4	2	3	3	4	**16**
EO[21]	1.0.1	2008	4	972kB	3	0	5	3	4	**15**
GAtoolbox[29]	*n.a.*	2007	4	40kB	3	0	4	2	5	**14**
ParadisEO[5]	1.1	2008	4	20.5MB	3	2	5	3	0	**13**
BEAGLE Puppy[12]	0.1	2004	5	232KB	5	0	2	1	4	**12**
Push 3[20]	3.1.0	2006	1	332kB	1	0	3	3	4	**11**
SHARK[18]	2.1.3	2008	3	5.8MB	2	0	5	2	2	**11**
dBEAGLE[13]	0.9.2	2004	0	3.3MB	0	2	2	2	3	**9**
MOMHLib++[19]	1.10b	2004	0	1.6MB	0	0	2	3	3	**8**
TEA[10]	2.6.0	2004	0	148kB	0	0	2	2	4	**8**

Table 1. EA library evaluation

5. *Size.* Small packages were marked higher than large. It should be noted that the large packages support a wide variety of heuristic optimization methods. However, as it was not the aim of the research described here to test alternative methods, this was not considered an advantage.

The result of the evaluation is shown in table 1. The Evocosm [24] package scored highest, and so was chosen as a basis for ACBEA. Evocosm implements a classical evolutionary algorithm as described in [15]: individual experiments are described by a genome made up of genes representing parameter values for the experiment; genomes that produce good experimental results are more likely to be used in creating the genomes used in the next generation than those that produce poor results. Each individual in the next generation is created by choosing two parents, and selecting each gene in the new individual from one of the parents at random[1]. Individual genes may also mutate to a value that differs from either of the parent genes. Optionally, an elitist strategy may be used to preserve the best individuals. Additionally, Evocosm implements an island model *i.e.* it maintains several populations that exchange some individuals periodically. Note that this library also underlies the Acovea [23] framework used in our earlier work.

3.2 Acbea

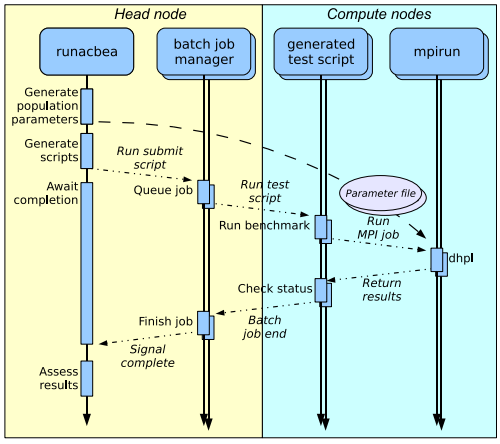

Fig. 2. ACBEA: evaluation of a population

ACBEA consists of a suite of programs that work together to automate the benchmark process. The most important of these is `runacbea`, which runs on the head node of a cluster and submits batch jobs for the cluster's compute nodes. The jobs are typically handled by a batch job manager.

`Runacbea`'s XML-format configuration file describes HPL's parameters and their allowable values. The file also contains information about the batch job manager and the implementation of MPI that is to be used.

The program's operation for a single population of benchmark evaluations is shown in figure 2. The sequence of operations is repeated for each population in a generation, and the overall sequence

[1] This differs from the classical concept of *crossover* in that no attempt is made to preserve groups of genes that are adjacent to one another.

is repeated until a specified number of generations has been run. If sufficient compute nodes are available, the task of fitness assessment for each population may be shared among several parallel jobs, so speeding evaluation. Each batch of benchmarks is run using MPI to launch multiple copies of `dhpl`, a customized variant of HPL' s `xhpl` benchmark program. While `xhpl` uses a short configuration file to describe a series of related tests, `dhpl` uses a file of arbitrary length to define the series of unrelated tests that represents all or part of a population. Conforming to the constraints presented in §3, the HPL problem solution code is unchanged. The result output format has been changed as little as possible. On terminating, `runacbea` summarizes its findings and produces a number of output files. The first contains a configuration for a subsequent run with a problem double the size on four times the number of cores. As four times compute power is being applied to a problem having eight times the complexity, each benchmark will take almost twice as long as those defined by the original configuration file. In order that the subsequent run may explore only the more profitable parts of HPL's parameter landscape, the parameter values allowed by the new configuration file exclude those which appear only in most poorly-performing 33% of individuals in the run. (This cut-off level may be changed.) The remaining outputs are configuration files for `xhpl`, representing the parameters that produced the best-performing individual in the each population of the final generation. These files may be used to run `xhpl` benchmarks directly. The decision to host `runacbea` on the head node of a cluster may be questioned, as the intention is to benchmark the compute nodes, while the main task of the head node should be to run administrative housekeeping functions for the cluster. In fact, `runacbea` may itself be viewed a housekeeping program: tests show that it and its child processes consume perhaps five seconds of processor time over an entire run, during which the compute nodes may clock up hundreds of hours.

3.3 The Benchmarking Process

The benchmarking process with ACBEA involves the following steps:

1. Gather information about the target cluster: nodes, cores and memory per node, MPI implementation, batch job manager ...
2. Use the provided `ten-sec-n` utility to obtain a value of N that makes HPL run for ten seconds on a single core . Let N_{ten_sec} be this value.
3. Edit the `runacbea` configuration file to create one suitable for testing all the cores in a small group of nodes n — four has been found to be a reasonable choice for n. The value of N in this file may be calculated using $N_{4\_nodes} = N_{ten\_sec} \times 0.7 \times \sqrt[3]{compute\_cores}$. The 0.7 factor compensates for the fact that no inter-node communication is used during the determination of N_{ten_sec}.
4. Optimize HPL configuration for a benchmark on the small group of nodes. In this step, `runacbea` runs an EA on five populations of forty individuals each for twenty generations. Each individual is evaluated in around ten seconds so this step may take half a day if a single group of nodes is used. The evaluations may be done in parallel over several groups to reduce the time required.
5. Use the best parameters found in step 4 for a new optimization run on groups of nodes four times larger (*i.e* sixteen if step 4 used four), solving problems of double the size: $N_{n^i\_nodes} = 2^{i-1} N_{n\_nodes} \forall i \geq 2$. Repeat this step until you reach a solution suitable for node groups having a size as near as possible to (but not exceeding) the number of nodes in the cluster.
6. Use the best configuration found at the previous step for the final benchmark evaluation on the full cluster. The problem size for this run can be calculated from the cluster's installed memory with the following formula:
   $N_{full\_theoretical} \simeq 0.8 \sqrt{\text{Total Memory Size in bytes} \times \frac{\texttt{sizeof(double)}}{8}}$

The perfect value of N should be manually adapted from $N_{full\_theoretical}$ by monitoring the memory usage on the cluster nodes to avoid swapping. This is an activity that ACBEA does not currently automate. Each run of this last step takes one hour on a cluster having up to 500 cores and 1–2 GiB of memory per core. Note that it is the only step that requires full cluster reservation.

7. Choose the best result for publication as the HPL benchmark score.

4 Scalability

The methodology implemented by ACBEA is based on two assumptions:

1. A single run of an experiment will produce a result that is representative of the results of multiple runs of the same experiment.
2. HPL parameters that produce good results in small, short benchmarks are likely also to produce good results in larger, longer tests.

If the first assumption is not true, the fitness values used by the EA may not be correct, with the result that the next generation does not reflect the genomes of the truly most fit individuals. This issue is investigated in §4.1. If the second assumption is false, there is no point in trying to use small benchmarks to explore HPL's parameter space; large, long-running tests would be the only ones that could yield useful information about full-cluster benchmarks. §4.2 reports on tests of scalability.

4.1 Individual Benchmark Repeatability

A series of tests was run on fifteen two-core nodes of the Chaos cluster (see table 2) to investigate the variability in the results obtained from repeated runs of the same test. As figure 3 demonstrates, variance expressed as a percentage of the result value drops rapidly at first, but the improvement becomes slower as run time increases. This suggests that with this configuration, an N chosen to give a run time of approximately twenty seconds provides a reasonable compromise between the duration of an ACBEA run (which typically entails 4,000 individual benchmarks) and the expectation that a single result is representative[2]. In further tests (not reported here), variability reduced (and, of course, execution time increased) as the number of nodes assigned to the problem was reduced. Consequently, an execution time of ten seconds is sufficient for benchmarks involving a small number of nodes.

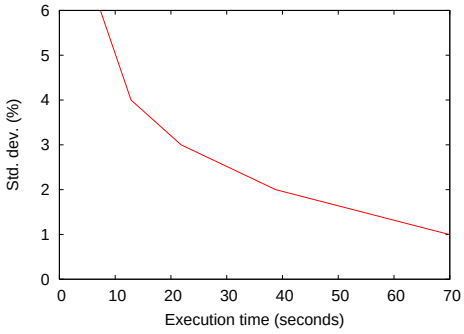

Fig. 3. Variance in results of repeated tests

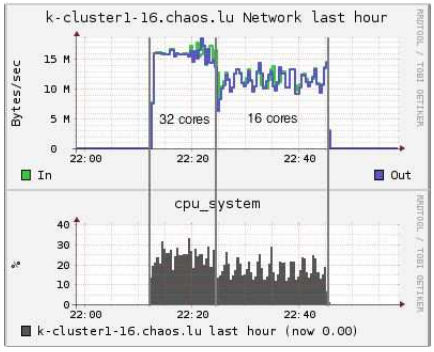

Fig. 4. The effect of a badly-sized test

An alternative way of interpreting the findings is that the problem must not be too small for the number of cores allocated to solve it. If it is, communications activity begins to dominate calculation, resulting in performance figures that are both poor and highly variable. This is illustrated in figure 4, which shows the system

[2] Better interconnect than Gigabit Ethernet was found to reduce variability.

CPU time used by a dual-core cluster node involved in solving the same problem ten times, first on 32 cores, then on sixteen. In the 32 core case on the left, the percentage of system time is higher, indicating that the problem is badly sized for the larger number of cores.

4.2 Interdependence Between Parameters

Our earlier paper [8] reported an investigation into the effect of N, problem size, on the optimum value of NB, block size, a parameter found to have a large effect on performance. The conclusion was that the two were independent, with the result that small problems could be used to determine an optimum value of NB that would also be valid for large problems. The work did not investigate the scalability of other parameter combinations, nor did it check whether the findings were specific to the Intel platform, or to the Linux libraries and tools used. Further studies reported here address these issues, and broadly confirm that the results of small benchmarks may be used as a basis for larger experiments.

N versus NB. In order to check whether the earlier conclusion was true in general, similar tests were run on other platforms and with a variety of BLAS implementations. Space precludes reporting these in detail, but they confirmed the original findings. Figure 5 shows representative results obtained with two different BLAS libraries on a four-core HP/PA host running HP/UX.

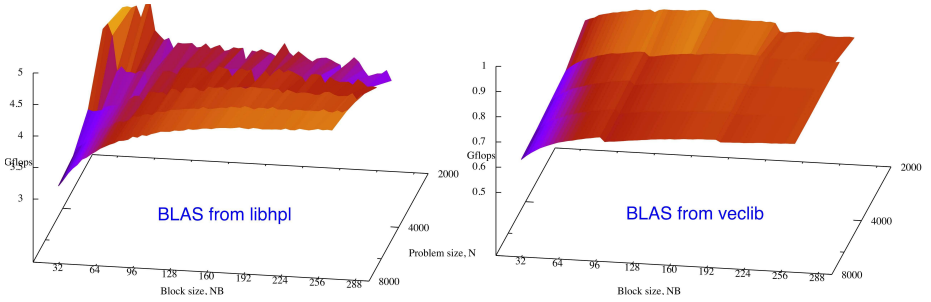

Fig. 5. NB versus N on HP Precision Architecture

P, Q versus N. To divide work among a number of compute nodes, HPL configures the nodes into a $P \times Q$ matrix. The shape of the matrix affects communications patterns and volumes between particular pairs of nodes. An investigation was carried out into whether a shape that was optimal for small problems was also optimal for large. Figure 6 shows a sample of the results. Increasingly large problems are solved while P and Q are varied, keeping their product, and HPL's other parameters constant. It can be seen that the ordering of the curves barely changes as N is increased, suggesting that information gained from small problems about matrix shape can be applied in large problems. Because adjacent curves do cross on occasions, ACBEA includes new dimensions that are related to the old when creating the configuration file for a subsequent run.

SWAPPING versus N. Studies were also carried out on several machines into the scalability of the $SWAPPING$ parameter, which determines when HPL switches from one data-exchange strategy to another, and which has been observed to have much less effect on benchmark performance than NB or $P \times Q$. Again, the trials suggested that a $SWAPPING$ value that produces good results in small benchmarks will also produce good results in large.

5 Cluster benchmarking

This section is concerned exclusively with the results of the ACBEA package's automatic tuning of HPL parameters; while it would be instructive to compare automatically-produced results with those obtained by other methods such hand-tuning,

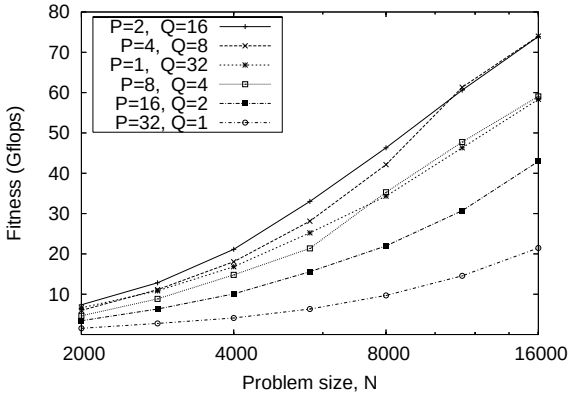

Fig. 6. Relative performance of matrix shapes versus N

or the spreadsheet-assisted procedure proposed in [7], any such study must be the subject of future work. Table 2 describes the clusters that were targeted and the results achieved. It has been remarked, for example in [25,27], that HPL is a good tool for "shaking down" compute clusters. This was certainly found to be the case when ACBEA was built and run on a variety of hosts. Consequently, we are able to report fewer final results here than might have been hoped. More complete descriptions of the French systems that participate Grid'5000 may be found in [3].

Cluster name	Location	CPU type/ speed (Ghz)	Total cores	Mem/ core	Inter-connect	MPI	Gflops/ cores
capricorne	Lyon	Opteron/2	112	1GiB	1GE, Myri-2000	MPICH	48/32
chaos-b	Luxembourg	Xeon/3.4	16	4GiB	1GE	OpenMPI	55/16
chaos-k	Luxembourg	Pentium D/3.2	32	2GiB	1GE	OpenMPI	98/30
chinquint	Lille	Xeon/2.8	368	1GiB	Myri-10G	OpenMPI	160/32
genepi	Grenoble	Xeon/2.5	272	1GiB	1GE	MPICH	45/8
granduc	Luxembourg	Xeon/2	176	2GiB	1GE	OpenMPI	671/168
violette	Toulouse	Opteron/2.2	114	1GiB	1GE	OpenMPI	262/96

Table 2. ACBEA target systems

Chaos-b, Luxembourg. `Chaos-b` consists of just two eight-core nodes. The full ACBEA procedure was run, and a benchmark score of 55.05 Gflops was obtained with $N = 25,600$, $P = 1$, $Q = 16$. This is a considerable improvement upon the disappointing 26 Gflops reported for the same cluster in [8]. The reason for this discrepancy is not known, although the current tests used a better-optimized BLAS library. A study was also made of the repeatability of the ACBEA process: do repeated runs produce similar or identical recommendations for optimum parameters? The results of four trials of the first optimization phase were in broad agreement. For example, two of the trials gave 72, 96, and 104 as the allowed values for NB in the second phase. (The others gave just 72 and 96, and 72, 104 and 144 respectively.) Other parameter choices were also similar or identical across the four runs. This suggests that the ACBEA process is repeatable — although see the discussion of problem sizing in 4.1.

Chaos-k, Luxembourg. This sixteen-node cluster of two-core nodes was extensively benchmarked for [8], attaining 116 Gflops. One of its nodes was unavailable during the testing reported here. Also, a new and larger Linux kernel made it impossible to use the $N = 84,000$ value used in those tests. Consequently, results are not comparable. After a full run of ACBEA, the five resulting `xhpl` configuration files were used to obtain a best result of 98.47 Gflops with $N = 80,000$, $NB = 88$, $P = 3$ and $Q = 10$. The parameters were derived from those of the fifth-most-successful individual in the optimization run, suggesting that the "best-of-best" individual does not always deliver parameters that are optimum in a larger benchmark.

Granduc, Luxembourg. Currently the largest of the University of Luxembourg's clusters, `granduc` was able to run the full ACBEA procedure. One node being off-line, the final benchmarks were run on 21 nodes (168 cores), giving a best result of 671 Gflops with $N = 192,000$ (using almost all available memory), $NB = 112$, $P = 2$ and $Q = 84$.

Capricorne, Lyon. The `Capricorne` cluster is used by Grid'5000 for experimental work, and was targeted as a test of ACBEA portability because it differs in three respects from the Luxembourg clusters: AMD rather than Intel processors; MPICH[17] instead of OpenMPI; and Myriad high-speed interconnect in addition to gigabit Ethernet. Unfortunately, we were unable to configure MPICH to use the Myriad for data transport, so fell back to using the slower, higher-latency Ethernet. Poor figures were obtained from an initial ACBEA run using eight cores on four compute nodes: the best-performing individual benchmark reached 15.33 Gflops. A second run targeting 32 cores on sixteen nodes obtained a best result of 44.84 Gflops. Because of these disappointing figures, a final test utilizing all cores was not run; the reason for the poor performance was investigated instead. The cause of the problem was found to be incorrect allocation of processes to nodes by MPICH: some nodes were over-subscribed, some under-, and some had the correct number of processes. The reason for this behaviour could not be determined, and the benchmarking attempt was abandoned.

Chinqchint, Lille. A recently-commissioned and powerful system having 368 cores on 46 nodes with ten gigabit Myriad interconnect, `chinquint` proved too unreliable to obtain anything approaching a full-system benchmark. It was possible to run two parallel four-node (32 core) tests for `runacbea`'s full twenty generations. The best individual test delivered an impressive benchmark result of 160.50 Gflops. This was almost twice the overall average of 83.11 Gflops in the final generation. Such a discrepancy is unusual. Sadly, it was not possible to find sixteen nodes reliable enough to run the next stage of the test, since it should have been possible to obtain well over a teraflop from the whole cluster.

Genepi, Grenoble. Like `capricorne` (see above), `genepi` has MPICH installed on its compute nodes. A first run of ACBEA targeting the eight cores and using eight parallel jobs yielded an average performance of 41.65 Gflops, with the best individual benchmark achieving 44.78. By confining benchmarks to single nodes, this configuration made essentially no use of the interconnect. Sadly, several attempts to run the next stage of the ACBEA process on 32 cores failed to run to completion due to intermittent MPICH problems with secure login between nodes. The experiment was consequently abandoned.

Violette, Toulouse. It was possible to run the complete ACBEA process on `violette` using its installed OpenMPI package. Both stages of optimization performed as expected, delivering five `xhpl` configuration files for final benchmarking. As the cluster has 114 cores (of which some were unavailable) rather than the 64 targeted by the configuration files, the P and Q parameters were adjusted to address 96 cores before final benchmarks were run using $N = 97,600$, a value that was found almost to saturate the nodes' memory. A peak score of 262.3 Gflops was obtained from sixty evaluations derived from the parameters of the five best-performing individuals in the second-stage optimization. As expected, the best result was obtained using the parameters of the "best-of-best" individual. Surprisingly, it used a layout of $P = 16$, $Q = 6$, although over-square matrices generally perform poorly.

6 Conclusions and Future Work

This paper has described how an evolutionary algorithm may be used to produce competitive HPL benchmark results for a computing cluster without the need for

intimate knowledge of the benchmark program, or of the software needed to support it. The ACBEA package has proved to be portable to a number of systems, although these have been fairly uniform in operating environment, batch job management and so on. However, portability alone is not sufficient: the target system must be sufficiently robust to support both the demanding benchmark and an evolutionary harness that launches it many thousands of times during the course of an evaluation. At the current state of development, ACBEA still requires a fair amount of knowledge on the part of its user. For example, is up to the user provide the parameters passed to the MPI implementation, which can have a great effect on its performance, and consequently on that of the benchmark as a whole. Files must also be edited by hand to set up a starting problem size, and to define the node topology to be used for the evolutionary process. This done, the user must step through the lengthy procedure described in §3.3 in order to obtain a benchmark result. Future work will be focused on increasing ACBEA's ease of use, and on using discovery techniques to reduce the amount of information that must be supplied before a benchmark can be run.

The focus of this paper has been on obtaining results: no attempt has been made to compare ACBEA's results with figures that have been independently obtained by hand-tuning or other methods. It would be instructive to make such comparisons in the future. The work reported in §4 suggests that HPL's other parameters are largely orthogonal to N, the problem size, but does not suggest theoretical or physical reasons as to why this might be the case. Also, all clusters tested to date have provided a fully-interconnected communications topology, which strongly favours a $BCAST$ parameter of zero. Consequently, no information has been obtained as to whether $BCAST$ is scalable or not. Future work could potentially address both of these issues.

The authors would like to thank the administrators and support staff of the Grid'5000 project for their assistance.

References

1. HP C++. [Online] `www.hp.com/go/c++`.
2. Intel Software. [Online] `software.intel.com/en-us`.
3. The Grid5000 Project. [Online] `www.grid5000.fr`.
4. The Top500 project. [Online] `www.top500.org`.
5. S. Cahon, N. Melab, and E-G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *J. of Heuristics*, 10(4):357–380, May 2004.
6. N. Capit and al. A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid05)*, 2005.
7. Microsoft Corporation. Building and Measuring the Performance of Windows HPC Server 2008-Based Clusters for TOP500 Runs. Technical report, November 2008.
8. D. Dunlop, S. Varrette, and P. Bouvry. On the Use of a Genetic Algorithm in High Performance Computer Benchmark Tuning. In *IEEE International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'08)*, pages 105–113, Edinburgh, UK, June 2008.
9. R. Eigenmann, G. Gaertner, W. Jones, H. Saito, and B. Whitney. SPEC hpc2002: The next high-performance computer benchmark. In *ISHPC*, pages 7–10, 2002.
10. M. Emmerich and al. TEA (Toolbox for Evolutionary Algorithms).
11. E. Gabriel et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, Sept 2004.
12. C. Gagné. BEAGLE Puppy. [Online] `beagle.gel.ulaval.ca/puppy`.
13. C. Gagné, M. Parizeau, and M. Dubreuil. Distributed BEAGLE: An Environment for Parallel and Distributed Evolutionary Computations . In *Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS) 2003*, pages 201–208, May 11-14 2003.

14. C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *Intl J. on Artificial Intelligence Tools*, 15(2):173–194, April 2006.
15. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.
16. K. Goto and R. Van De Geijn. High-performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw.*, 35(1):1–14, 2008.
17. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.
18. C. Igel and al. SHARK. [Online] `sourceforge.net/projects/shark-project`.
19. A. Jaszkiewicz and G. Dąbrowski. MOMH multiple-objective metaheuristics.
20. M. Keijzer and al. Push Programming Language.
21. M. Keijzer, J.J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: A general purpose evolutionary computation library. In *5th European Conference on Artificial Evolution*, pages 231–244, London, UK, 2002. Springer-Verlag.
22. Argonne National Laboratory. MPICH2: High-performance and Widely Portable MPI. [Online] `www.mcs.anl.gov/research/projects/mpich2`.
23. S.R. Ladd. Acovea: Using Natural Selection to Investigate Software Complexities. [Online] `www.coyotegulch.com/products/acovea/`, 2007.
24. S.R. Ladd. Evocosm: A C++ Framework for Evolutionary Computing. [Online] `www.coyotegulch.com/products/libevocosm/`, 2007.
25. J. Levesque. Breakthrough Science on a Petaflop XT5. In *Cray XT Workshop*, 2009.
26. D. Levine. Users Guide to the PGAPack Parallel Genetic Algorithm Library. [Online] `ftp://info.mcs.anl.gov/pub/tech_reports/reports/ANL9518.ps.Z`, 1996.
27. T. Minyard and al. Experiences and Achievements in Deploying Ranger, the First NSF "Path to Petascale" System. In *TeraGrid'08*, June 2008.
28. A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL — A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. [Online] `www.netlib.org/benchmark/hpl/`, Jan 2004.
29. K. Sastry. Single and Multiobjective Genetic Algorithm Toolbox in C++. [Online] `www.illigal.uiuc.edu/pub/papers/IlliGALs/2007016.pdf`, 2007.
30. V. Sripathi and A. Krishnan. Analyze and optimize the HPL benchmark on x86-64 cluster. Technical report, North Carolina State University, 2008.
31. R.M. Stallman et al. *Using GCC: The GNU Compiler Collection Reference Manual*. FSF, 2005.
32. M. Wall. GAlib — A C++ Library of Genetic Algorithm Components.
33. R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, Jan 2001.