An Industry Standard Benchmark Consortium

Characterization of the EEMBC Benchmark Suite

By Jason Poovey, North Carolina State University

EEMBC addresses the needs of embedded designers by providing a diverse suite of processor benchmarks organized into categories that span numerous real-world applications. Research done at North Carolina State University investigates the benchmark suites through the use of benchmark characterization to create a description of each workload.

Benchmark characterization involves re-describing a workload as a set of quantifiable abstract attributes. Designers can then use these measured attributes to determine program similarity. By combining these characteristics with their knowledge of the actual application, designers can select the most relevant benchmarks from the EEMBC suite to test the potential *in situ* performance of an embedded processor.

We used a mixture of design characteristics including instructions/clock (IPC), branch mis-prediction ratios, and dynamic instruction percentages. We also collected hardware design metrics for caches and functional units that target the hardware requirements to achieve certain performance goals.

Methodology

This study collected characteristics for the MIPS, PowerPC, x86, and PISA (used in Simplescalar) architectures. The resulting combination of metrics provides an accurate representation of the workload's activity. Thus, designers can find which workloads are similar to their own and then leverage the most relevant benchmarks as a proxy for architecture design. The characteristics assist in this process by indicating the minimum hardware needed to achieve a target level of performance.

We relied primarily on trace-driven simulation to characterize the processors listed above. We used trace-driven simulation to collect data for cache design experiments as well as to reveal the distribution of dynamic instructions.

Architecture Independent	Architecture Dependent
▪ Cache Design for target miss ratios ▪ 1% and 0.1% Target Miss Ratios ▪ Block sizes ranging from 2^4 to 2^7 ▪ Set Associativities: Direct Mapped, 2-way, 4-way, and fully associative ▪ Functional Unit Requirements Distribution ▪ Target 85% Utilization ▪ ALU, MULTDIV, LSU, BRANCH, SHIFT ▪ Dynamic Instruction Mixes	▪ Instructions per Cycle ▪ Branch Mis-prediction Ratios ▪ Bimodal Predictor

Table 1 – Measured Benchmark Characteristics

Cache Design

Block size, set associativity, and total size are the defining characteristics of caches. Cache performance may vary significantly if any of these variables are changed. Therefore, it is often very costly to perform an entire design space search because each cache configuration of interest requires additional simulation. We used a single-pass simulator to allow for the simultaneous evaluation of caches within a specified range of these three variables. Rather than picking particular cache configurations, this tool isolates cache configurations that meet user-specified performance goals. The single-pass simulator characterizes workloads in terms of hardware requirements, rather than simply measuring the performance of a particular cache realization. We chose miss ratios of 1% and 0.1% as target performance goals to demonstrate the suggested cache sizes that would achieve the desired performance for L1 and L2 caches, respectively. If cold misses cause a cache configuration to have a higher miss rate than the user specified target, then the intrinsic miss ratio is targeted

Functional Unit Distribution

When designing a new system, designers must decide the number of functional units (such as load/store units and multiply/divide units) that should be included in their design. Rather than iteratively modifying the number of functional unit types and re-simulating, the distribution measurements indicate the number of functional units needed for each type. The functional unit distribution simulates an idealized out-of-order machine with an infinite width and a perfect branch predictor. True dependencies between instructions thus become the only bottleneck. We then collected data to determine the number of functional units requested at any given time. In this study, the distribution results represent the number and type of functional units necessary to meet workload demands for 85% of the execution time. The functional units simulated were ALUs, load/store units, multiply/divide units, branch units, and shift units.

Experimental Analysis

The functional unit distributions show the hardware needed to achieve maximum parallelism on an idealized machine. To graphically represent the metrics of our simulations, we used Kiviat graphs, which visualize multivariable data in a way that easily reveals program behavior. As Figure 1 demonstrates, the results vary greatly between the benchmark suites. This is significant for two reasons: it points out the application-specific nature of each benchmark suite, and it shows that more than one suite must be run to comprehend the capabilities of the processing platform.

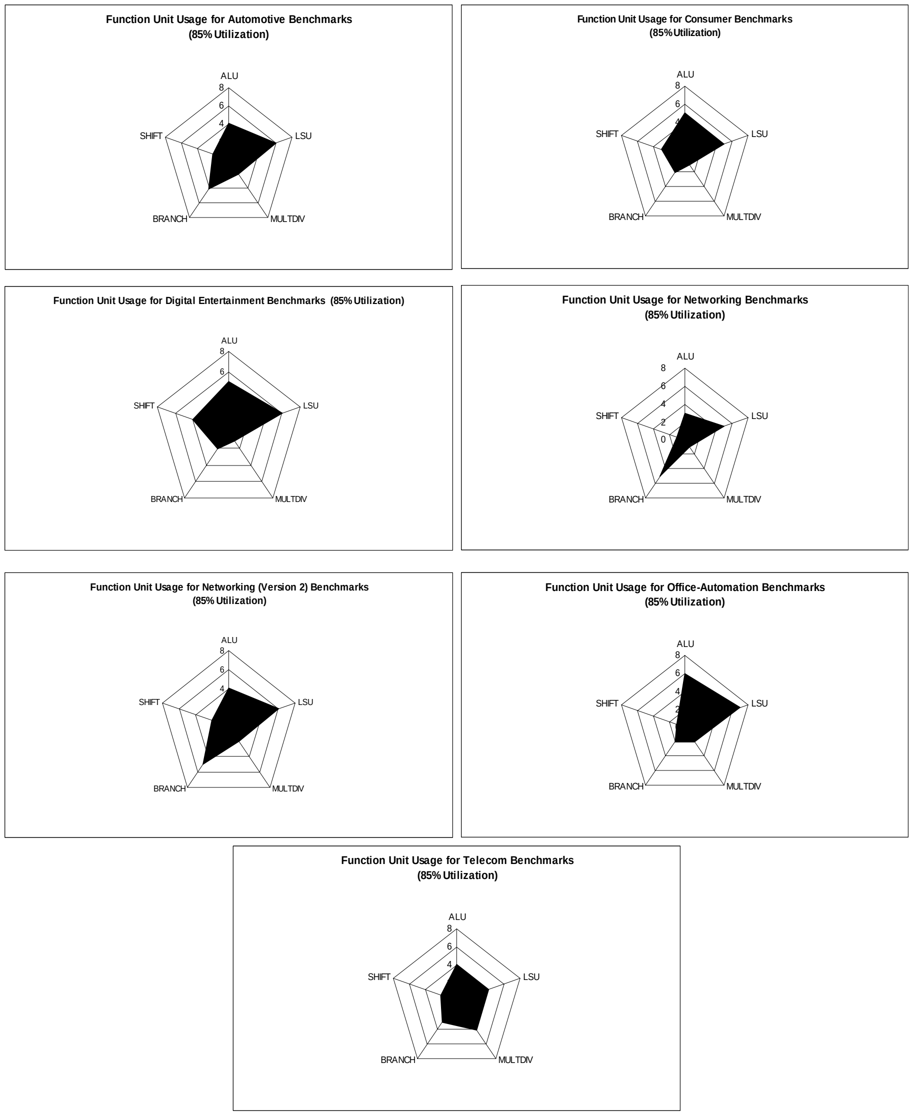

Figure 1 – Using Kiviat Graphs to Represent Functional Unit Distribution (85% Utilization)

EEMBC’s Networking 2.0 suite has larger functional unit requirements than Networking 1.1, indicating that greater parallelism is available in the latest version. TheAutoBench 1.1 automotive/industrial suite exhibits similar strains on functional units as Networking 2.0, with the difference being that Networking 2.0 has slightly higher requirements for branch instructions. In general, Networking had the largest percentage of branch instructions, and thus required the largest number of corresponding functional units. Higher numbers of load/store functional units are beneficial in all suites except for TeleBench 1.1. This is the case not because of a lack of memory instructions, but because as the instructions distribution indicates, TeleBench 1.1 contains an average percentage of memory instructions for all benchmarks. These memory instructions do not exhibit high parallelism, and thus do not benefit from additional load/store functional units.

In our analysis, we determined that the DENBench 1.0 digital entertainment benchmarks place the most demand on shift functional unit requirements, where four execution units are needed to optimize performance. On the other hand, the OABench 1.1 office automation suite exhibits a unique behavior, where only the ALU and LSU units are stressed.

Benchmark-Specific Analysis

The analysis above shows that there is workload variety between the different benchmark suites. Further analysis shows that there is also significant variety internal to the particular workload categories. For example, within AutoBench 1.0, *aifft* had very large cache requirements, whereas *iirflt* did not. *Pntrch* showed a high percentage of memory accesses, but did not require a significant cache size to obtain the desired performance goals.

Branch predictor accuracy is high for most benchmarks. However, some workloads, such as *aifftr* and *aiifft*, show spikes in the mis-prediction rates. However, this is similar to many embedded environments, where code is optimized for the absence of a branch predictor, which is omitted to save space or power.

Similar workloads have similar shapes in the Kiviat graphs. Figure 2 shows a collection of Kiviat graphs grouped based on workload similarity. In this figure, nine clusters were determined via visual inspection.

No single suite exhibits homogeneous characteristics. Even the ConsumerBench 1.1 digital imaging suite, which targets the most specific applications, spans two classification categories. The most heterogeneous is AutoBench 1.1, which covers four of nine Kiviat classifications. OABench 1.1 is interesting in that it contains only three benchmarks, each of which were classified into different categories. Also of note is that workloads from differing suites exhibit similar characteristics. For example, *canrdr* is very similar to many networking applications. This means that workload activity is similar even between different suites, with only minor differences in the magnitude of the specific metrics.

In the Networking suites, the *pktflow* and *pktcheck* benchmarks are implemented using four different packet sizes. As the size of the packets increase, the cache activity also slowly increases. Within the networking suites, there are differences between the Networking 1.0 and Networking 2.0 *ospf* benchmarks. The Networking 2.0 version has greater ALU activity. Finally, the *rgbcmy* and *rgbyiq* benchmarks require much larger caches to achieve a 0.1% miss ratio versus a 1% miss ratio, showing that these benchmarks have many conflict misses that require a larger cache size to remove.

An Industry Standard Benchmark Consortium

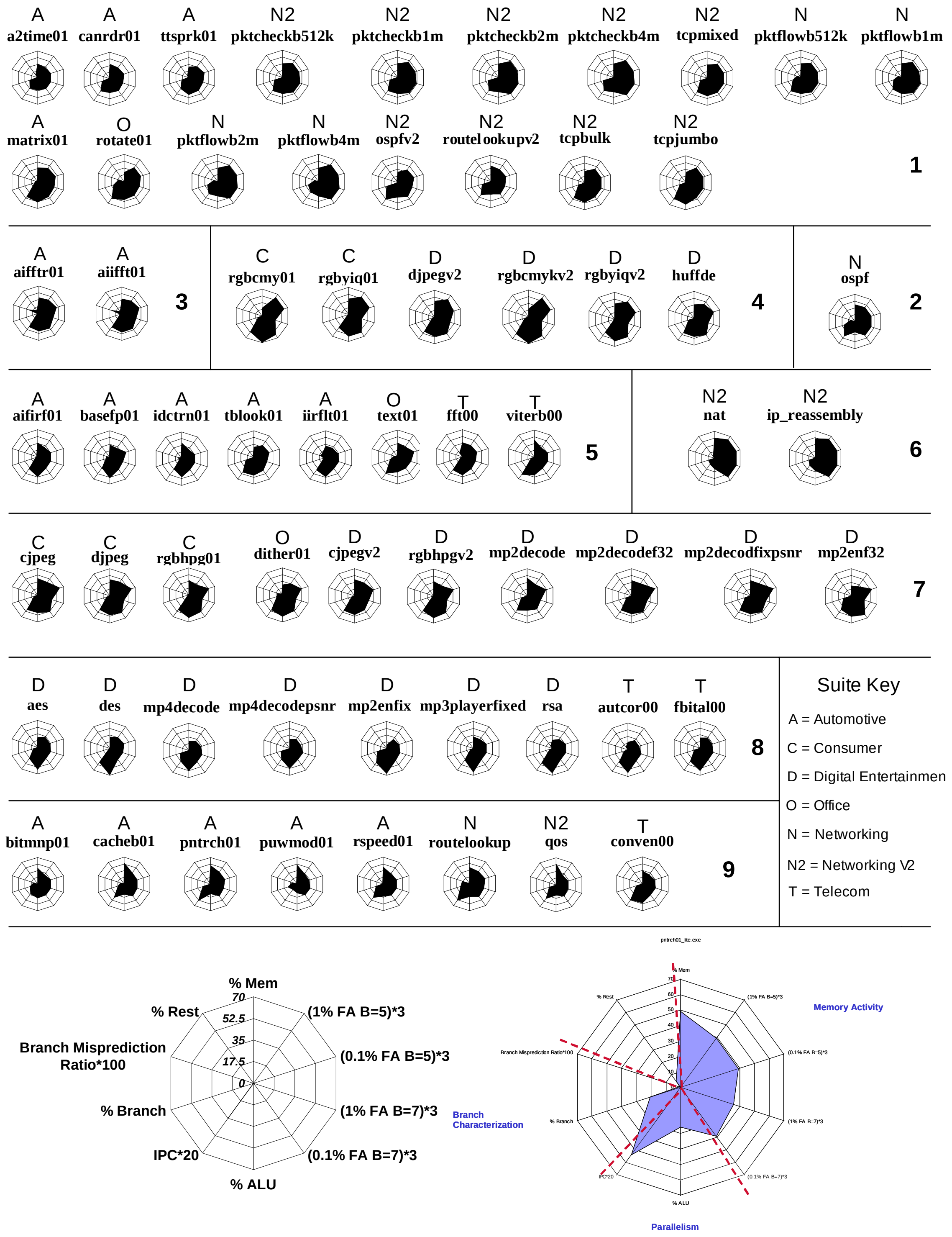

Figure 2 - Kiviat Plots of Combined Characteristics

Conclusion

This experiment shows the diversity of the EEMBC benchmark suite as well as providing insight into the specifics of each workload's activity. By using a set of hardware design and performance metrics, the results display an accurate representation of the workload's inherent behavior. As expected, we found diversity within and between the suites. This diversity ensures that designers can use combinations of EEMBC workloads to represent most real-world workloads and use this characterization data as a starting point to make effective design choices.