Building and Validating a Reduced TPC-H Benchmark

Hans Vandierendonck
Ghent University
Dept. ELIS
St.-Pietersnieuwstraat 41
B-9000 Gent, Belgium
hans.vandierendonck@elis.ugent.be

Pedro Trancoso
Department of Computer Sciences
University of Cyprus
75 Kallipoleos Ave., P.O.Box 20537,
1678 Nicosia, Cyprus
pedro@cs.ucy.ac.cy

Abstract

The properties of computer system such as performance, energy, reliability, etc. are commonly evaluated by running benchmarks. However, the benchmarking process is complicated to set-up and use and running the benchmarks takes a substantial amount of time. Furthermore, when designing a computer, architects resort to simulation of the system, increasing the benchmarking time by several orders of magnitude. This problem can be alleviated by reducing the execution time of the benchmark suite. In this paper, we investigate a method to reduce the number of queries in TPC-H, a decision support system benchmark. Our evaluation shows that out of the 22 original queries, a subset of only 6 achieves a high representativeness at only 40% of the original execution time. The results also show that subsets built for a particular database size may be used for evaluating computer systems with other database sizes. Finally, we validate our approach against the full benchmark execution for a set of architecture case studies. The validation results show that the proposed subsets lead to similar conclusions than the ones drawn using the full benchmark.

1. Introduction

Benchmarking, *i.e.* evaluating a computer system by running a set of representative programs, is a complex and time-consuming task. By running a set of these programs, which we call benchmarks, on a computer system, performance analysts can characterize the system and rank it against the others. Benchmarking is usually performed to market and select commercial systems. In addition, obtaining accurate performance characterization is crucial during the development of a computer system. This helps in steering the design of the computer towards the project goals.

As a result, *benchmark suites*, *i.e.* a collection of benchmark programs, have been developed for many market segments, as well as many application categories. Well-known examples are the SPEC CPU [7] benchmark suite representative for scientific applications, the EEMBC [9] representative for embedded systems and TPC-C [4] and TPC-H [3] representing database applications belonging to the class of online transaction and decision support system, respectively.

However, setting up and running such benchmark suites can be complex and time-consuming. This is especially true for the TPC-H and TPC-C benchmark suites. Setting up the benchmarks and the experimental environment requires some level of mastery. Finally, the full setups of common benchmark suites are too large for simulation purposes [12].

Consequently, researchers have investigated ways to reduce the "load" of benchmark suites. By *subsetting* a benchmark suite, i.e. selecting a minimal but representative subset of the individual benchmark programs in a suite, the execution time and simulation time can be strongly reduced while maintaining high fidelity to the original suite [5, 6, 11, 14]. Achieving this goal is possible as individual benchmarks frequently show similar performance, as well as a similar sensitivity to various aspects of performance, e.g. memory latency and bandwidth, aggressiveness of speculation, issue width, etc. This similarity implies that multiple benchmarks will give the same information from a performance evaluation experiment. Hence, there is redundancy among the benchmarks and the performance of several benchmarks may be extrapolated from the performance of a single one of them.

The similarity of individual benchmarks can be determined in several ways. One method is to count low-level events such as the number of cache misses, instruction mix and branch prediction accuracy [6]. Although

these metrics are detailed, it is hard to prove that any particular set of low-level metrics is sufficient to define similarity for all aspects of performance. Therefore, in this paper, we use real performance results: the execution times of the benchmarks running on a set of real computers. Similarity between benchmarks is determined based on the fact that the benchmarks *rank* the computers in the same way.

In this paper, we apply benchmark subsetting to the TPC-H benchmark, a decision support system (DSS) database workload [3]. First, we discuss related work in Section 2. In Section 3, we present a subsetting method based on real execution times and apply it to the TPC-H benchmark in Section 4. The representativeness of the subsets is evaluated for various database sizes. The basic method is further analyzed in Section 5. In Section 5.1, we investigate whether a subset derived for one database size is valid also for a different size. Then, in Section 5.2 we investigate optimizations to the method. Next, we evaluate the applicability of our method with respect to research in micro-architecture in Section 6. In particular, we show that the subsets guide an architect to the same design choices as the full set of queries in a few case studies. Finally, Section 7 summarizes the main conclusions of paper.

2. Related Work

Reducing the benchmarking effort has become an important topic in the research literature. Three approaches can be identified. As all three approaches are highly orthogonal, their techniques may be applied simultaneously.

2.1. Sampling

The execution of a benchmark manifests itself in several distinct but repeating phases [13]. To accurately quantify performance, it suffices to measure each phase only once. Furthermore, simulation time can be further reduced by considering only a small sample of each phase. This method is implemented in the SimPoint tool and has become very popular in computer architecture research.

SMARTS [17] implements a different sampling technique. Instead of purposefully looking for structure in the execution of a benchmark, they propose to take may short samples from the benchmark. This approach obtains a good match with the overall benchmark. Yi *et al.* [18] found that SMARTS is slightly more accurate than SimPoint. Note that sampling is only applicable to simulation, while the other approaches are applicable to all benchmarking efforts.

2.2. Reducing Benchmark Size

It is not always mandatory for a benchmark to have a long execution time in order to show a particular behavior. This behavior may also present itself for a slightly different input data set, resulting in a lower execution time. However, constructing such smaller input data sets is not always straightforward and a deep understanding of the benchmarks is required to achieve good results.

KleinOsowski and Lilja attempted this for some of the SPEC CPU 2000 benchmarks, which they call *MinneSPEC* [8]. They manually cut down the input files and compared the benchmark characteristics to the original versions. Their versions of the benchmarks require much less simulation time. However, it has been shown that several of their benchmarks are not representative for the original ones [6].

A similar attempt was made for the TPC-H and TPC-C benchmarks by Shao, Ailamaki and Falsafi [12]. They identify three dimensions along which the benchmarks can be scaled: workload complexity (i.e. number of queries), dataset size and concurrency level. They note that it is important to keep these three dimensions in the correct relationship to each other, in order to abide by the scaling rules of the TPC specifications. Thus, they analyze what database operations are typically activated and reduce the benchmarks in such a way that the mix of database operations is not affected. Using this method, they claim to capture $> 95\%$ of the processor and memory performance.

2.3. Subsetting

Subsetting the SPEC CPU benchmarks was undertaken by Eeckhout, Vandierendonck and De Bosschere [6]. They measure low-level performance metrics such as instruction count, instruction mix, cache miss rates, etc. and apply principal components analysis and hierarchical cluster analysis techniques to determine benchmark clusters. This method is later refined by substituting independent components analysis for the principal components analysis [5].

Phansalkar *et al.* [11] argue for the use of so-called micro-architecture independent characteristics to determine program similarity. Such characteristics are more abstract (e.g. locality instead of cache miss rates) and hence should be applicable for any micro-architecture. Unfortunately, they do not compare their micro-architecture independent characteristics to the low-level characteristics.

A number of disadvantages can be identified when measuring benchmark similarity using low-level char-

acteristics. First note that the measurements of all of these characteristics yields a vector. The similarity between two benchmarks is expressed as the Euclidean distance between the respective vectors. Many characteristics will, however, measure similar properties. For example, cache miss rates for different cache configurations tend to be correlated [15]. This needs to be compensated for by computing the correlation between characteristics across *a set of benchmarks* and accounting for this correlation in the similarity metric [6]. Thus, according to this approach, the similarity between two benchmarks depends on the other benchmarks that are included in the study. Furthermore, it is very hard to know whether the right and sufficient characteristics have been identified.

The similarity metric used in this work does not have these disadvantages. It is investigated in [16], where it is shown that the type of similarity metric used is not the major determining factor in the representativeness of benchmark subsets. Instead, the clustering algorithm has a larger impact. Furthermore, the representativeness obtained with any clustering technique my increase or decrease randomly when increasing the number of benchmarks in the subset. This similarity metric was also applied when subsetting the TPC-H benchmarks, showing an 80% representativeness while reducing the execution time to 20% [14]. This paper extends [14] by further investigating and refining the subsetting procedure and providing a preliminary validation for architectural research.

3. The Benchmark Subsetting Method

The process of subsetting a benchmark suite involves three steps. The first step involves deciding on the use and goals for the benchmarking procedure. After taking this decision, it is possible to choose a metric for classifying which benchmarks are more similar and which are less similar. This metric is known as the *metric of similarity*.

The second step is dividing the benchmarks into groups or clusters with the most similar benchmarks placed in the same cluster. This procedure can be automatically performed using statistical data analysis techniques.

Finally, in the third step, one must select a representative benchmark from each cluster. Several heuristics can be used for this selection procedure. The benchmark subset is composed of all those benchmarks that are selected during this last step.

3.1. Similarity of Benchmarks

In this paper, we assume that two benchmarks are similar provided that their execution results in the same relative performance for the computer systems. Running a benchmark on a set of computers tells you which computer is the fastest for that benchmark. However, running a different benchmark may result in a different ranking of the computers. If the rankings of the computers are virtually the same, then there is no purpose in running the second benchmark, as it gives little additional information. Therefore, the second benchmark is redundant and may be removed from the benchmark suite.

To argue the validity of our similarity metric, let's assume that we have two machines where machine A has a very high-performance memory system, while machine B is better at exploiting instruction-level parallelism. If we now have two benchmarks that are equally dependent on available memory bandwidth, then both will rank machine A as faster than machine B. Our similarity metrics considers these benchmarks similar and one of them may be dropped from the suite.

When a third benchmark has little requirements for high memory bandwidth but depends more ILP, then it may show that machine B is faster. This third benchmarks is different from A and B according to our similarity metric and it should be retained in the benchmark suite.

3.2. Determining Clusters of Similar Benchmarks

Given a metric of similarity, the set of available benchmarks is split into clusters in such a way that benchmarks that are most similar to each other are placed in the same cluster. This can be achieved by means of statistical cluster analysis techniques such as *K-means clustering* [10].

First, we prepare the data for the K-means algorithm by representing each benchmark as a point in the space of all conceivable benchmarks. The coordinates of a benchmark in the benchmark space are derived from the performance measurements available for that benchmark. In particular, each computer system is associated to a particular axis of the benchmark space. The coordinate of the benchmark on a particular axis equals the performance of the benchmark on the computer associated to that axis.

K-means clustering is a technique to cluster n points, in our case benchmarks, such that close points are clustered together. The number of clusters to compute is provided as an input to the algorithm. The

algorithm first computes K random cluster centers. During each iteration of the algorithm, each point is assigned to the cluster with the nearest cluster center. The cluster centers are then recomputed and set equal to the geometrical center of the cluster. These steps are iterated until no more changes occur.

Before starting the K-means clustering, it is necessary to normalize the data. The execution time measurements for each computer system must be normalized such that the mean across the queries is zero and the standard deviation is one. Otherwise, computer systems with very large execution time could dominate the distance metric and skew the results.

3.3. Selecting Cluster Representatives

Once clusters of similar benchmarks are constructed, a subset is obtained by selecting one benchmark from each cluster. The chosen benchmark should be representative for the majority of benchmarks in that cluster. The most obvious candidate is the benchmark closest to the centroid of each cluster. In this paper, we will show that this *closest-to-centroid* heuristic is not optimal. Therefore, we formulate alternative heuristics.

As the goal of subsetting a benchmark suite is to reduce execution time, the *minimum-execution-time* heuristic selects from each cluster the benchmark with the smallest execution time. This minimizes execution time, hopefully without sacrificing representativeness.

The *single-most-accurate* heuristics checks for each benchmark how representative this single benchmark is of the full suite, *i.e.*, how similar the ranking of the computers obtained by this benchmark is to the ranking obtained by the full benchmark suite. The metric used to evaluate how similar two ranks are, is explained in the next section.

Finally, we also consider selecting a *random* benchmark from each cluster. Well-motivated choices should be more representative then random choices, allowing us to gauge the quality of the heuristics.

3.4. Quantifying Representativeness of a Subset

As illustrated above, K-means clustering can be easily applied to compute a subset of the queries that might yield similar results in a performance evaluation experiment. If this is true, we say that the subset is representative for the full set of queries. We evaluate representativeness by Kendall's Tau rank correlation metric [2]. This metric checks how far the ranking of computers obtained using the subset resembles the ranking obtained by the full set. It checks for every pair of computers whether the faster computer according to the subset is also the faster according the full set (*concordance*) or not (*discordance*):

$$\tau = \frac{concordances - discordances}{number\ of\ pairs}$$

The τ metric is, like a correlation coefficient, comprised between -1 and 1 and should be interpreted as such. Thus, a Tau value of 1 is ideal and a Tau value below 0.5 is not acceptable for our goals.

4. Subsetting the TPC-H Benchmark

4.1. Experimental Setup

TPC-H is a decision support system benchmark that consists of a suite of business oriented queries. Both the queries and the data that populate the database were carefully chosen in order to keep the benchmark's implementation easy but at the same time to be relevant. The benchmark is composed of 22 read-only queries and 2 update queries. These queries are performed on considerably large amounts of data, have a high degree of complexity, and were chosen to give answers to critical business questions. The queries represent the activity of a wholesale supplier that manages, sells, or distributes a product worldwide. Both the queries and data are provided by TPC.

In particular, queries are created by a program named *qgen* while data is generated by a program called *dbgen*. The data size ranges from 1GB up to 10000GB. The data is loaded and the queries are executed on a regular DBMS system.

As with other benchmarks, in the case of TPC-H, the results obtained from the execution of the benchmark on a particular computer system are published by the manufacturers of the system and found on the TPC website (`http://www.tpc.org/`). Here, performance measurements are published for various database sizes of 100GB, 300GB, 1000GB and 3000GB. As of the time we collected the results, the number of evaluated systems was 16, 15, 15, and 8, respectively. There are also performance measurements for 10000 GB databases, but we did not use those as there was only one published measurement in this category at the time of performing this research.

Clustering is performed using the K-means procedure built into Matlab (`http://www.mathworks.com/`). Kendall's Tau is computed using an Excel Macro.

Table 1. Four-query clusters constructed using the measurement results for different database sizes. Cluster representatives chosen with the closest-to-centroid heuristic are shown in boldface.

	100 GB	300 GB	1000 GB	3000 GB
cluster 0	Q2–3, Q6, Q11, Q14, **Q16**, Q17, Q19–20, Q22	Q2, Q6, **Q11**, Q12, Q14, Q15, Q22	Q3–5, Q7–8, **Q10**, Q12–13, Q17, Q19–20	Q2–8, **Q10**, Q11, Q14–16, Q19–20, Q22
cluster 1	Q4–5, **Q7**, Q8, Q10, Q12, Q15	Q1, Q5, **Q7**, Q8, Q16, Q17, Q19	**Q2**, Q6, Q11, Q14–15, Q16, Q22	**Q17**
cluster 2	**Q1**, Q13	Q3, **Q4**, Q10, Q20	**Q1**, Q9	**Q1**, Q9, Q13
cluster 3	Q9, Q18, **Q21**	Q9, Q13, Q18, **Q21**	Q18, **Q21**	Q18, **Q21**

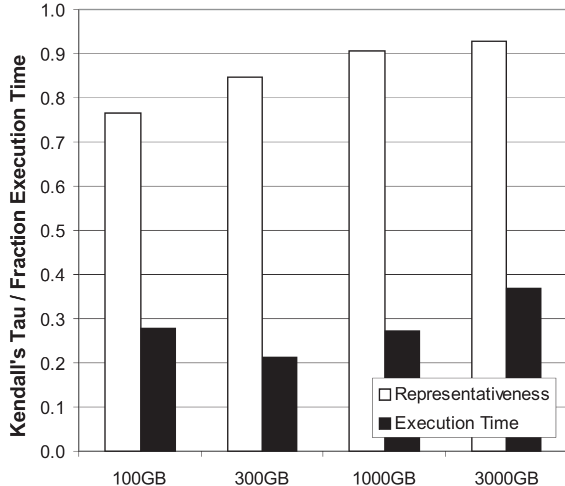

Figure 1. Representativeness and execution time of a 4-query subset for different database sizes.

4.2. Clustering Results

Applying K-means clustering on the TPC-H data yields clusters as listed in Table 1. We computed clusters of different sizes, but show results only for $K = 4$ clusters for brevity. We used the TPC-H performance data for the database of sizes of 100 GB, 300 GB, 1000 GB and 3000 GB. The cluster representatives are selected using the closest-to-centroid heuristic and are shown in boldface.

In the case of the 100 GB database, the algorithm finds 2 large clusters containing 10 and 7 queries, respectively and 2 small cluster, containing 2 and 3 queries, respectively. Thus, our method detects that many queries are considerably similar, but a few are dissimilar to the others. These "eccentric" queries may hold more information than the rest of the queries and be more valuable to the performance analyst [15].

Analyzing the performance measurements for different database sizes yields slightly different clusters. The measurements for each database size are performed on different computers, as each computer is designed for a particular market segment. This fact explains slight differences between the cluster compositions.

On the other hand, there are sometimes strong resemblances across database sizes. For example, queries 18 and 21 are invariably located in the same cluster, emphasizing a strong resemblance. There is a somewhat weaker resemblance between queries 1, 9 and 13: at least two of these queries are located in the same cluster. Finally, one frequently finds the same representatives across database sizes.

Figure 1 shows Kendall's Tau for subsets of 4 queries and based on data for different database sizes. The cluster representatives where chosen using the *closest-to-centroid* heuristic. There is a clear trend that subsets computed for the larger databases are more representative. This is caused in part by having fewer measurements for the larger databases, which require fewer queries to reconstruct them.

The final goal of subsetting the queries is to reduce the benchmarking time. Figure 1 also shows, with dark

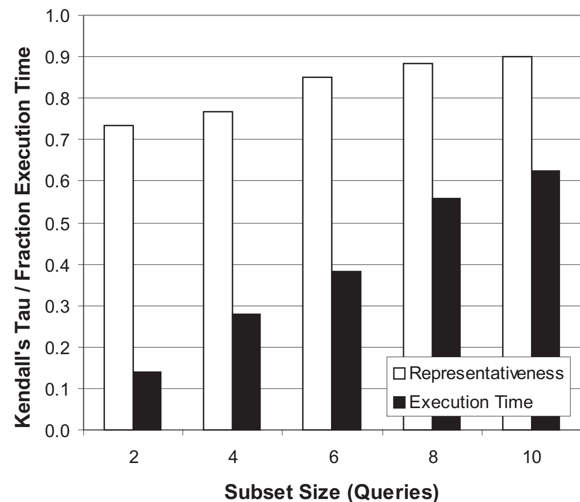

Figure 2. Representativeness and execution time of subsets of different size for the 100GB database.

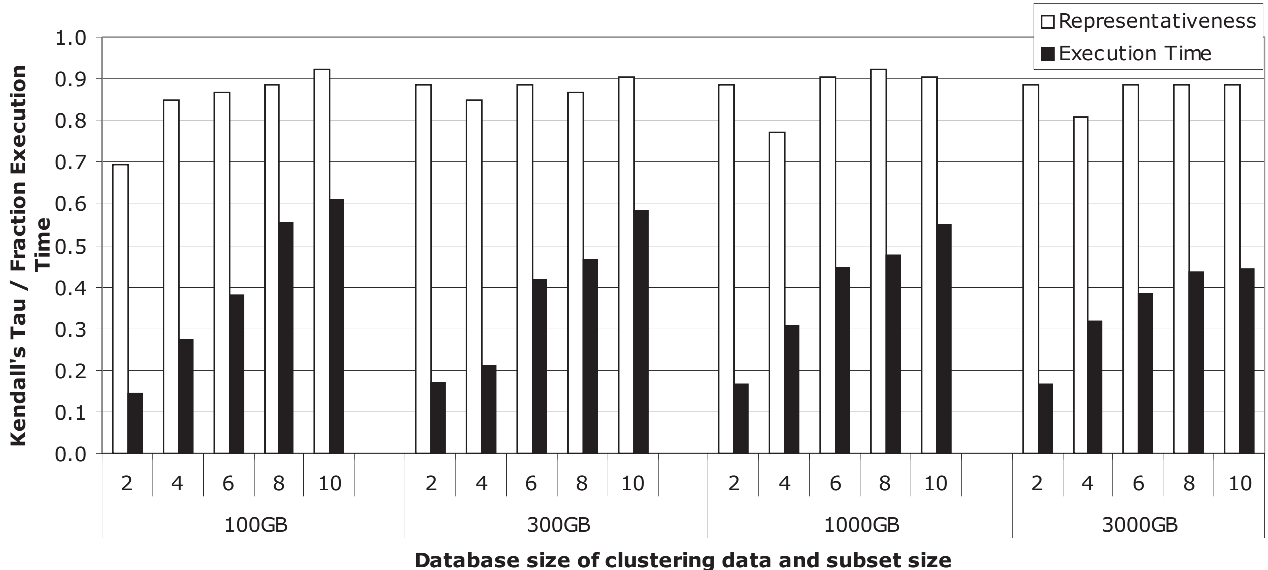

Figure 3. Representativeness and reduction in execution time for subsets computed from measurement data for different database sizes. The subsets are validated for the 300 GB database.

bars, the fraction of time needed to benchmark the system using a 4-query subset, relative to the time needed to run the full suite. Only 20–35 % of the execution time is needed to come up with an acceptable performance estimate.

Increasing the size of the subset improves the representativeness at the cost of a larger benchmarking time. Figure 2 shows how these properties vary with the subset size for the 100 GB database. Representativity starts at 73% for a 2-query subset and climbs to 90% for a 10-query subset. However, this results in an increase of the execution time from 14% to 62%. For the results in Figure 2, representativeness increases significantly from 4 to 6 queries while execution time rises from 6 to 8 queries. Thus, the 6-query subset provides the best trade-off between accuracy and performance.

5. Further Analysis

In this section we investigate different aspects of the subsetting procedure and their impact on representativeness.

5.1. Dependency of Subsets on DB Size

For the results shown so far, when creating a subset for a certain database size, we have used the performance measurements for that same database size. The database size may have an important impact on the similarity of clusters, *i.e.* some queries may exhibit more similar or less similar characteristics depending on the database size used in the analysis. However, the previous section showed only slight differences between cluster compositions for each database size. Therefore, it is unlikely that the database size has a large impact on the constructed subset.

To verify this observation, we evaluate how representative the clusters constructed for each database size are for the 300 GB workload (Figure 3). The subsets computed using the 300 GB database results have a representativeness fluctuating between 0.86 and 0.91. The subsets computed using different database sizes obtain the same level of representativeness except for a few exceptions. The 2-query cluster computed with the 100 GB results and the 4-query clusters computed with the 1000 GB and 3000 GB results are not as representative as the others: 0.70, 0.78 and 0.80, respectively. Overall, representativeness is within the same range.

These data also show an important short-coming of the K-means clustering procedure. While it is intuitive to expect that using more clusters implies better representativeness, this is seldomly true in practice. Consequently, it is hard, in general, to state in advance the optimal number of clusters. Therefore, it is always necessary to perform a proper confirmation test, like the one discussed here, to confirm that a particular number of clusters delivers the desired level of representativeness.

The important conclusion of these experiments is that subsets computed for one database size are applicable to other database sizes without loss of representativeness. Thus, any of these four database sizes can be used to compute and characterize the subsets of queries.

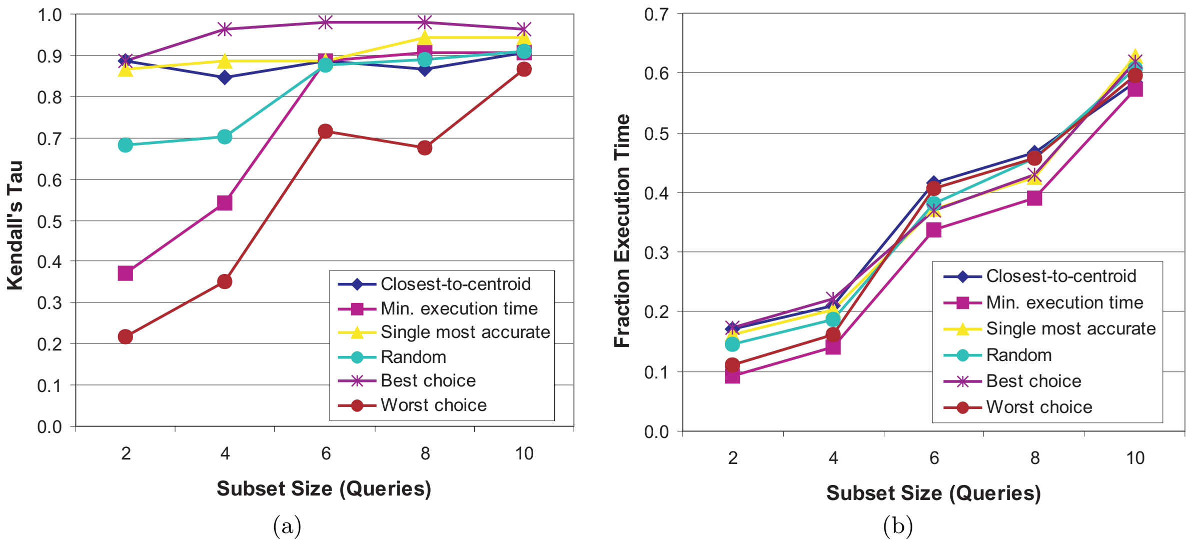

Figure 4. Impact of the choice of a cluster representative on the representativeness of the cluster (left) and the fraction of execution time remaining (right) for the 300 GB database results.

5.2. Choice of Representatives

As described before, in this work we use K-means clustering in order to separate the queries into groups of similar queries. After this grouping, a representative query has to be selected for each group or cluster. As mentioned in Section 3.3, there are several alternatives for the selection of this representative query. In this work we explored three different heuristics for the selection of the representative: *closest-to-centroid*, *min-execution-time*, and *single-most-accurate*. While the first one results from the clustering algorithm, the second one addresses the goal of reducing the total benchmarking time, and the third one addresses the goal of keeping the subset accurate. We also consider a random selection of a subset, in order to check the quality of the heuristics [16]. Figure 4-(a) shows Kendall's Tau metric and Figure 4-(b) shows the execution time when selecting representatives using the three heuristics mentioned above. The results are for the 300 GB database size. Different cluster sizes are shown along the horizontal axis.

The *closest-to-centroid* heuristic performs very well for small subsets, but falls a little short of the random heuristic for 8-query subsets.

Using the *min-execution-time* heuristic reduces execution time by 7–10% compared to the *closest-to-centroid* heuristic. However, it performs poorly for small subset sizes of 2 and 4 queries. As such, this heuristic must be used with care.

Overall, the *single-most-accurate* heuristic performs the best for all subset sizes. It is outperformed only by the *closest-to-centroid* on the 2-query subset. Furthermore, the execution times are usually 1–4% less than those of the *closest-to-centroid* heuristic.

In order to evaluate the quality of the selection heuristics, Figure 4 also shows the best and worst possible choices of the representatives in terms of maximizing representativeness (Kendall's Tau). These choices were found using exhaustive search over all possible subsets of the queries. The results show that the quality of the *single-most-accurate* representatives is within 9.5% of the best possible. In addition, this gap seems to decrease as the subset size increases. For example, for the 10-query subset size the gap is only 1.9%.

In addition to the error resulting from the selection of the representative for each cluster, another source of possible error may be the formation of the clusters themselves, *i.e.* what queries are clustered together. While it is difficult to determine the best combination of queries for each subset size, the quality of the clusters is determined by comparing the absolute best Kendall Tau for a selection of n queries compared with the best selection out of the determined clusters. For the 300GB data set results presented in Figure 4 and the 2- and 4-query subsets, the best for the corresponding clusters is the same as the absolute best (not shown). Therefore, it is possible to conclude that the clusters represent the data with high accuracy and are not an important source of error.

6. Relevance for Architectural Studies

So far, the subsetting methodology has been validated for analyzing the performance of existing computers. We have shown that the subsets can accurately

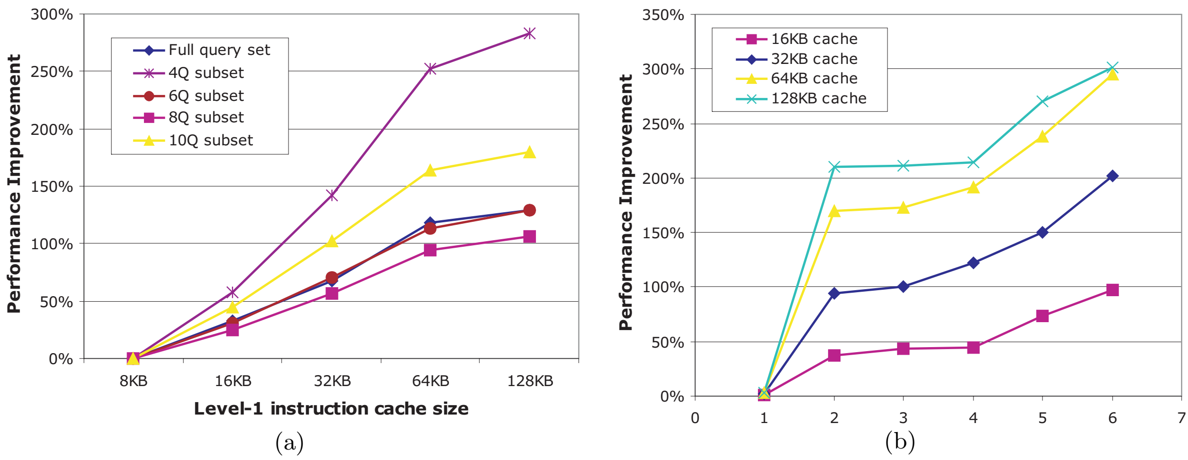

Figure 5. Measuring the impact on performance of increasing the first level instruction cache size for different subset sizes (left) and a performance improvement S-curve for the 6-query subset (right).

rank the computers from fastest to slowest. Another use of the subsets is when designing a new processor. The conclusions that one wants to draw in this application are different from before. In this scenario, a designer wants to dimension certain microarchitectural structures, determine parameters, determine whether certain features should be included or excluded from the design, etc. The subsets must now meet the requirement that they do not lead a designer towards a processor that is overly optimized towards the subset. Instead, it should achieve the desired performance, energy, and other goals for the full benchmark suite. In this section, we emulate this type of decision-making by means of case studies in dimensioning cache sizes. We show that the subsets lead to the same design decisions as the full benchmark suite.

We depart from a baseline processor modeled by Wattch [1]. The processor is a 4-issue out-of-order processor with a 64-entry register update unit and 32-entry load/store-queue. The branch predictor is a 4096-bit bimodal predictor. The first-level instruction and data cache are each 8 KB large and 2-way set-associative with a 32 byte block size. The second-level cache is 256 KB and 4-way set-associative. The TPC-H queries are executed on 100 MB databases to limit simulation time. The queries are executed using the PostgreSQL database program compiled for the PISA ISA.

The subsets are determined using the performance measurements for the 300 GB database. The representatives are determined using the closest-to-centroid heuristic.

6.1. Case Study 1: Instruction Cache

Our first case study is dimensioning the instruction cache: is it worthwhile to have an instruction cache larger than 8 KB? If so, how large should it be? The instruction cache size has a major impact on performance (Figure 5-(a)). Increasing the cache size rapidly increases performance up to 64 KB where it levels off. The 64 KB instruction cache improves performance by 118% on average for the full set of queries. Estimating the performance impact using different subsets show different performance estimates. The 4-query and 10-query subsets strongly overestimate the importance of instruction cache size, while the 8-query subset shows an underestimation. The 6-query subset however represents the full query set well as it accurately predicts the performance improvement of the instruction cache size. This effect occurs in all subsetting work [5, 6, 16]: it is practically impossible to predict before-hand what subset size will perform best. This is an open problem that needs to be addressed in future research.

Note that, even when performance is wrongly estimated, using a subset typically results in a correct design decision. Indeed, the knee of the curves are located at the same position (64 KB) for all subset sizes.

When determining the usefulness of a microarchitectural feature, dimensioning its parameters, architects may use so-called *S-curves*. The S-curve shows the performance improvement obtained by the feature for each of the benchmarks. The S-curve is constructed such that the performance improvements are ordered from smallest to largest. The S-curve obtained for dimensioning the first-level instruction cache are shown in Figure 5-(b) when using the 6-query subset. This data shows that there is one query for which the first-level instruction cache size has little impact (the query in question is Q18). Increasing the first-level instruction cache size is beneficial for the other 5 queries up to a 64 KB cache size. Beyond that, improvement is marginal. This shows again the validity of the choice

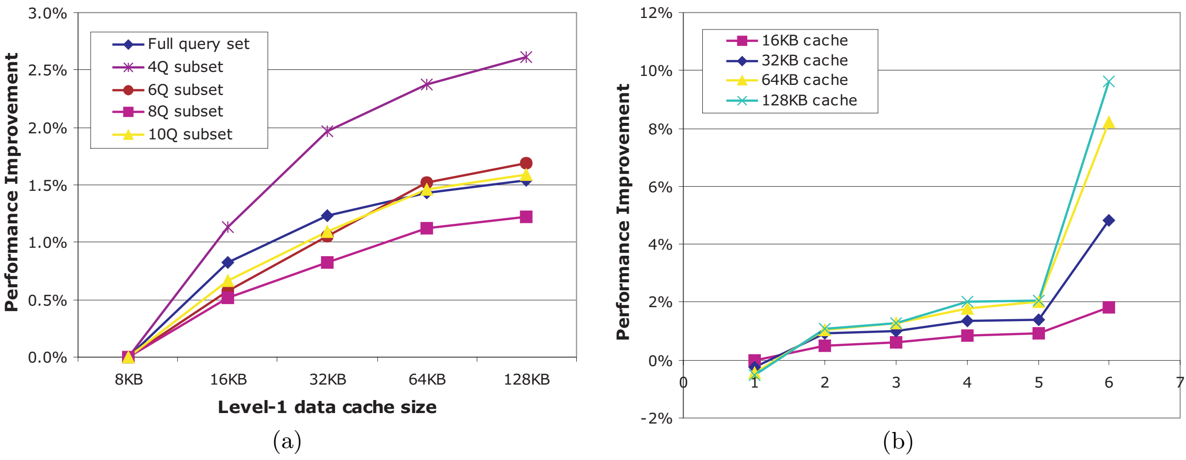

Figure 6. Measuring the impact on performance of increasing the first level data cache size for different subset sizes (left) and a performance improvement S-curve for the 6-query subset (right).

of a 64 KB instruction cache.

6.2. Case Study 2: Data Cache

Our second case study is dimensioning the data cache size. The first-level data cache size has less impact on performance than the instruction cache for the TPC-H queries. This makes a decision on the cache size much trickier and much more dependent on the actual subset that is used. However, design choices with low performance impact are frequently made by chip designers. Therefore, they are equally important.

The full query set indicates a good trade-off between cache size and performance at 32KB (Figure 6). Again, the 4-query subset strongly overestimates the performance impact, almost by 70%. However, the knee of the curve is at 32 KB, guiding a designer to the same cache size. The 6-query and 10-query subsets track the desired performance improvement rather closely. However, they slightly underestimate the usefulness of the 16 KB and 32 KB data caches.

The performance averages might lead to a design decision of a 64 KB cache, which appears too large in comparison to the data for the full query set. However, investigating the S-curves shows that the data cache size has little impact for 5 of the 6 queries. This behavior is determining for the overall average. However, there is one query (Q9) for which increasing the data cache is significant and a good trade-off for its cache is 64 KB. Thus, the conclusion is that data cache size has a small performance impact for 5 of the 6 queries and for these, a 32 KB cache is sufficient. However, for 1 of the 6 queries, increasing the data cache size to 64 KB has a significant performance improvement. Based on this observation, as well as the design goals of the particular project, one can opt for either of the cache sizes.

Note that this case study uses a very small number of benchmarks. In practice, many more benchmarks covering different types of workloads would be included in the graph, making the trade-off not as crude as the larger cache is useful for only one benchmark. Instead, the larger cache would be beneficial for several more benchmarks and there would also be benchmarks for which the benefit is not as pronounced.

Notice also that the fraction of queries with exceptional behavior corresponds to their fraction in the full set of queries. In the 6-query subset, one of the queries has a very strong dependency on the data cache size while there were two (Q9 and Q19) in the full set. There is also one query with almost no dependency on the instruction cache size, while the full set features four like that (Q2, Q18, Q20 and Q22). Thus, the subset includes the most irregular behaviors. Including such queries in the subset is crucial to understand the limit cases of a design.

6.3. Summary

In this paper, we have reduced the 22 TPC-H queries to a subset of only 6 queries, yielding an almost 60% reduction in execution time. Using only those 6 queries to scale processor resources results in the same trade-offs as the full set of queries, both in the case where a resource has a major performance impact (instruction cache) as in the case where the performance impact is small (data cache). This shows the validity of the proposed subset.

7. Conclusion

Benchmarking a computer system is time-consuming and complex to perform. The bench-

marking effort can be decreased by reducing the number of benchmarks in a benchmark suite. However, care must be taken that the reduced benchmark suite remains representative to the original suite.

In our method, two benchmarks are similar with respect to performance properties when the performance ranking of a set of computers obtained by running one benchmark is similar to the ranking of the same computers using the other benchmark. Thus, our method takes into account only differences between benchmarks that are relevant to performance.

We apply benchmark subsetting to the TPC-H, a decision support system database benchmark. The representativeness of the subsets is evaluated for various database sizes and we show that the similarity between TPC-H queries remains largely independent of the database size.

Improvements to the subsetting procedure are proposed and evaluated. In particular, we propose new heuristics for the selection of cluster representatives and show how they improve the subsets. Using these heuristics, we propose a reduction of the TPC-H suite from 22 to 6 queries, yielding a 60% reduction of execution time. Finally, we show by means of cases studies that the subsets are appropriate to design new processors. Basing quantitative decisions on the TPC-H subsets leads to the same design as when making the decisions based on the full TPC-H suite.

Acknowledgements

We are grateful to Christodoulos Adamou and the rest of the CASPER group for their help. Hans Vandierendonck is a post-doctoral researcher of the Fund for Scientific Research-Flanders (FWO-Vlaanderen).

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, jun 2000.

[2] W. J. Conover. *Practical Non-Parametric Statistics*. John Wiley & Sons, 1999.

[3] T. P. P. Council. TPC Benchmark^{TM} H (Decision Support), Standard Specification, jun 1999.

[4] T. P. P. Council. TPC Benchmark^{TM} C, Standard Specification, apr 2004.

[5] L. Eeckhout, R. Sundareswara, J. J. Yi, D. J. Lilja, and P. Schrater. Accurate statistical approaches for generating representative workload compositions. In *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, pages 55–66, Oct. 2005.

[6] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Designing computer architecture research workloads. *IEEE Computer*, 36(2):65–71, Feb. 2003.

[7] J. L. Henning. SPEC CPU2000: Measuring CPU performance in the new millenium. *IEEE Computer*, 34(7):28–35, July 2000.

[8] A. J. KleinOsowski and D. A. Lilja. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters*, 2002.

[9] M. Levy. EEMBC scores 1.0, part 1: Observations. *Microprocessor Report*, pages 1–7, aug 2000.

[10] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, 2000.

[11] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John. Measuring program similarity: Experiments with spec cpu benchmark suites. In *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2005)*, pages 10–20, mar 2005.

[12] M. Shao, A. Ailamaki, and B. Falsafi. Dbmbench: Fast and accurate database workload representation on modern microarchitecture. In *Conference of the IBM Centre for Advanced Studies on Collaborative Research*, 2005.

[13] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[14] P. Trancoso, C. Adamou, and H. Vandierendonck. Reducing TPC-H benchmarking time. In *10th Panhellenic Conference on Informatics (PCI 2005)*, nov 2005.

[15] H. Vandierendonck and K. De Bosschere. Eccentric and fragile benchmarks. In *2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 2–11, Mar. 2004.

[16] H. Vandierendonck and K. De Bosschere. Experiments with subsetting benchmark suites. In *Proceedings of the Seventh Annual IEEE International Workshop on Workload Characterization*, pages 55–62, Oct. 2004.

[17] T. Wunderlich and B. Falsafi. SMARTS: accelerating microarchitecture simulation via rigorous statistical samplin. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 84–97, jun 2003.

[18] J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins. Characterizing and comparing prevailing simulation methodologies. In *Proceedings of the 11th Conference on High-Performance Computer Architecture*, pages 266–277, feb 2005.