Abstract

Branch prediction is of great importance to deep-pipeline design. Speculative execution is necessary to more fully utilize chip resources, and the need for issuing correct speculative instructions leads to demand for very accurate predictors. Single predictors have been successful in predicting branches with high accuracy for many workloads, though the increasing number of instructions in-flight demands increasingly more accurate predictors. Hybrid predictors show promise of fulfilling this demand, combining the capabilities of many individual predictors at the cost of increased predictor size.

Phalanx is an implementation of a quickly reconfigurable hybrid predictor, intended to evaluate the effectiveness of many different predictor combinations. We have surveyed the effectiveness of several combinations of popular predictors against the SPEC2000 integer benchmark suite. The total size of memory size allocated to branch prediction is fixed at 8 KB. Phalanx divides this space among sub-predictors. Each configuration has been compared against its constituent predictors—each of which has been allocated memory equal to the combined size of all predictors in Phalanx.

1. Introduction

In modern processors, pipelines are growing deeper as design complexity increases. There are more instructions in flight due to high instruction issue rate, and it is becoming increasingly difficult to ensure that these instructions will reach the commit stage. One example of such processors is the Intel® Pentium 4, in which instructions can span as many as 20 pipeline stages. As a result, to avoid pipeline stalling for branch resolution, the amount of speculative work in the pipe becomes very large. Therefore, employing a high-performance branch predictor is critical to avoid wasting processor effort due to incorrect speculation.

Currently, high-performance branch predictors can achieve around 97% accuracy for most workloads. The remaining 3% represents the misprediction rate, and can nonetheless result in substantial loss in performance due to the large number of instructions issued per cycle and the number of cycles these instructions are in the pipeline before an incorrect branch prediction is

resolved. In other words, branch prediction becomes a bottleneck of CPU performance, especially in deep-pipelined processors.

The purpose of this project is to build and simulate several configurations of a comprehensive branch predictor, called Phalanx. This branch predictor is composed of several sub-predictors as well as a meta-predictor to select among these sub-predictors and make the final prediction of whether a conditional branch should be predicted taken or not taken. Phalanx is intended to be used to evaluate many configurations, and does not tailor its multi-predictor architecture to any specific number of composition of sub-predictors.

Through using many predictors, we hope to improve the prediction miss rate and therefore overall CPU performance. It has been shown that using two predictors [6,8,10] can reduce the miss rate associated with either constituent predictor evaluated independently to some extent. We will explore the more general case of using multiple (more than two) predictors in parallel.

2. Related Work

In general, Phalanx uses several combinations of the two-level predictors described by Yeh and Patt [1,2,3] and the bimodal predictor as in [5,6]. The two-level predictors use some number of branch history registers (BHR) indexing into pattern tables implementing two-bit finite-state machines. Specifically, Phalanx includes from the two-level family the global-address, global table (GAg) scheme, the per-address history and global pattern scheme (PAg), the per-address history and per-address pattern scheme (PAp), and gshare (GAg as outlined above, but using the least significant bits of the program counter under the xor operator with the BHR to provide the index).

The bimodal predictor is not a member of the same two-level family. This scheme also implements a table of two-bit saturating counters to facilitate predictions on a per-address basis, but this table is indexed directly by the least significant bits of the program counter [5]. Different branch prediction schemes have different advantages. The notion of hybrid prediction has been explored in literature [6,7,8,9,10]. One hybrid predictor proposed by McFarling[6] uses a

combination of the bimodal and gshare predictors. The combination of the gshare and bimodal predictors performs well using McFarling's meta-prediction method. The result of McFarling's experiment shows that this combination always achieves a higher prediction rate compared to either scheme alone, under the condition that the size of the hybrid predictor is three times as large as the size of the single predictors. McFarling also experimented with different divisions of resources among constituent predictors, giving the gshare predictor a greater portion of the available storage space. This combination also achieves higher accuracy than the single predictors, under the same assumptions. McFarling's work shows that multiple branch predictors can be combined to achieve higher accuracy by keeping track of which predictor is more accurate for each individual branch. This scheme can reach around 98% accuracy for most workloads. [6]

Other combinations of two-level adaptive branch prediction schemes were explored by Chang et al.[8] They used two-bit saturating counter predictor, Per-address address history and Per-set pattern (PAs), gshare and static predictor. PAs is a variation that combines the features of PAg and PAp. The combinations include: gshare and two-bit saturating counter, gshare and static prediction, gshare and PAs, PAs and two-bit saturating counter, PAs and static, as well as gshare alone as a control. The result shows that the combination of gshare and PAs performs the best among other combinations. However, since PAp is proven to perform better than PAs [3], we have elected to include PAp in Phalanx instead of PAs.

Additionally, some work has been done by Patil and Emer [10] in combining inexpensive stateless branch predictors with stateful predictors. The focus of this work was on the effect of aliasing that arises in many common stateful prediction schemes. Static predictors are by definition immune to this effect.

3. Phalanx

The Phalanx project was intended to explore the idea of combining multiple (more than two) branch predictors to form a single, large hybrid predictor design. Hybrid predictors employing two sub-predictors have been very successful [6], although the exploration of multi-predictor hybrids has not enjoyed the same degree of success. There are many challenging and

performance-affecting considerations when designing and implementing multi-predictor hybrids; chief among these are resource division, predictor overlap, and meta-prediction (the heuristics by which a prediction is selected from a pool of predictors). Phalanx explores the effects of overlap and prediction strategies, using combinations of stateful, two-level adaptive branch prediction strategies (GAg, PAg, PAp, gshare) [1,6] and the bimodal predictor [5,6]. Phalanx attempts to divide resources (approximated by the number of bits used in storage elements) as equally as possible among constituent predictors, subject to total available space, and rounding each predictor's size to appropriate powers of two, for addressability.

Each Phalanx configuration is a hybrid branch predictor consisting of a combination of commonly used predictors operating in parallel to generate predictions. Since Phalanx is intended to evaluate many different configurations (inclusions of sub-predictors) rapidly, Phalanx uses meta-prediction schemes that can be applied to an arbitrary number of constituent branch predictors. This approach differs from that of many other hybrid implementations in that the meta-prediction scheme is not tailored to each individual inclusion configuration, but allows rapid changes in predictor arrangement with very little modification to the meta-prediction mechanism. Specifically, Phalanx employs two meta-predictors that vary significantly in philosophy and implementation.

The first of these meta-predictors uses a weighted voting method to select a predicted path at each branch. Under this scheme, every predictor included in Phalanx is assigned a numeric confidence. The confidence is a two-bit saturating counter, incremented whenever a constituent predictor predicts a branch correctly, and decremented on incorrect predictions. When predictions are requested, each sub-predictor is granted a voting weight according to the predictor's current numeric confidence. In effect, the resulting prediction—branch taken or not taken (T/NT)—depends solely on which of T/NT receives the greatest combined voting weight from all constituent predictors.

It is possible under the Phalanx – Majority Vote scheme (described above) for ties to occur. We have opted to always break ties in favor of a taken prediction, although this is another policy decision to consider. Other heuristics may be of use here, including backward-T/forward-

NT, or to simply favor a particular predictor over another in the event of a tie. One of the advantages of the Phalanx – Majority Vote scheme is that it allows predictor confidence to change quickly. The size of the confidence values allows confidences to quickly adapt in response to changes in predictor success rates, given a weighting method that uses confidence-based voting (i.e. voting weights vary with confidence). This advantage is shared by the other meta-prediction scheme implemented in Phalanx.

The Phalanx – History meta-predictor uses 16-bit shift registers to record the correctness of the last 16 predictions for each of the constituent predictors. Based on these histories, the currently best-performing predictor is selected to provide all branch predictions. The predictions of all other sub-predictors are ignored for the purpose of deciding the overall prediction determined by Phalanx—though predictions are nonetheless checked for correctness when the branch is resolved. The history registers are updated whenever a branch instruction is resolved, and its final address is known.

Phalanx – History defines the "best performing predictor" as the predictor that has the longest run of 1's in its history register, starting with the least-significant bit. This equates to the predictor that has been predicting most consistently correctly in recent execution (i.e. the last 16 branches). As in the case of the Phalanx – Majority Vote meta-predictor, ties are possible. Tie resolution is decided by population count—for two or more predictors with the same length run of correct predictions in the LSB positions in respective history registers, the "best performing predictor" is defined as the predictor with the greatest number of 1's in its history register. In the event of ties in population count as well as longest run, the predictor is chosen arbitrarily (for the simulation, we select the first predictor defined in the configuration file).

The fundamental difference between the Phalanx – Majority Vote and Phalanx – History meta-predictors is that Phalanx – Majority Vote uses the predictions of all constituent predictors to determine the prediction for a given branch, whereas Phalanx – History relies only on the prediction of a single predictor that has performed well in the recent past. Phalanx – Majority Vote employs all available resources for each prediction and quantifies predictor agreement. Phalanx

– History uses only the best-performing resources for each prediction, sacrificing predictor agreement.

The earliest versions of the Phalanx predictors employed both stateful and stateless predictors, and simple meta-prediction schemes selected among the predictors. It quickly became evident that while stateless predictors are simple (and inexpensive, in terms of on-chip area requirements) they cannot individually produce consistently accurate predictions. The most accurate of the stateless predictors examined was much less accurate than even the most rudimentary of the stateful predictors considered for inclusion in the Phalanx project. Moreover, the correct predictions of some stateless predictors approximate subsets of the correct predictions of stateful predictors—including such predictors would not likely improve correct prediction rates. Because of these two observations, Phalanx includes only stateful predictors.

4. Implementation Details

The Phalanx predictor was implemented for simulation purposes using the SimpleScalar 3.0d branch prediction simulator, `sim-bpred`. The SimpleScalar suite offers a variety of simulation tools, including an out-of-order execution simulator, on which we originally intended to evaluate the Phalanx predictor. The validity of the branch prediction statistics produced by the out-of-order simulation engine (`sim-outorder`) fell into question during initial development of Phalanx, and work on the out-of-order simulator was abandoned. As a caveat against future work using the SimpleScalar 3.0d out-of-order engine for branch prediction studies, the implementation of the branch prediction interface is not complete.

Specifically, special-purpose values and incomplete parameter definitions prevent the out-of-order engine from allowing predictions of arbitrary addresses. Predictors not employing branch target buffers (BTBs) are artificially handicapped by incomplete or nonexistent definitions of branch targets. Stateful predictors with BTBs are also handicapped by this effect, but only until the BTBs have been populated with meaningful values after the warm-up period. These values are supplied by the update stage of the branch prediction system in SimpleScalar, which produces correct results. Since stateless predictors take no action during the update phase, the

correct data passed to the branch prediction subroutines only benefits stateful predictors. Unfortunately, a further effect on stateful predictors is that warm-up periods are artificially extended by the incorrect information provided by the out-of-order engine—the predictors are unable to produce meaningful address predictions for most branches in the early stages of execution. The branch prediction simulator (`sim-bpred`) in SimpleScalar 3.0d does not exhibit the same incorrect behavior as the out-of-order engine, although it produces only branch prediction statistics (it produces no performance statistics, i.e. instructions per cycle).

The net effect of `sim-outorder`'s shortcomings is that all stateless predictors behave identically to the static predict-not-taken predictor. Less noticeably, stateful predictors experience longer warm-up times as compared to those exhibited by the same predictors and benchmarks under `sim-bpred`.

Additional effort was spent to allow the Phalanx implementation's constituent predictors to share a single branch target buffer logically within SimpleScalar, although this led to significantly longer simulator execution times—the final implementation allows each predictor use of an exclusive BTB (of uniform size across all predictors) and simplifies the overall system. In a true implementation, these target buffers could be combined into a single buffer, since the contents each of the buffers instantiated in the simulation engine remains identical throughout execution, as each sub-predictor is updated with the same branch targets as every other predictor.

5. Simulation and Results

5.1 Simulation Methodology

Phalanx was simulated using the SPEC2000 integer benchmark suite for the Alpha instruction set. This SimpleScalar (`sim-bpred`) simulator's strength is that it produces branch prediction rates quickly when compared to the number of dynamic instructions executed. Using a set of scripts to standardize simulation across four meta-prediction schemes and ten different configurations (inclusion) of Phalanx, we simulated for fifty million instructions on each of our benchmarks: gzip, vpr, gcc, mcf, crafty, eon, vortex, bzip2, timberwolf. The principle behind our

simulation methodology was to distill a set of consistent, analyzable data, from which we could draw insightful conclusions.

To focus on the effect of different predictor schemes, we fixed several variables across these simulations. The most important of these was the 8kbit size of the predictor. This predictor size reflects sizes that could be used in future processors with deep pipelines. This size applies both to the Phalanx hybrid predictor and the single predictors it was measured against. That is, a Phalanx configuration using four sub-predictors will have four predictors of roughly 2kbit size, but single predictors (used for comparison) will be sized to 8kbit. Furthermore, for history-based meta-prediction, we fixed the history register length at 16, and voting-based meta-prediction fixes confidence size at two bits per predictor.

5.2 Inclusion Schemes / Naming Conventions

For our simulations, we devised a 5-bit nomenclature for our inclusion schemes. In this scheme, each simulation is associated with an inclusion scheme by a bit vector. If we include the sub-predictor in Phalanx, we set the corresponding bit in the bit vector. Binary 1 denotes included, 0 denotes excluded. From left to right, the bits denote the following predictors: GAg, PAg, PAp, gshare, bimod. For example, the Phalanx configuration `10110` uses sub-predictors GAg, PAp, and gshare.

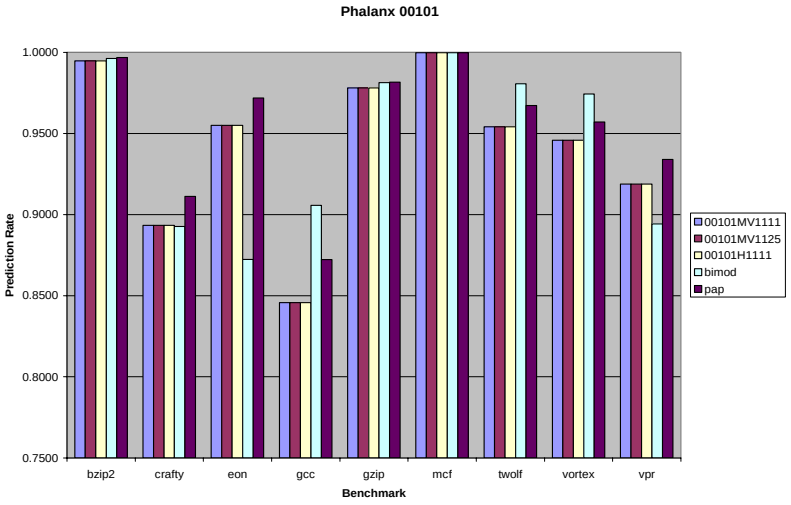
We did not simulate all 32 possible combinations of inclusions schemes. Based on our initial simulations, we were able to draw some conclusions about which inclusion schemes would be redundant. We believed that simulating all 32 schemes would yield a great deal of redundant information. The inclusions we simulated were selected such that most configurations represent hybrids that include both two-level and bimodal predictors, and the majority of configurations include more than two sub-predictors. We opted not to include most of the three-predictor combinations that include the GAg predictor, as initial benchmarking indicated that this predictor hinders performance when included in Phalanx. For comparison, we also opted to include the all-inclusive predictor (`11111`), and the all-inclusive two-level predictor (`11110`).

1. `00011` (gshare + bimod)
2. `00101` (pap + bimod)
3. `00111` (pap + gshare + bimod)

4. `01001` (pag + bimod)
5. `01011` (pag + gshare + bimod)
6. `10001` (gag + bimod)
7. `10011` (gag + gshare + bimod)
8. `11100` (gag + pag + pap)
9. `11110` (gag + pag + pap + gshare)
10. `11111` (gag + pag + pap + gshare + bimod)

The naming conventions used in the following sections represent the overall Phalanx configuration used to produce the given results. These names consist of the bit vector representation of the inclusion scheme (or the names of single predictors, if applicable), followed by either MV (denoting use of the Phalanx – Majority Vote meta-predictor) or H (denoting use of the Phalanx – History meta-predictor), followed by four hexadecimal digits. These digits denote the voting weights of increasing confidences used by the Phalanx – Majority Vote meta-predictor. These digits are meaningless for results of the Phalanx – History meta-predictor.

5.3 Simulation Results



When assimilating data for our analysis, we compared Phalanx's performance to the performance of its constituent predictors of the same total size as the hybrid predictor. (A complete set of charts and tables for our results can be found in Appendices A and B.) For example, as shown in the chart, when evaluating the Phalanx `00101` configuration, we compared the performance of the hybrid to the performance achieved by its sub-predictors, bimodal and PAp, when each is run as a single predictor. The first observable result is that the Phalanx inclusion scheme had a dramatic effect on prediction rate. Some Phalanx configurations we achieved strong performance, yet others had extremely poor performance compared to the single predictors. The configurations that produced the most- and least-accurate predictions are discussed below.

Another interesting result is that no Phalanx inclusion scheme was ever able to significantly outperform the best predictor of comparable size. Meta-prediction was never able to combine the predictions of the sub-predictors to yield an improvement over a single predictor of the same size.

Furthermore, we compared the results from the Phalanx – Majority Vote and Phalanx – History meta-predictors, given a specific configuration of Phalanx. Phalanx used four different meta-prediction schemes (three of which are based on the Phalanx – Majority Vote meta-predictor). The performance of the meta-predictors is not clearly distinguishable in most cases, gives us evidence that meta-prediction schemes are less important than other variables in combined branch predictors.

Lastly, we compared results of simulations for 5M instructions versus 50M instructions. The purpose of this comparison was to determine how our predictors warm-up rate varied against those of the constituent predictors. The notable result here is that as the number of instructions increases, the larger constituent predictors degrade in performance more significantly than that of the Phalanx predictors for the inclusion schemes shown. While Phalanx's performance also degrades, it does not degrade to as great an extent as the single predictors. This result will be examined in more detail below.

6 Conclusions

Although the performance of the Phalanx predictors was not as high as we anticipated, we were able to draw many conclusions based on our results. Here we present our conclusions on meta-prediction and inclusions schemes.

6.1 Combinations of Predictors

The best-performing configuration observed from our experiments is `00101` (PAp + bimod). Although this configuration never outperforms the best single predictor of the same total size for a given benchmark, it consistently performs well. Not surprisingly, this is a combination similar to McFarling's combined predictor. The worst-performing configuration observed from this experiment is `11111` (PAp + PAg + GAg + gshare + bimod), the all-inclusive predictor. This configuration consistently performs poorly, often the worst across a given benchmark.

Firstly, we consider why `11111` is the worst predictor. It seems counter-intuitive that the predictor with the most constituents performs poorly. However, there is a logical explanation for this phenomenon. Since Phalanx limits total size to a fixed number of bits, each additional constituent predictor that is added to Phalanx causes each other sub-predictor to sacrifice storage space to accommodate the new sub-predictor. This, in effect, makes every other sub-predictor less intelligent. Furthermore, overlapping predictors perform redundant work, which detracts from the usefulness of each predictor. Including all four of the two-level predictors has the dual effect of small predictor storage sizes combined with redundant information in the various predictors.

In general, the performance of a Phalanx predictor performance can be predicted, considering two broad observations:

1. *Fewer, smarter predictors are better than many, less intelligent predictors*
2. *Minimize redundant predictions—Increase predictor diversity*

The first observation is that Phalanx inclusion schemes with fewer predictors generally outperform those that include a greater number of predictors. Essentially, the greater the number of sub-predictors, the less information a given predictor may use to predict branches. This observation is analogous to a scenario where a few experts compete against many novices in a contest. The experts not only have are more well-informed than the novices, but they will reach consensus more quickly. The same is true with predictors in Phalanx. Most inclusion schemes with two sub-predictors outperform those with three sub-predictors. The exceptions to this rule

are the two-predictor combinations using GAg, which perform on the same level as three-predictor hybrids.

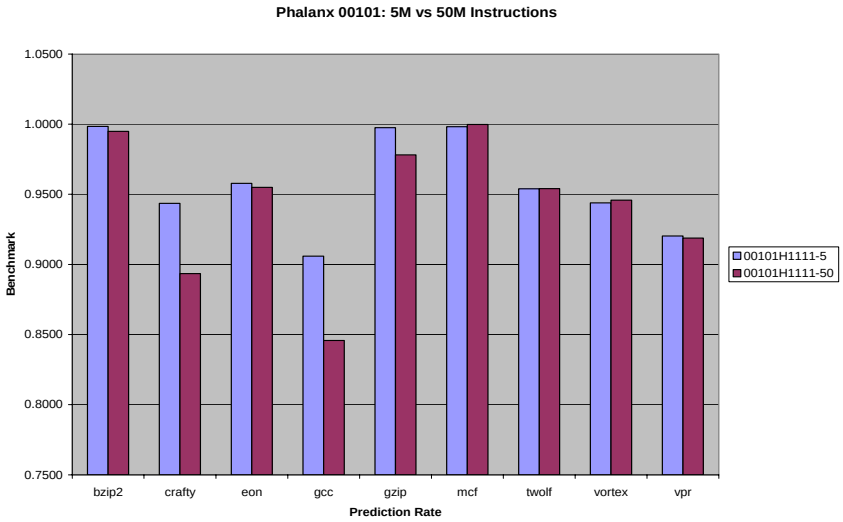
The second characteristic of a good inclusion scheme is that it does not include multiple predictors of the same family which will make redundant predictions. Multiple predictors generating the same prediction for similar reasons limits the robustness of the hybrid predictor. One principal of a combined predictor is that it chooses the best among multiple predictions for a given branch. If all the predictions are made based on common heuristics, then this principal is compromised—it would be wiser to use a single, large predictor. Therefore, with no diversity among constituent predictors, the main benefit of a combined predictor is lost.

6.1 Meta-prediction

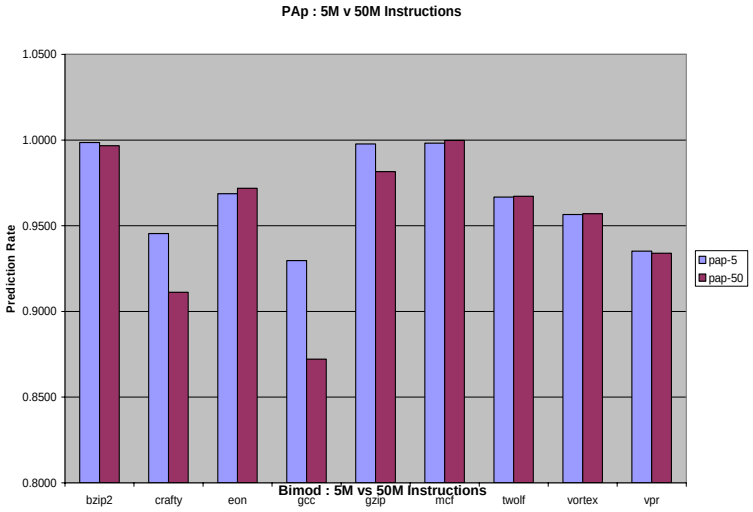
Meta-prediction is used to select one prediction among many sub-predictions, which may disagree. We anticipated this aspect of the project to play a significant role in the prediction rate of Phalanx. To this end, we developed four different meta-prediction schemes: three schemes based on the Phalanx – Majority Vote meta-predictor, and one Phalanx – History scheme. However, as the results showed, the prediction rate was nearly identical across all meta-prediction schemes. We propose a few explanations for these results:

1. *Constituent predictors make correct predictions most of the time.* Slight modification shows that even under the Phalanx – History meta-predictor, the individual constituent predictors very often correctly predict branches. Many correct predictors tend to reduce the need for meta-prediction.
2. *The meta-prediction schemes examined have common principles.* Both use the most recent performance of the sub-predictors to determine per-branch predictions. In retrospect, it would have been interesting to consider less intelligent meta-predictors—perhaps random selection—for comparison.
3. *Ties are arbitrated in similar ways across meta*-prediction schemes. While this has a lesser effect than 1 and 2 (as predictors are often in agreement), nonetheless some variation here could likely change meta-prediction effectiveness.

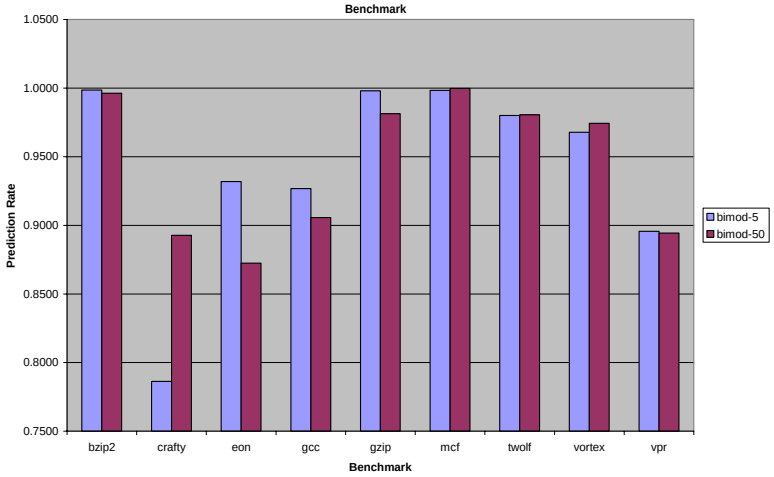
6.2 Effect of the Number of Instructions Simulated



We examined the performance of Phalanx for simulations of length 5M instructions and 50M instructions. We have determined that during the longer simulations, both the hybrid and single predictors performed worse on average than in the shorter simulations. However, the reduction in performance is less dramatic for the hybrid predictors, which indicates that combining predictors improves the performance of at least some branches. This effect is not attributable to predictor warm-up, as predictors should, in theory, improve in performance over time as pattern tables and history registers change to reflect the particular workload.



An exception to this rule is the performance of the stand-alone bimodal predictor on the crafty integer benchmark—its performance improves dramatically between the 5M and 50M instruction simulations. This effect exists somewhat for the PAp predictor, though only very slightly and for different benchmarks.



We notice that for the best-performing multi-predictor configurations, the reduction in Phalanx's accuracy is roughly equivalent to the average reduction in correctness among the constituent predictors. This is an intuitive result, but shows that the selected meta-prediction schemes cannot always correctly ascertain which prediction is most likely to be correct.

7. Future Work

The work done for the Phalanx project leaves many questions unanswered. Given resources and time, it would be possible to explore a greater range of hybrid predictors with the groundwork done for the Phalanx project. For this work, we have selected a fixed (maximal) size for all predictors we evaluated of 8 kbits. It would be interesting to examine the effect of varying this number—intuition suggests that increasing this maximum size would allow combination predictors to use individually larger constituent predictors, and theoretically achieve a greater performance increase than the already-large single predictors used as controls in the experiments above. Due to time constraints, we have not had a chance to explore other different options.

We also have not considered the effect of the length meta-prediction history registers on the Phalanx – History meta-predictor. It is not obvious whether more history bits would yield an improved prediction rate, as the exact histories of individual predictors during execution has not been studied. Reducing the length of the history registers used by the meta-predictors may adversely affect prediction; conversely, lengthening these registers may improve performance. According to several papers that we referenced, dynamic branch classification has proven to be useful in choosing which sub-predictor should be chosen among other sub-predictors in order to yield a better accuracy. More advanced meta-prediction strategies could determine on a per-branch basis which predictors are most likely to predict a given branch correctly, rather than relying on recent performance history. This decision would be based on predictor specialization, and would use special-case predictors in combination with general-case predictors.

An obvious variation that we believe would produce useful results would be to include other predictor families (aside from the two-level and bimodal families used in Phalanx).

Intuitively, employing branch predictors that predict branches using different methods will help to minimize redundant predictions among constituent predictors, and would lead to richer sub-predictions. Additionally, we could avoid redundant information stored in more than one sub-predictor.

Furthermore, Phalanx only considers the case where each sub-predictor is allotted an equal portion of the available storage space. Successful hybrid predictors often opt to weight the sizes of sub-predictors unequally [6,8,10]. Uneven resource allocation would lend itself well to the most effective combination schemes seen above, and could be used to improve the performance of the poorest performing combinations through increased sub-predictor size for one or two constituent predictors (at the expense of others). This would lead naturally into a modification of the Phalanx – Majority Vote meta-predictor, giving greater voting weights to predictors that are large in overall size.

Finally, a number of statistic-based heuristics could be incorporated into Phalanx, especially among the meta-predictors. Among these could be a bias toward a prediction of taken for looping branches, or a prediction biased toward not taken for forward directed branches. This modification would be simple under the Phalanx – Majority Vote meta-predictor.

8. References

[1] T-Y. Yeh and Y. Patt, “Two-level Adaptive Training Branch Prediction”, *Proceedings of the 24th Annual International Symposium on Microarchitecture,* 1991

[2] T-Y Yeh and Y. N. Patt, “Alternative Implementations of Two-Level Adaptive Branch Prediction”, *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992, pp. 124-134

[3] T-Y Yeh and Y. N. Patt, “A Comparison of Dynamic Branch Predictors that Use Two-Levels of Branch History”, *The 20th Annual International Symposium on Computer Architecture*, May 1993, pp. 257-266

[4] J. Hennessy and D. Patterson, "Computer Architecture: A Quantitative Approach, 2nd Edition," Morgan Kaufmann Publishers, Inc., 1996

[5] J. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", *IEEE Computer Magazine 17*, 1 Jan. 1984

[6] S. McFarling, “Combining Branch Predictors”, *Digital Equipment Corporation WRL Technical Note TN-36*, June 1993

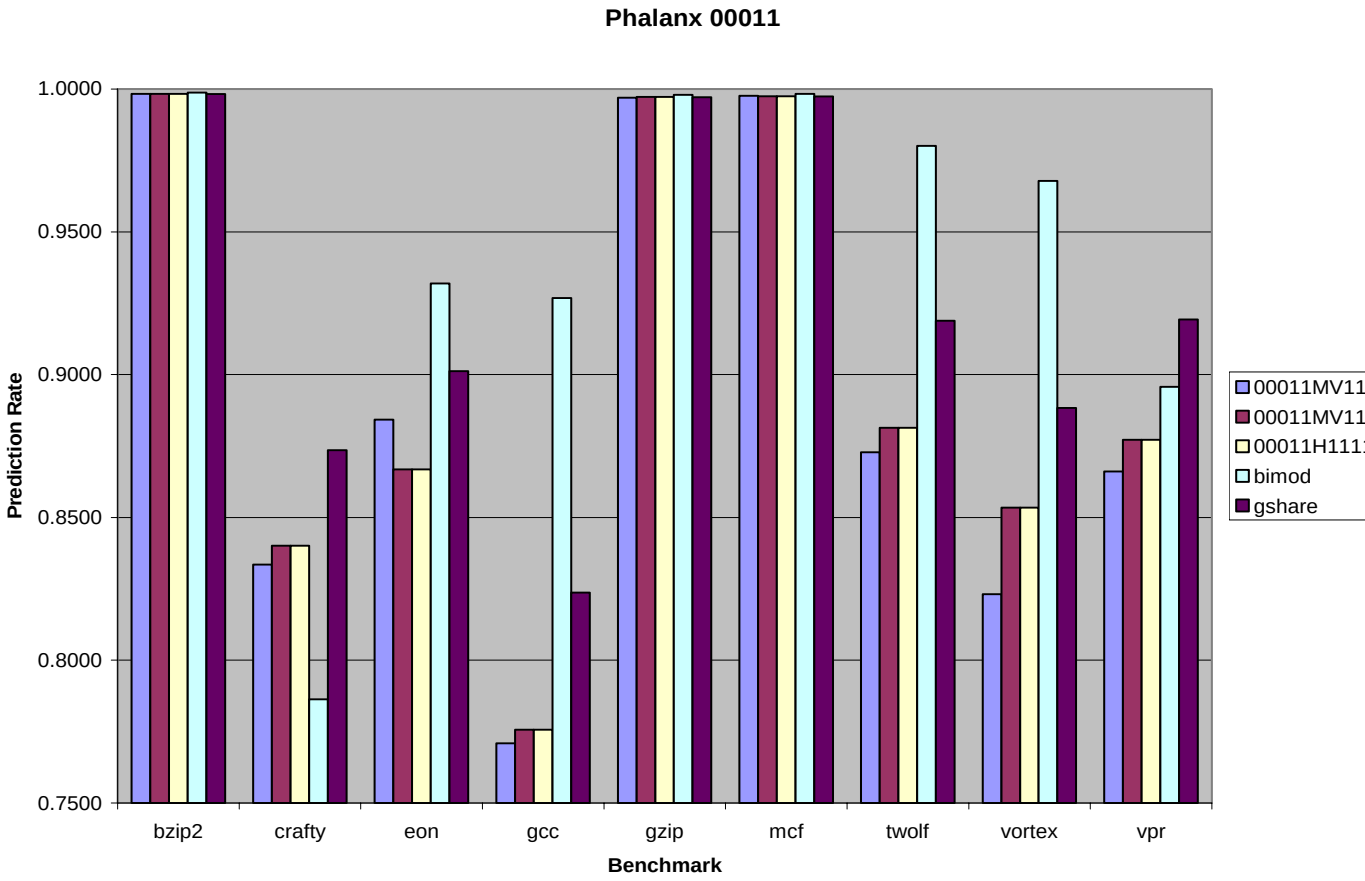
[7] C. Young, N. Gray, and M. D. Smith, “A Comparative Analysis of Schemes for Correlated Branch Prediction”, *Proceedings of the 22nd Annual International Symposium on Computer Architecture,* 1995, pp. 276 – 286

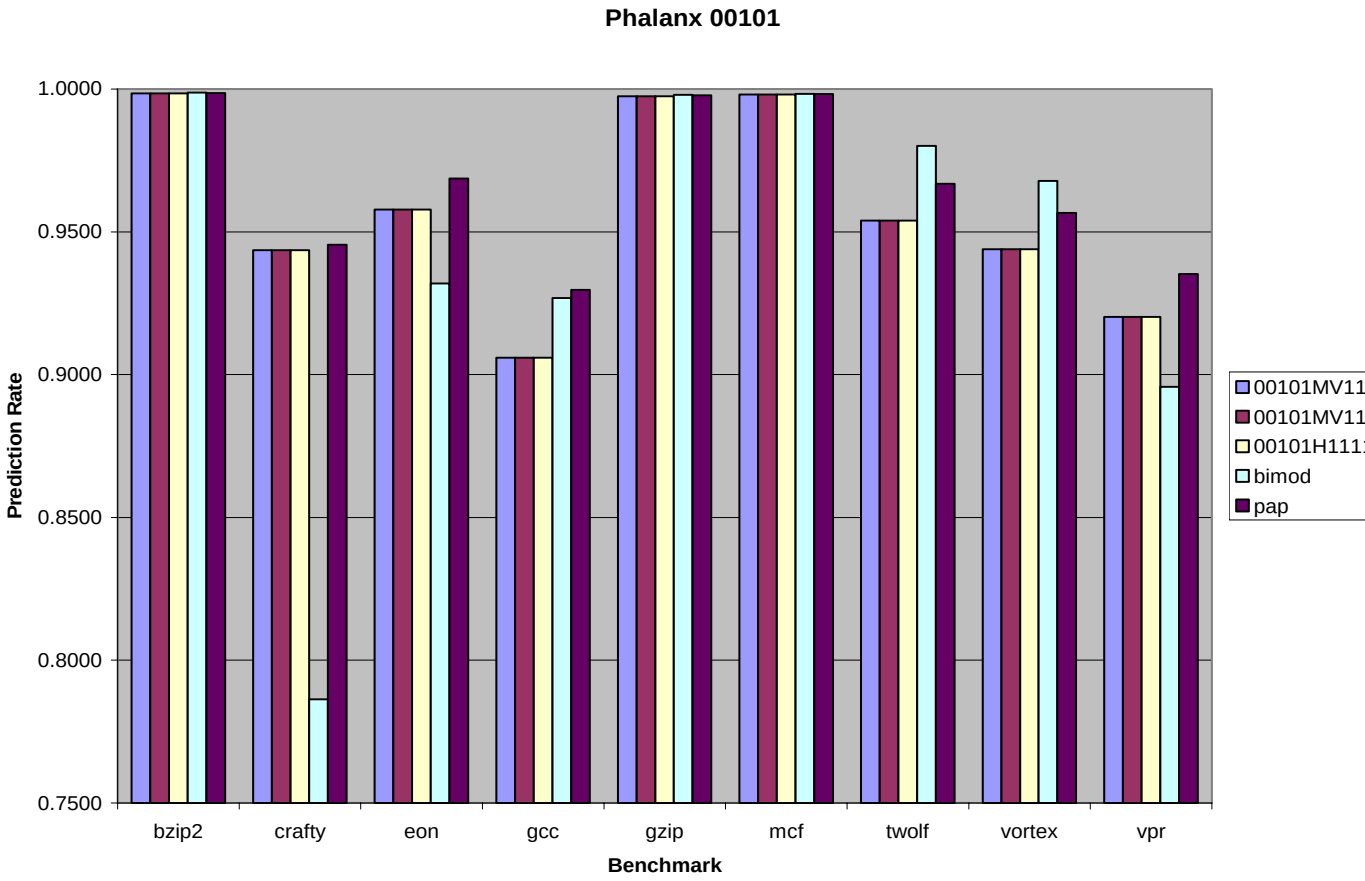
[8] P. Chang, E. Hao, and Y. Patt, “Alternate Implementations of Hybrid Branch Predictors”, *Proceedings of the 28th Annual International Symposium on Microarchitecture,* 1995.

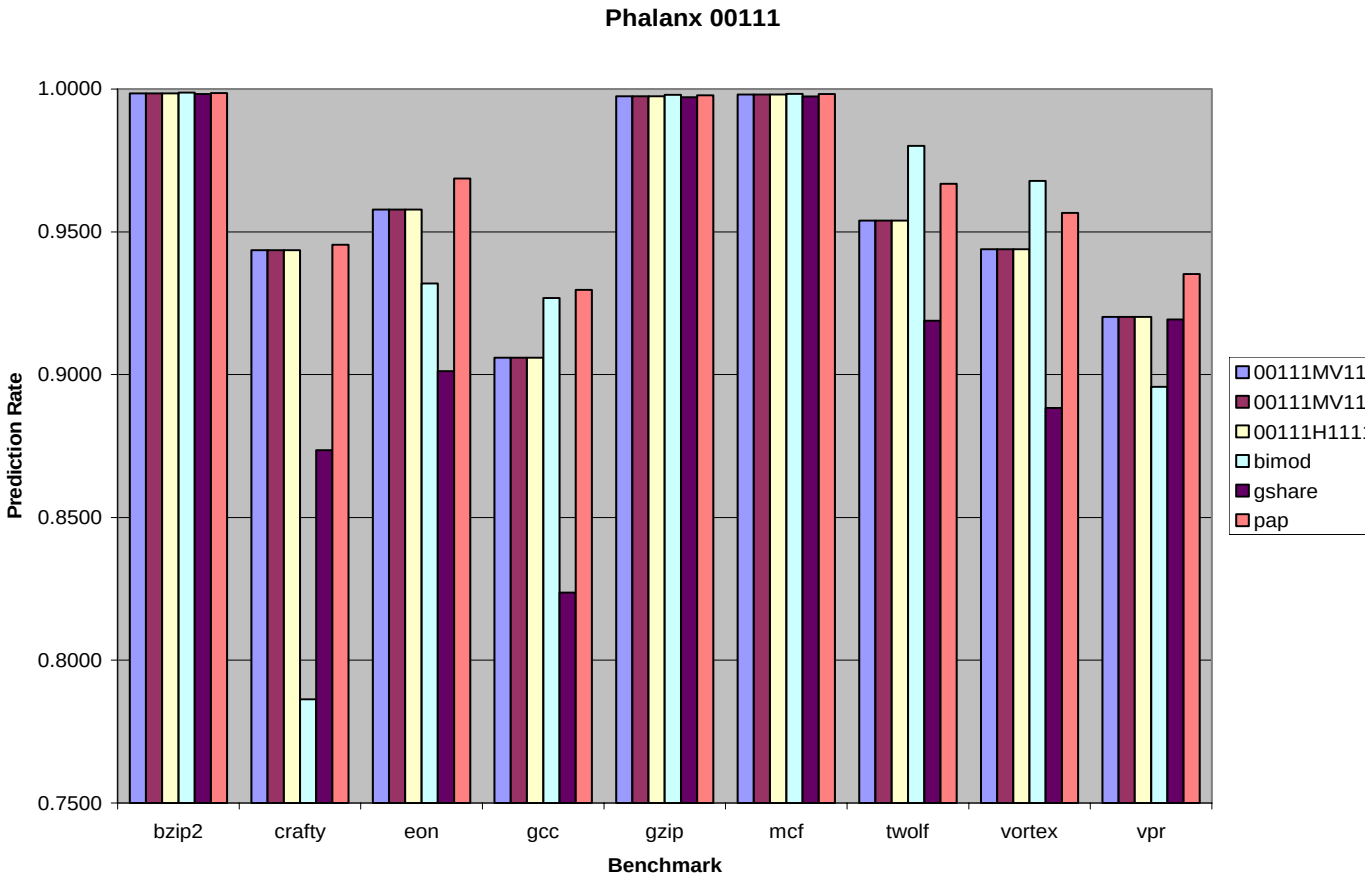
[9] K. Driesen and U. Hölzle, “The Cascaded Predictor: Economical and Adaptive Branch Target Prediction”, *Proceedings of the 31st Annual International Symposium on ACM/IEEE*, 1998, pp. 249–258

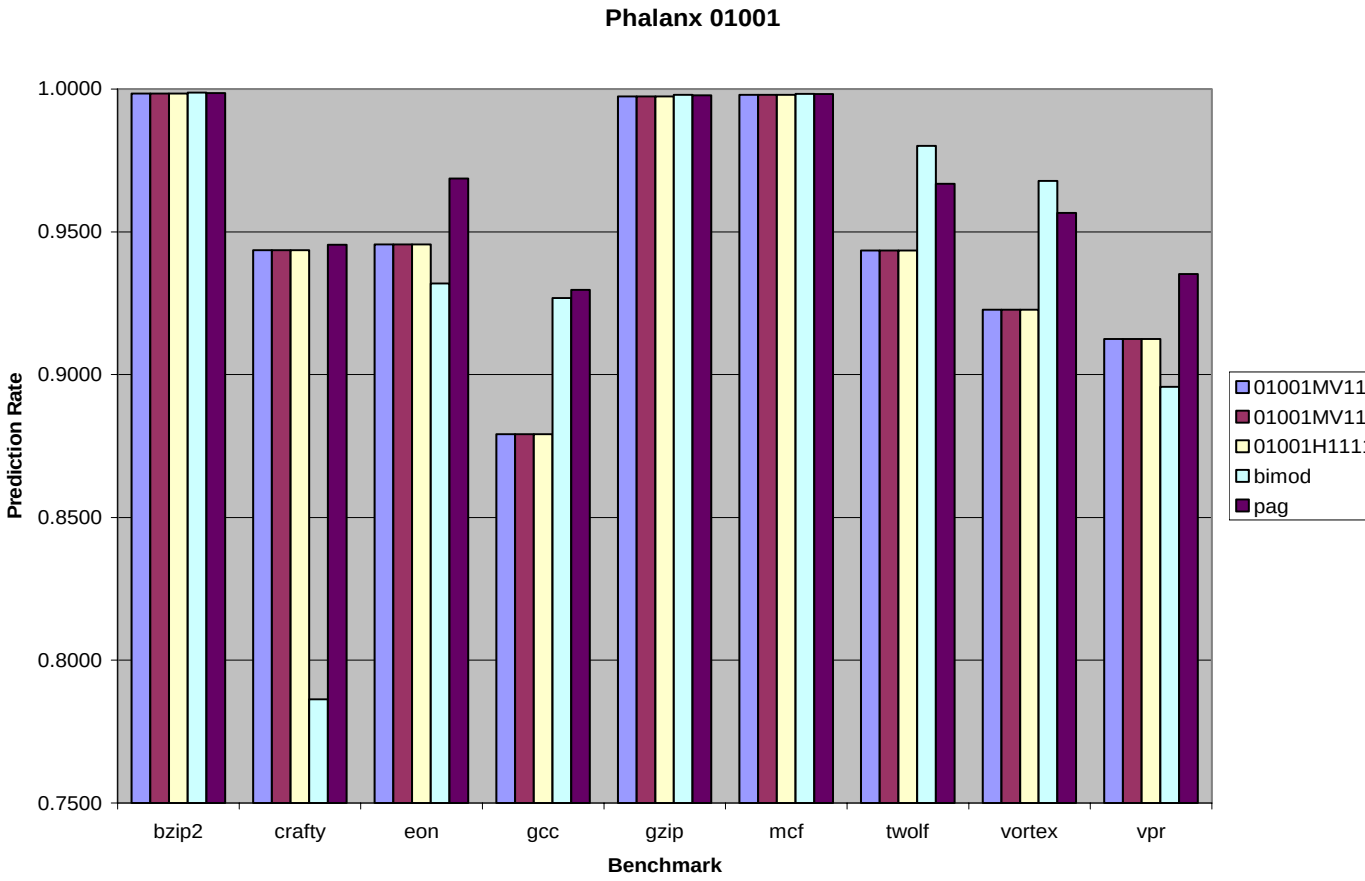
[10] Harish Patil and Joel Emer. “Combining Static and Dynamic Branch Prediction to Reduce Destructive Aliasing”, *Proceedings of the 6th Annual International Symposium on High-Performance Computer Architecture*, 2000, pp. 251-262

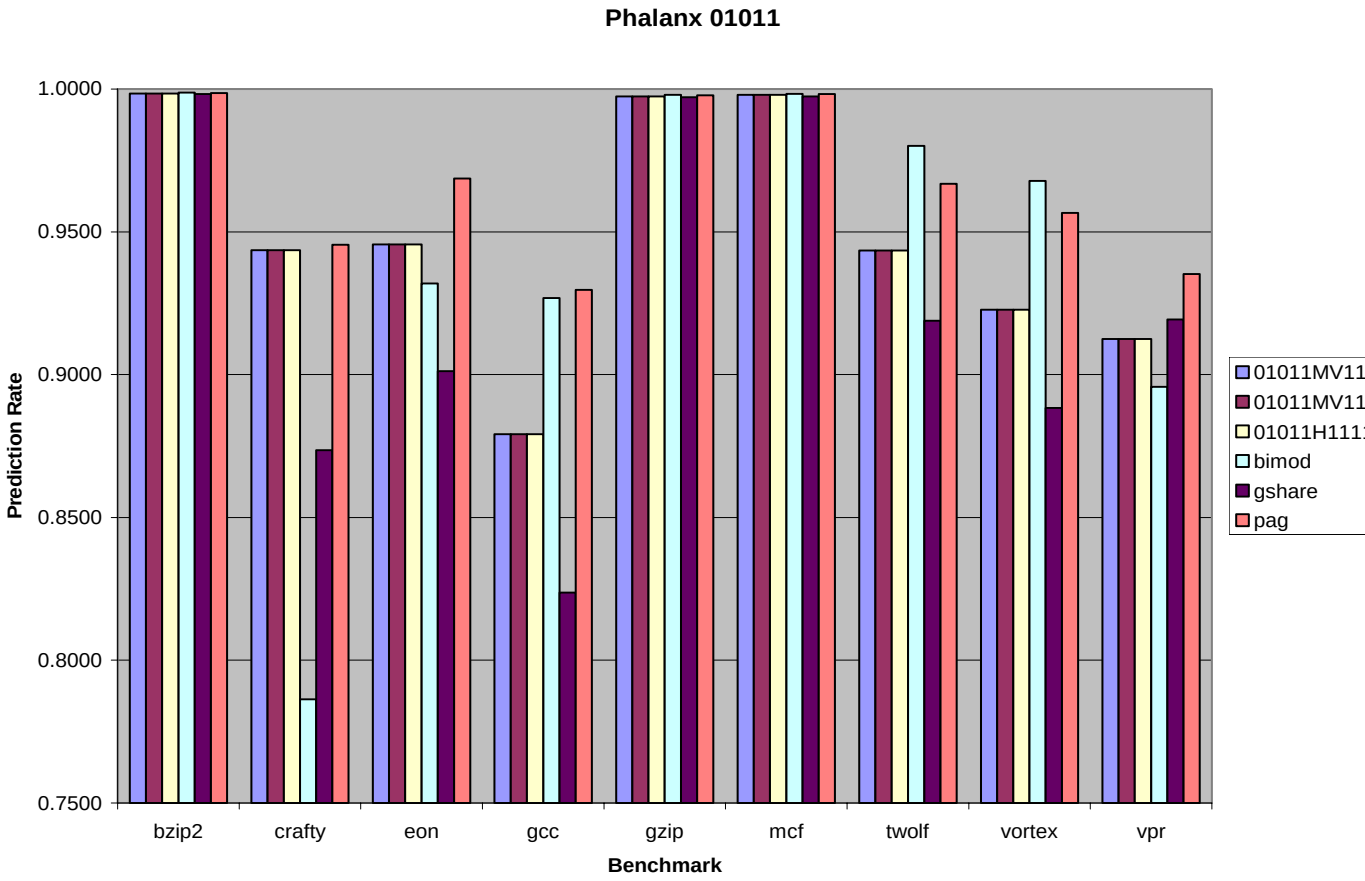
Appendix A
Charts and Tables
5M Instructions

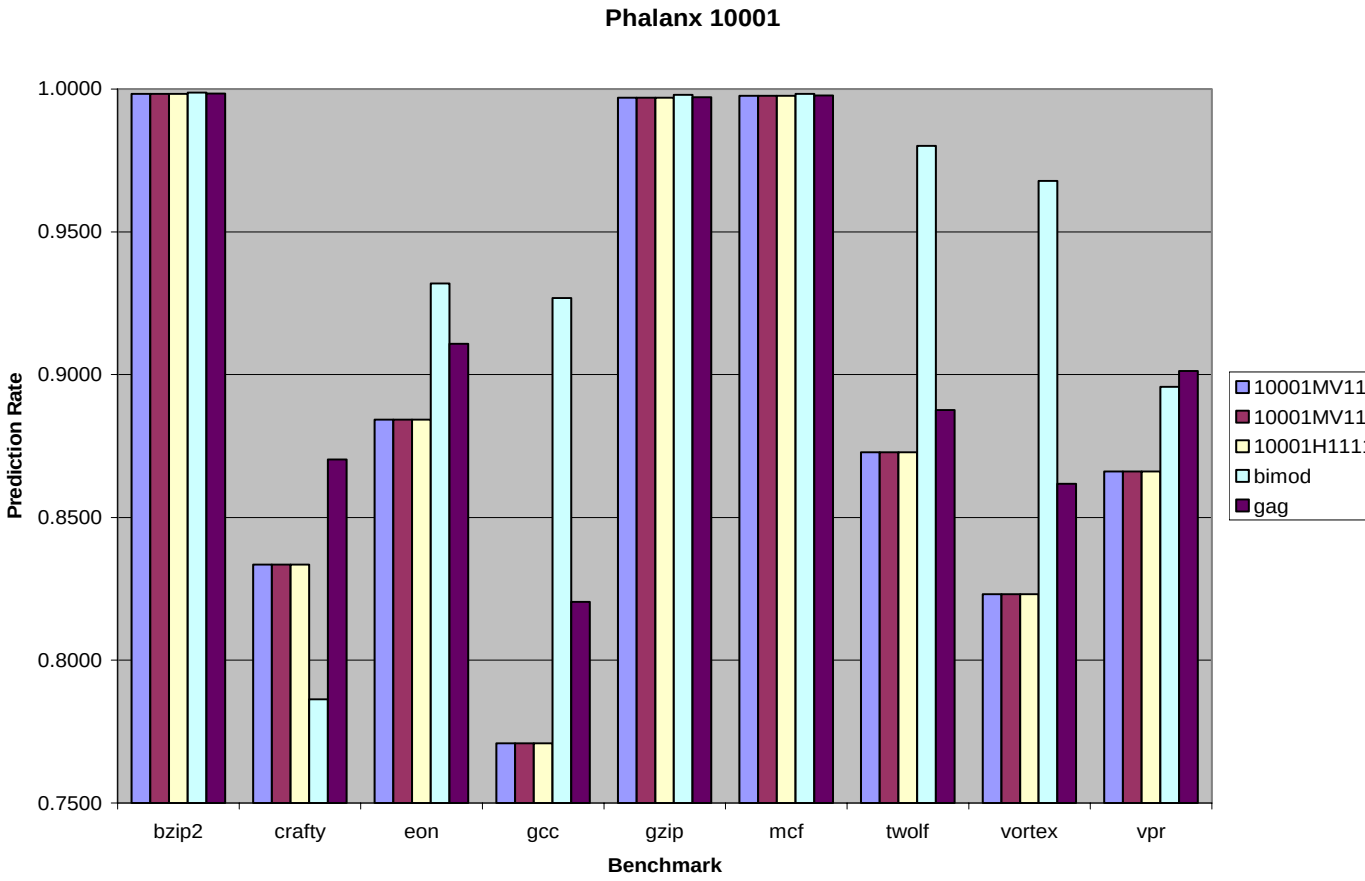

Phalanx 00011
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
00011MV11
00011MV11
00011H111
bimod
gshare

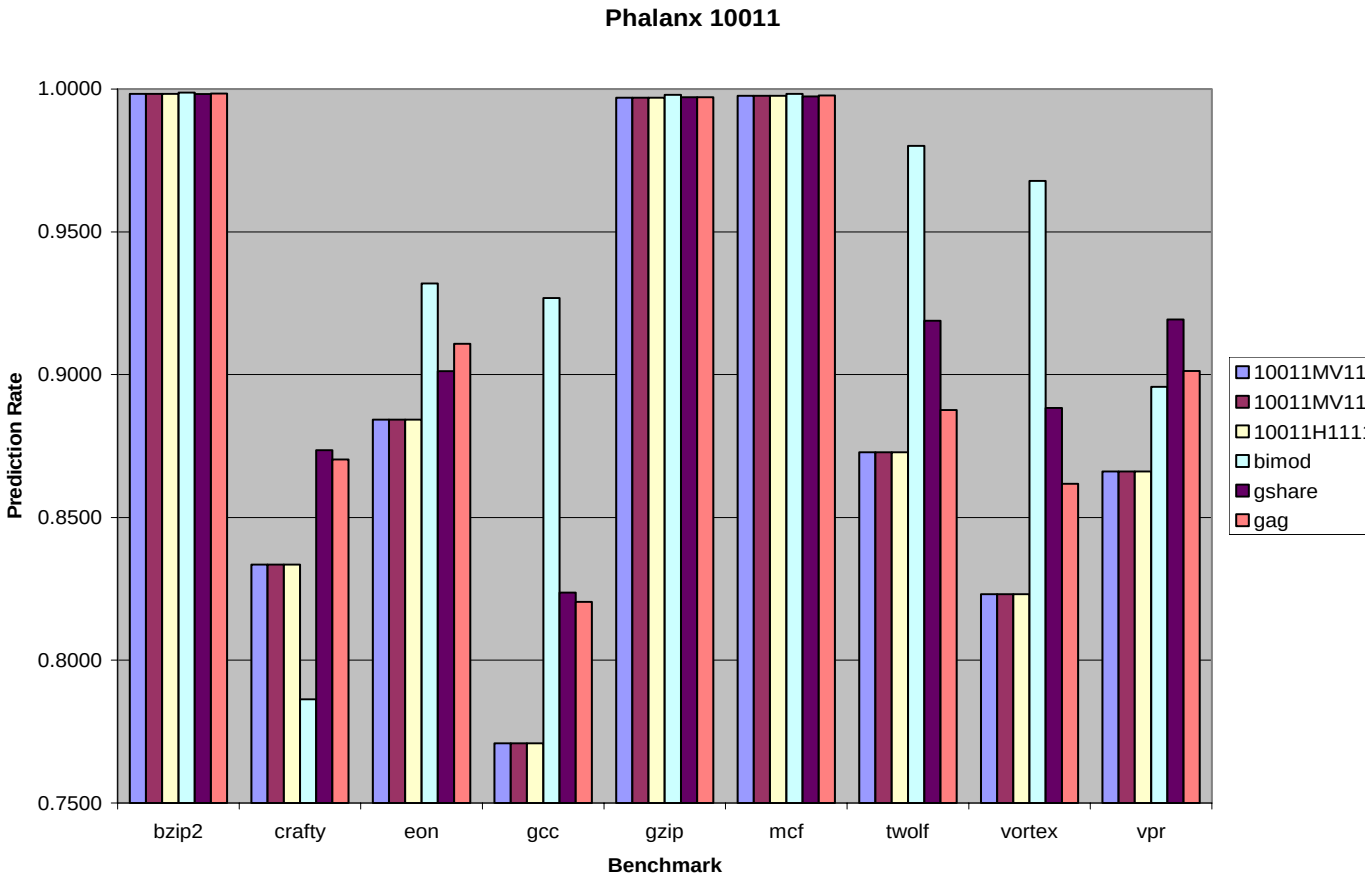

Phalanx 00101
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
00101MV11
00101MV11
00101H111
bimod
pap

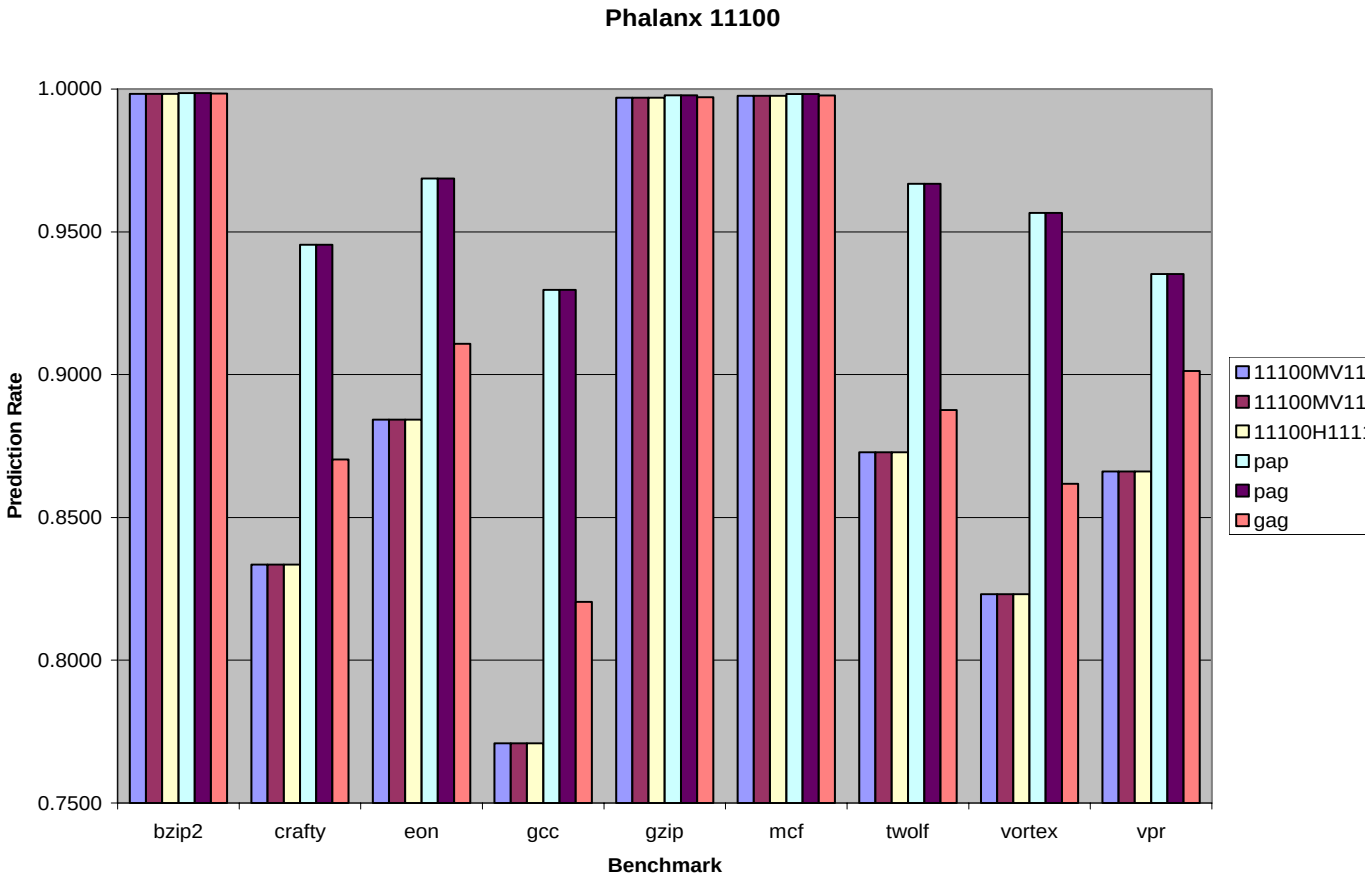

Phalanx 00111
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
00111MV11
00111MV11
00111H111
bimod
gshare
pap

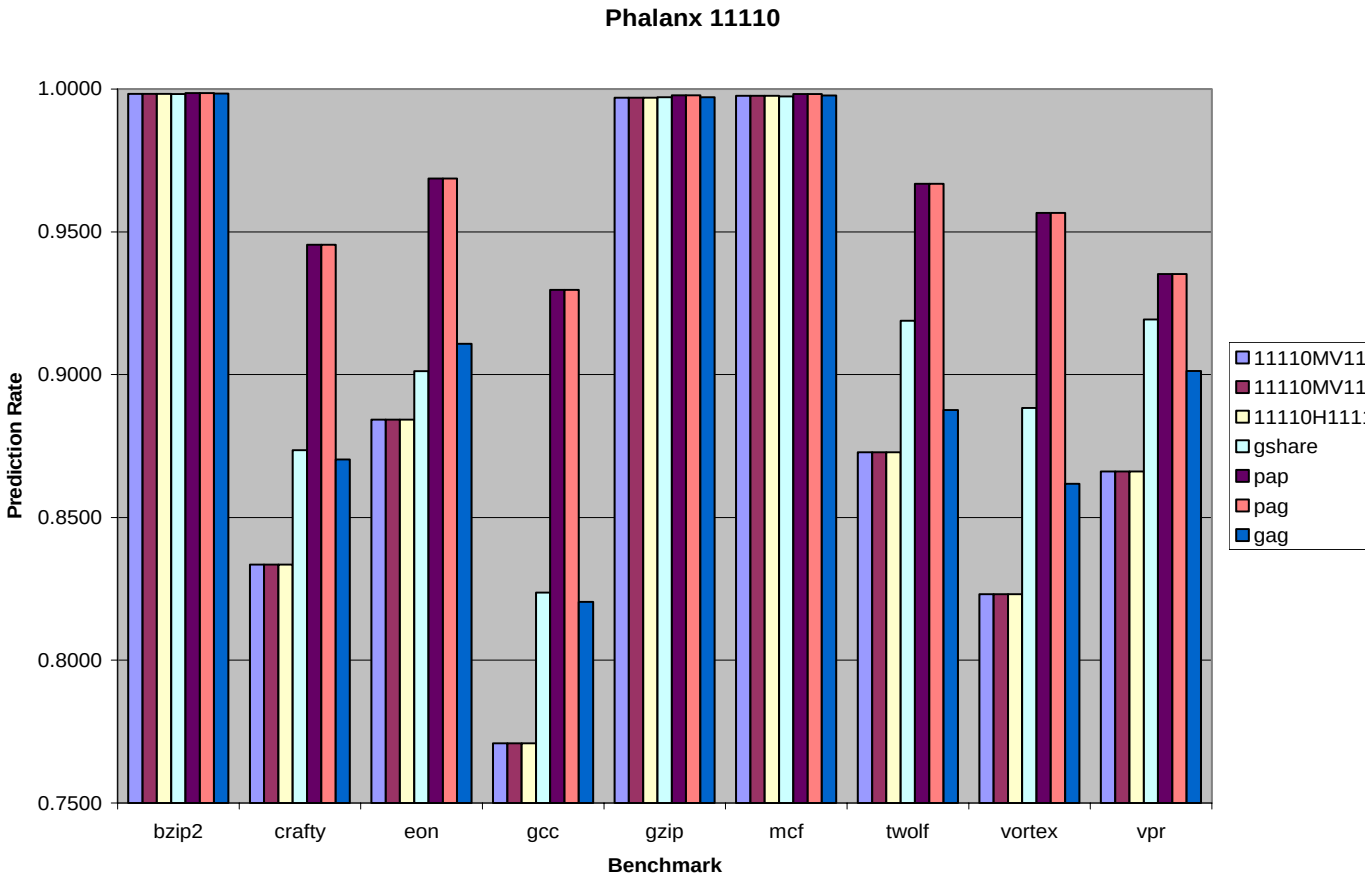

Phalanx 01001
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
01001MV11
01001MV11
01001H111
bimod
pag

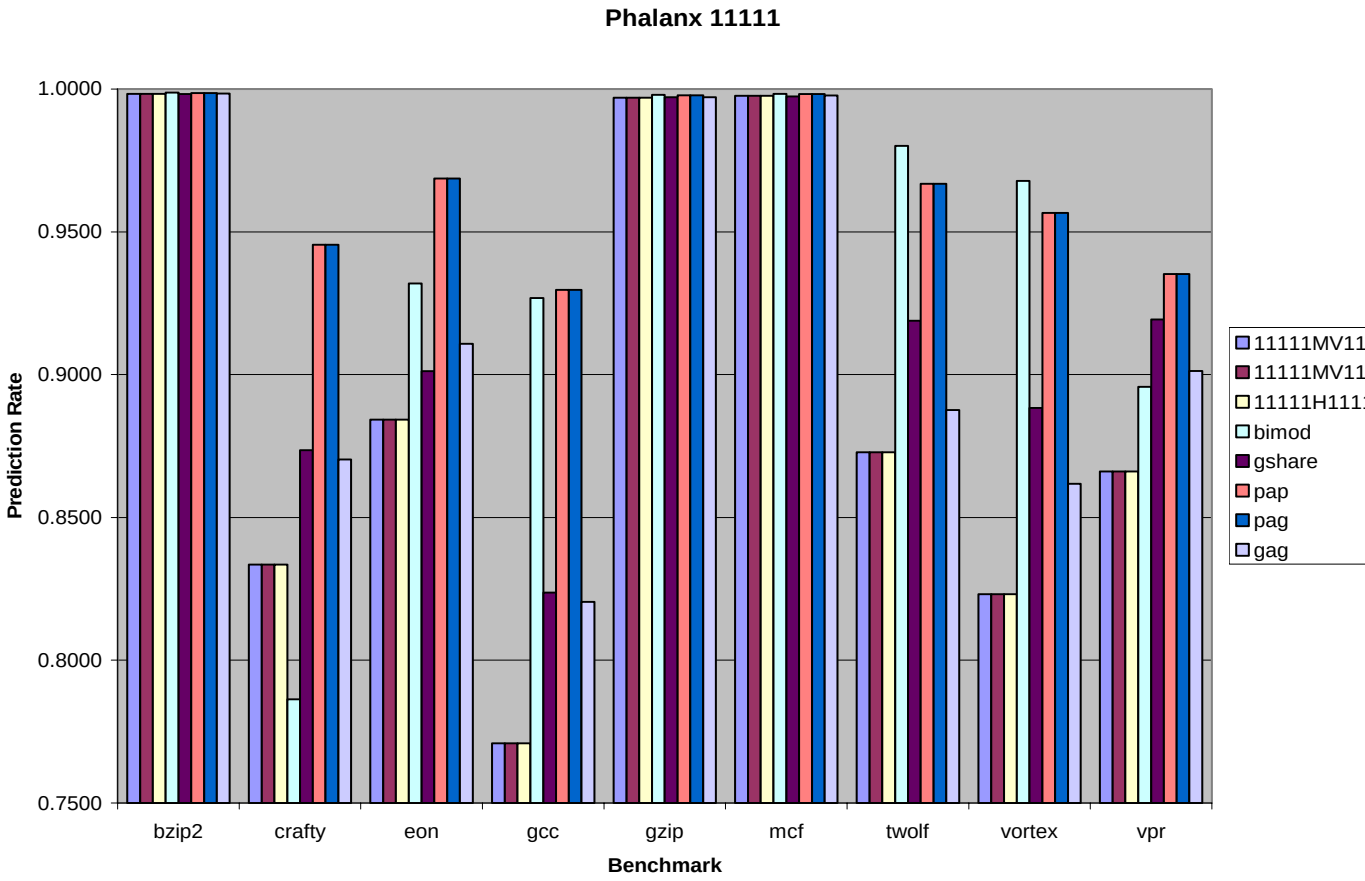

Phalanx 01011
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
01011MV11
01011MV11
01011H111
bimod
gshare
pag


Phalanx 10001
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
10001MV11
10001MV11
10001H111
bimod
gag


Phalanx 10011
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
10011MV11
10011MV11
10011H111
bimod
gshare
gag


Phalanx 11100
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
11100MV11
11100MV11
11100H111
pap
pag
gag

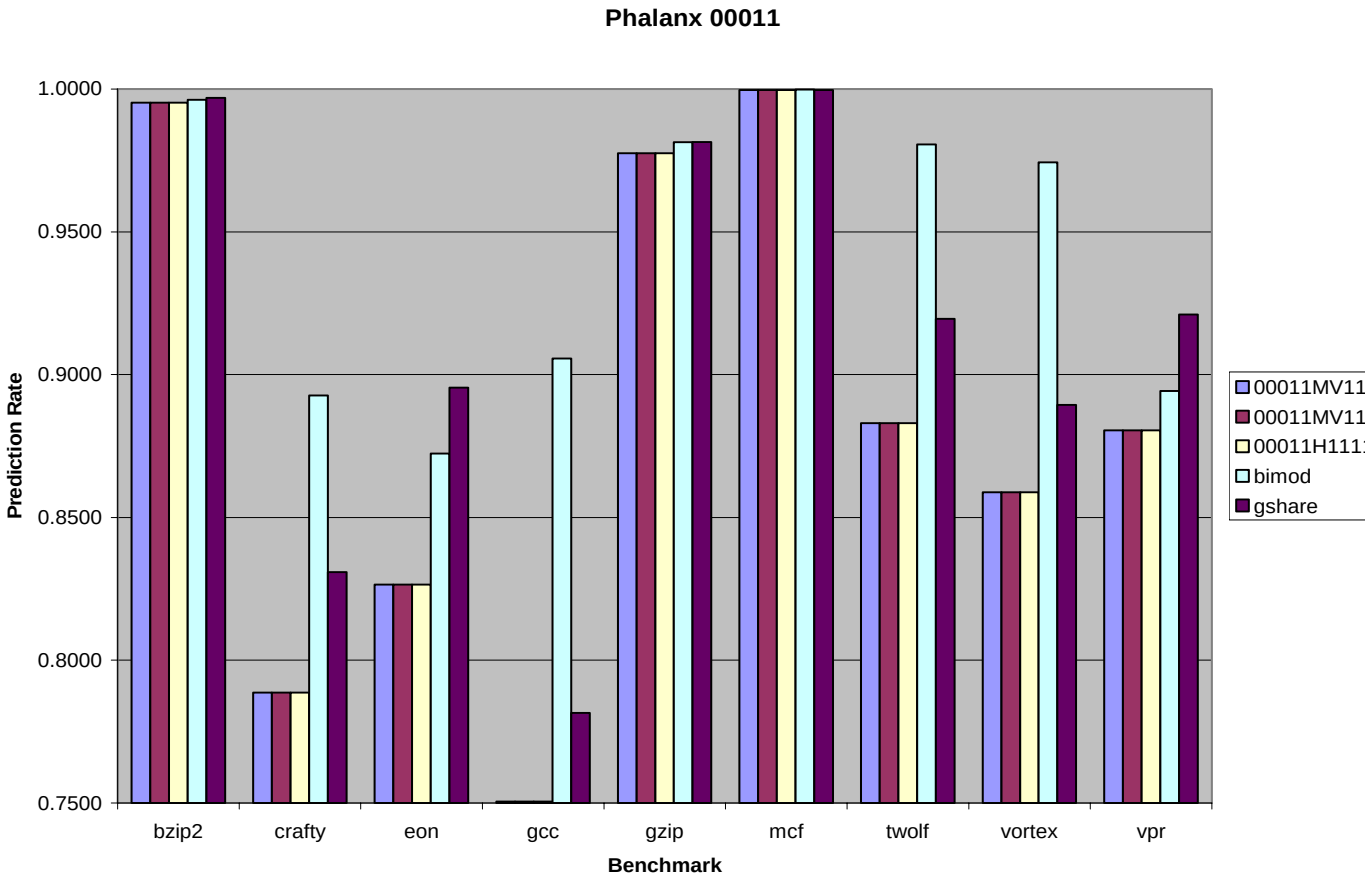

Phalanx 11110
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
11110MV11
11110MV11
11110H111
gshare
pap
pag
gag

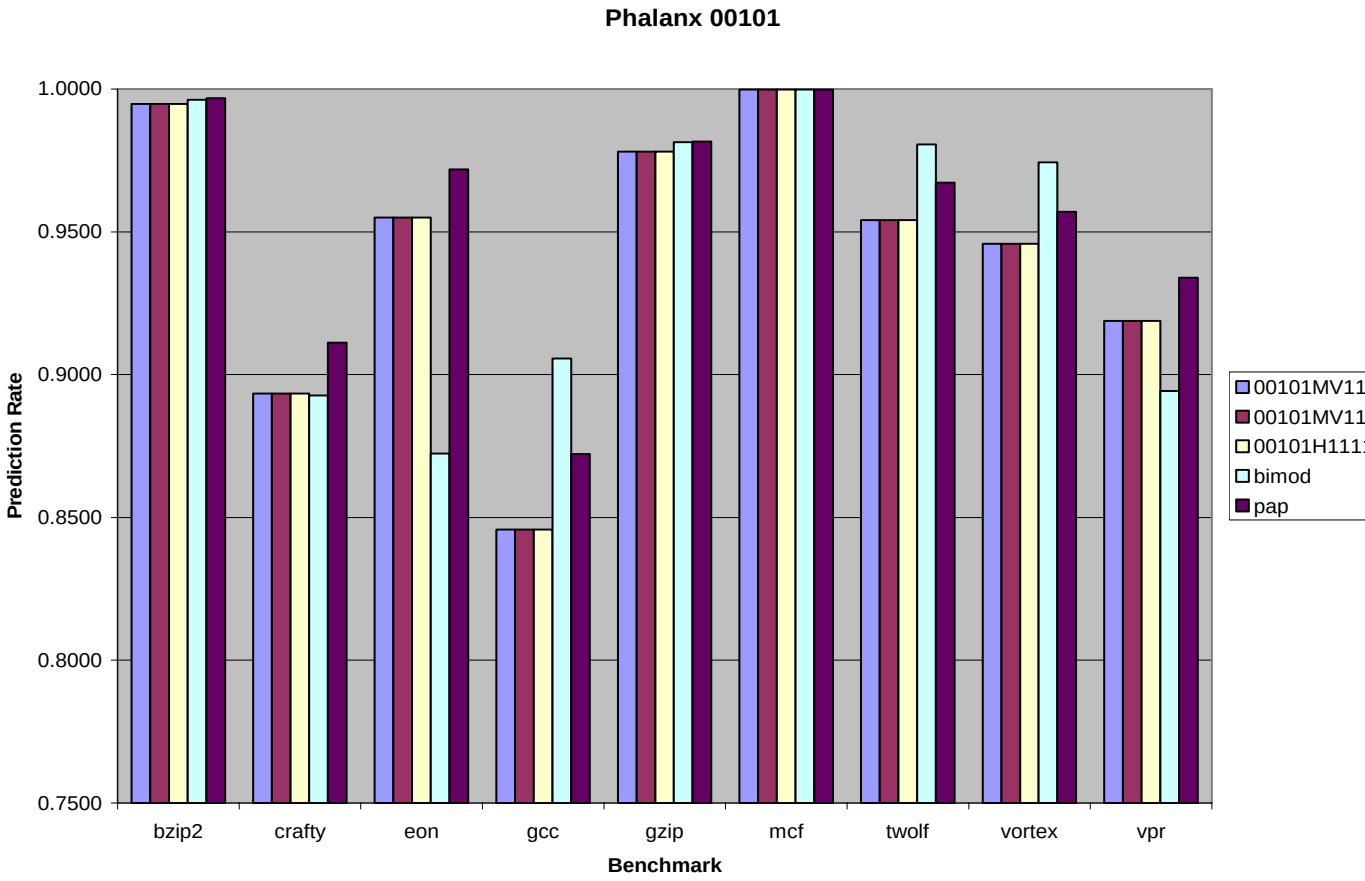

Phalanx 11111
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
11111MV11
11111MV11
11111H111
bimod
gshare
pap
pag
gag

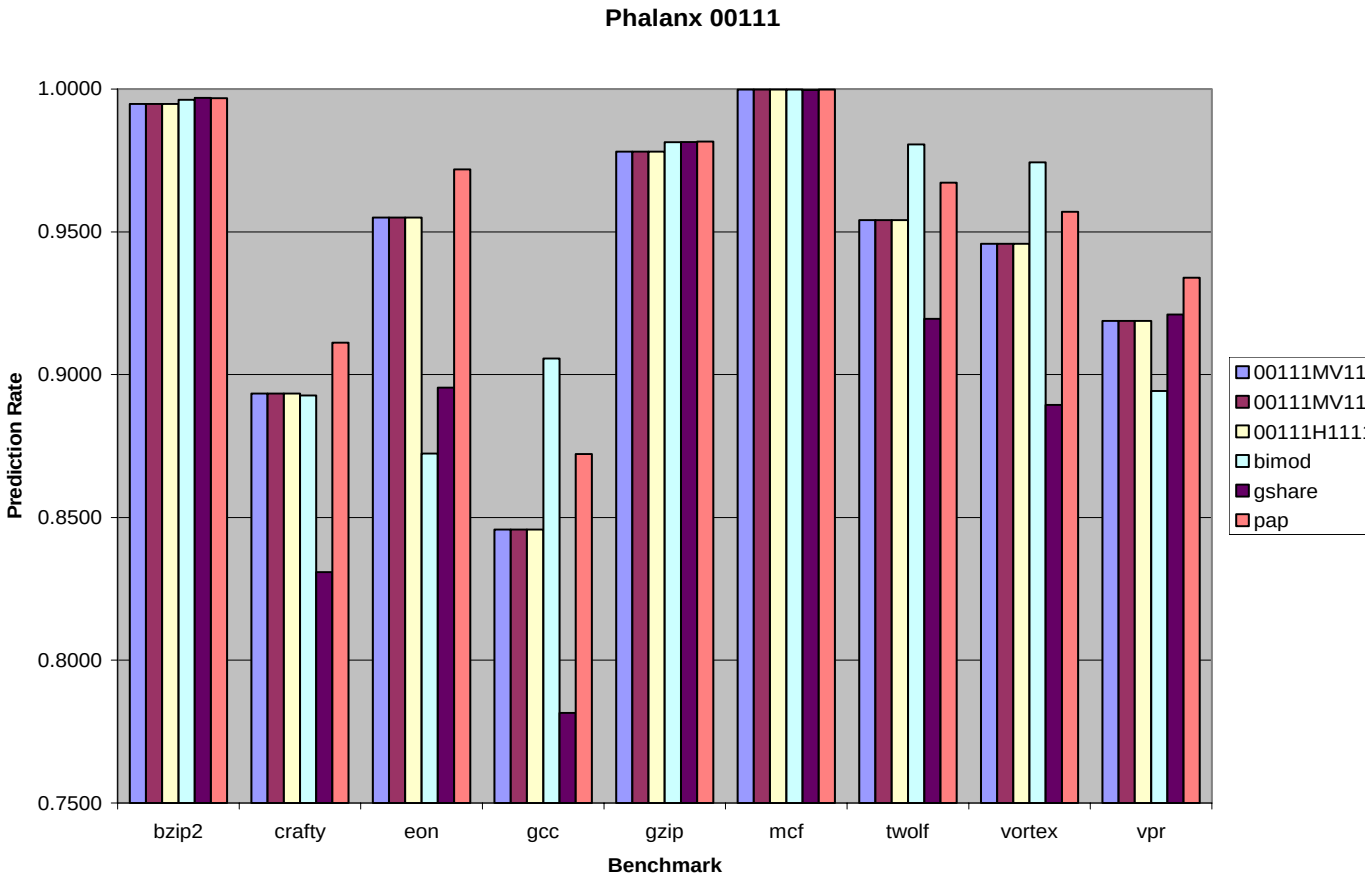
	bzip2	crafty	eon	gcc	gzip	mcf	twolf	vortex	vpr
00011MV1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
00011MV1125	0.9983	0.8401	0.8668	0.7757	0.9972	0.9975	0.8814	0.8534	0.8772
00011MV125A	0.9983	0.8401	0.8668	0.7757	0.9972	0.9975	0.8814	0.8534	0.8772
00101MV1111	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
00101MV1125	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
00101MV125A	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
00111MV1111	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
00111MV1125	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
00111MV125A	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
01001MV1111	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01001MV1125	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01001MV125A	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01001MV1111	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01001MV1125	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01001MV125A	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01011MV1111	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01011MV1125	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01011MV125A	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
10001MV1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
10001MV1125	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
10001MV125A	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
10011MV1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
10011MV1125	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
10011MV125A	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11100MV1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11100MV1125	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11100MV125A	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11110MV1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11110MV1125	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11110MV125A	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11111MV1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11111MV1125	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11111MV125A	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
00011H1111	0.9983	0.8401	0.8668	0.7757	0.9972	0.9975	0.8814	0.8534	0.8772
00101H1111	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
00111H1111	0.9985	0.9436	0.9578	0.9059	0.9975	0.9981	0.9539	0.9439	0.9202
01001H1111	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01001H1111	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
01011H1111	0.9984	0.9436	0.9456	0.8791	0.9974	0.9980	0.9435	0.9228	0.9125
10001H1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
10011H1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11100H1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11110H1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
11111H1111	0.9983	0.8335	0.8843	0.7709	0.9970	0.9976	0.8728	0.8231	0.8661
bimod	0.9987	0.7863	0.9319	0.9268	0.9980	0.9983	0.9801	0.9678	0.8957

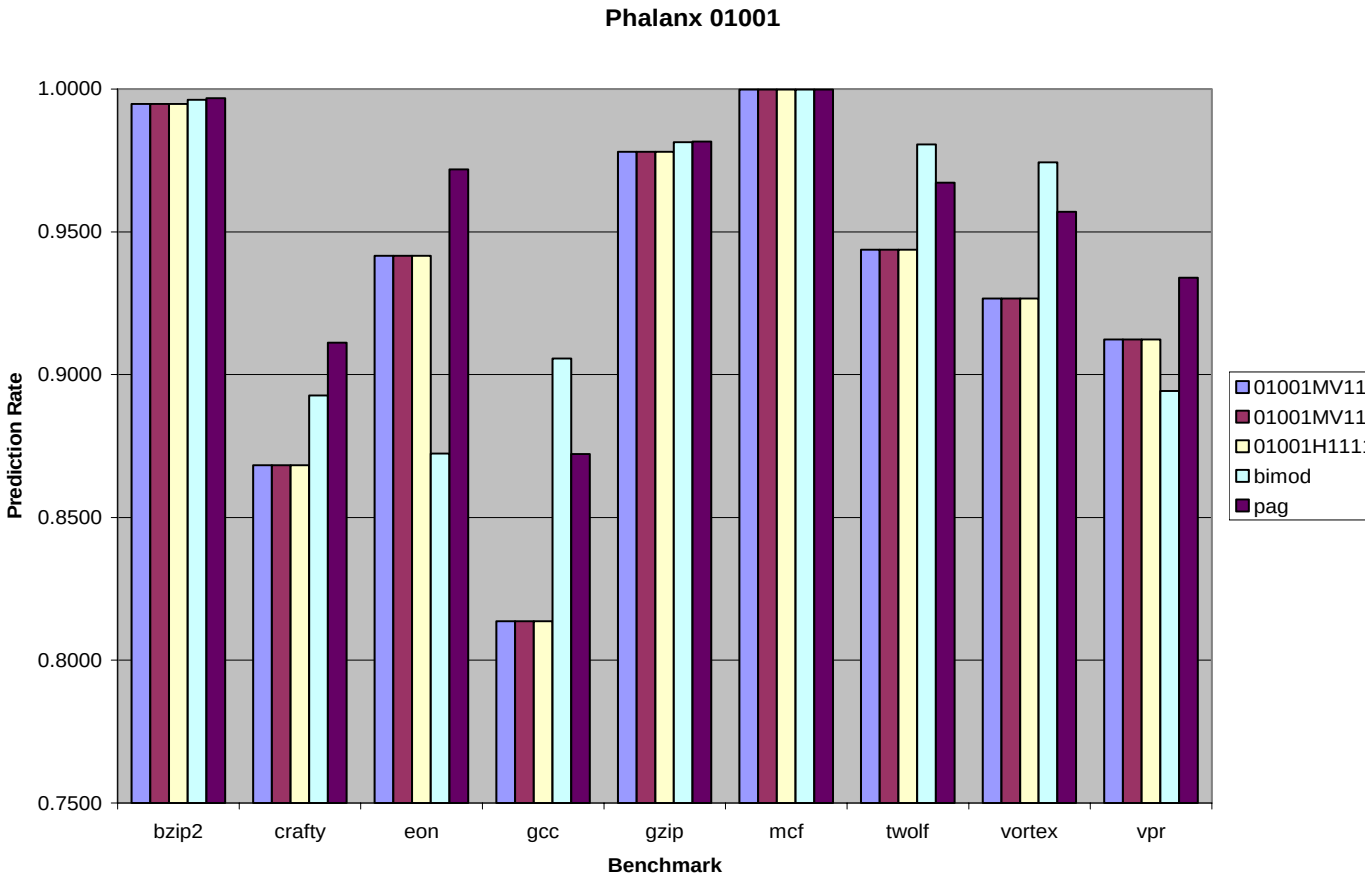
gshare	0.9982	0.8736	0.9012	0.8237	0.9971	0.9974	0.9189	0.8883	0.9193
pap	0.9986	0.9455	0.9687	0.9297	0.9978	0.9982	0.9668	0.9566	0.9352
pag	0.9986	0.9455	0.9687	0.9297	0.9978	0.9982	0.9668	0.9566	0.9352
gag	0.9984	0.8703	0.9108	0.8204	0.9971	0.9977	0.8876	0.8618	0.9013

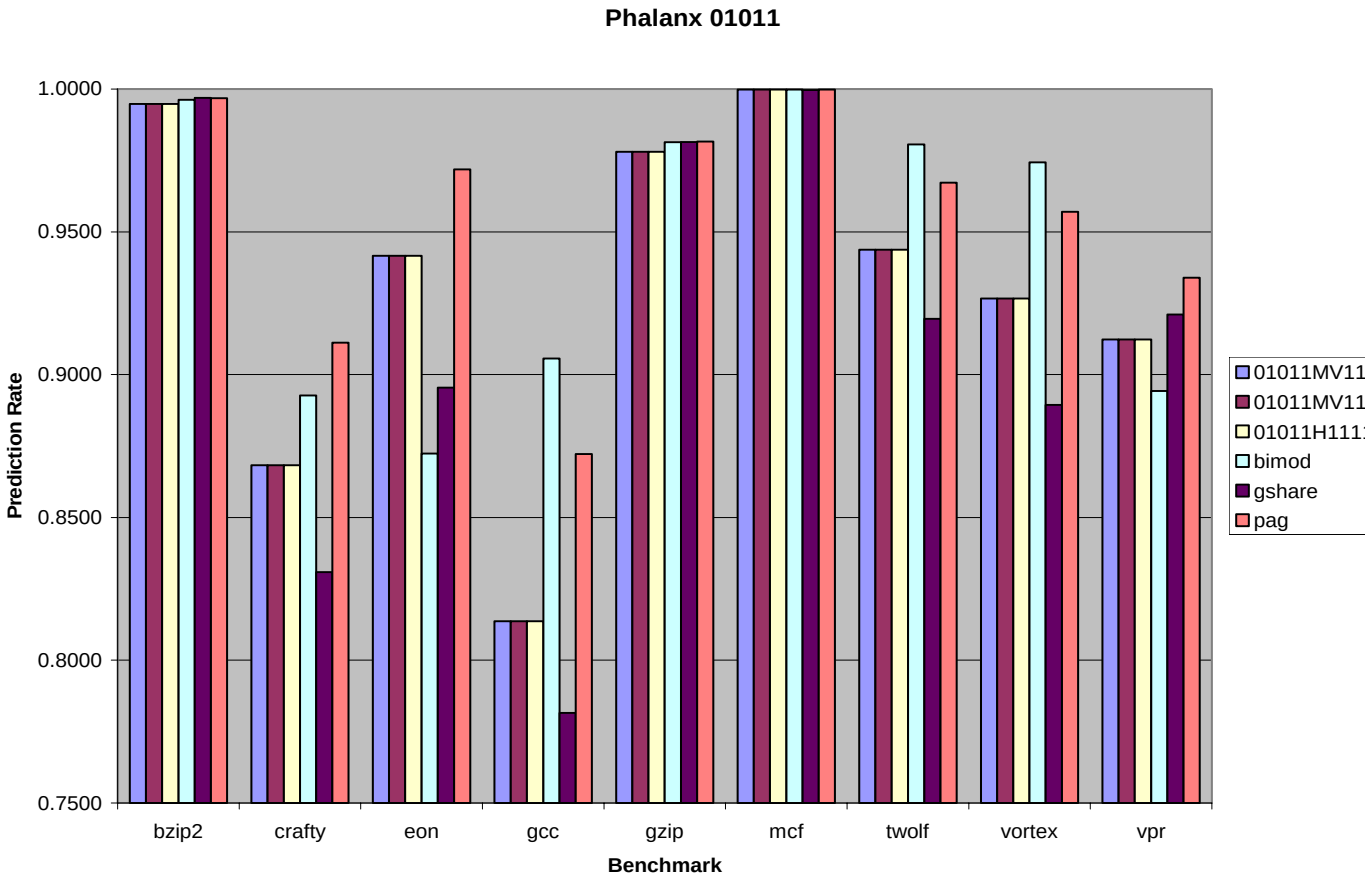
Appendix B
Charts and Tables
50M Instructions

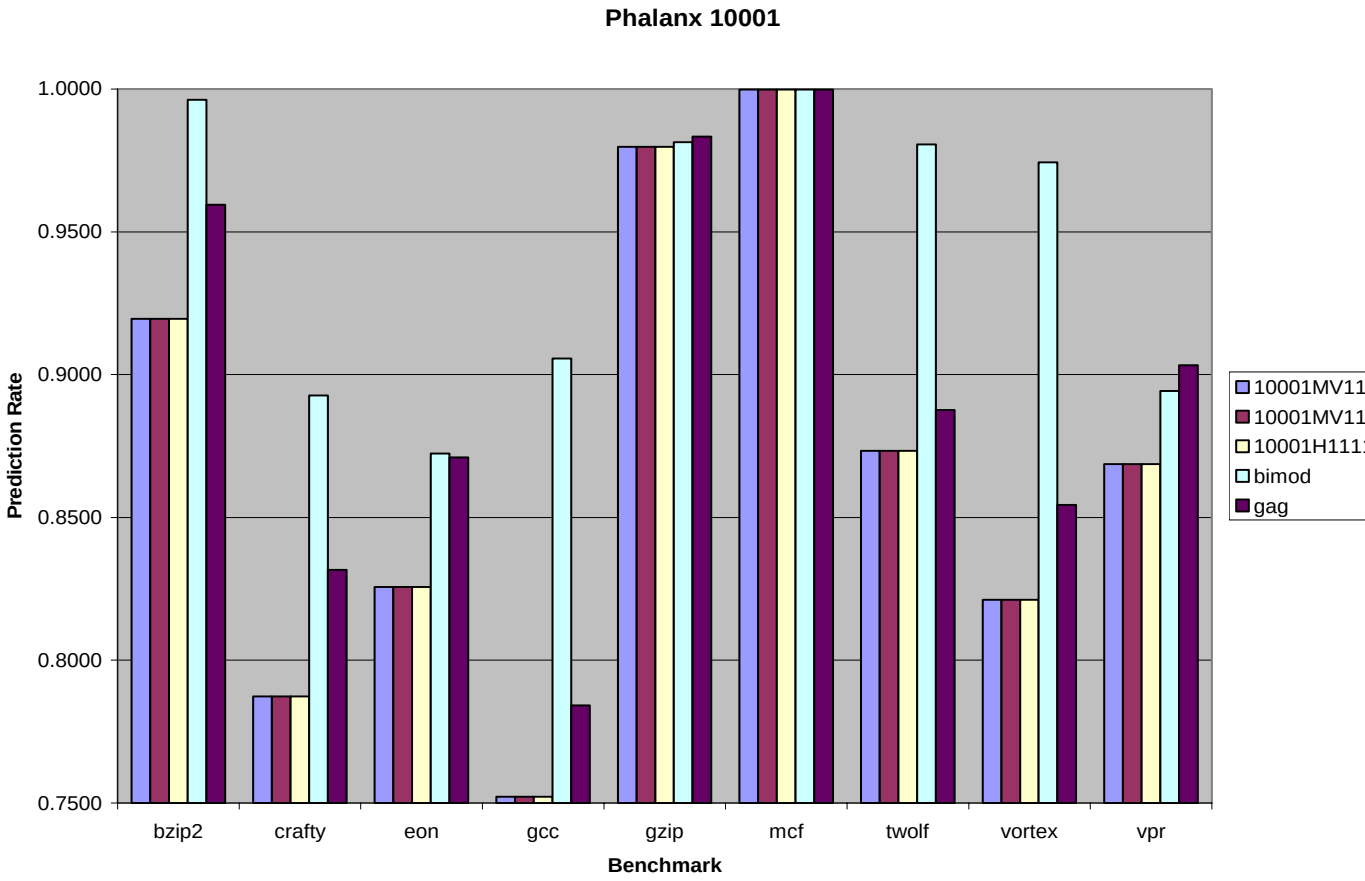

Phalanx 00011
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
00011MV11
00011MV11
00011H111
bimod
gshare

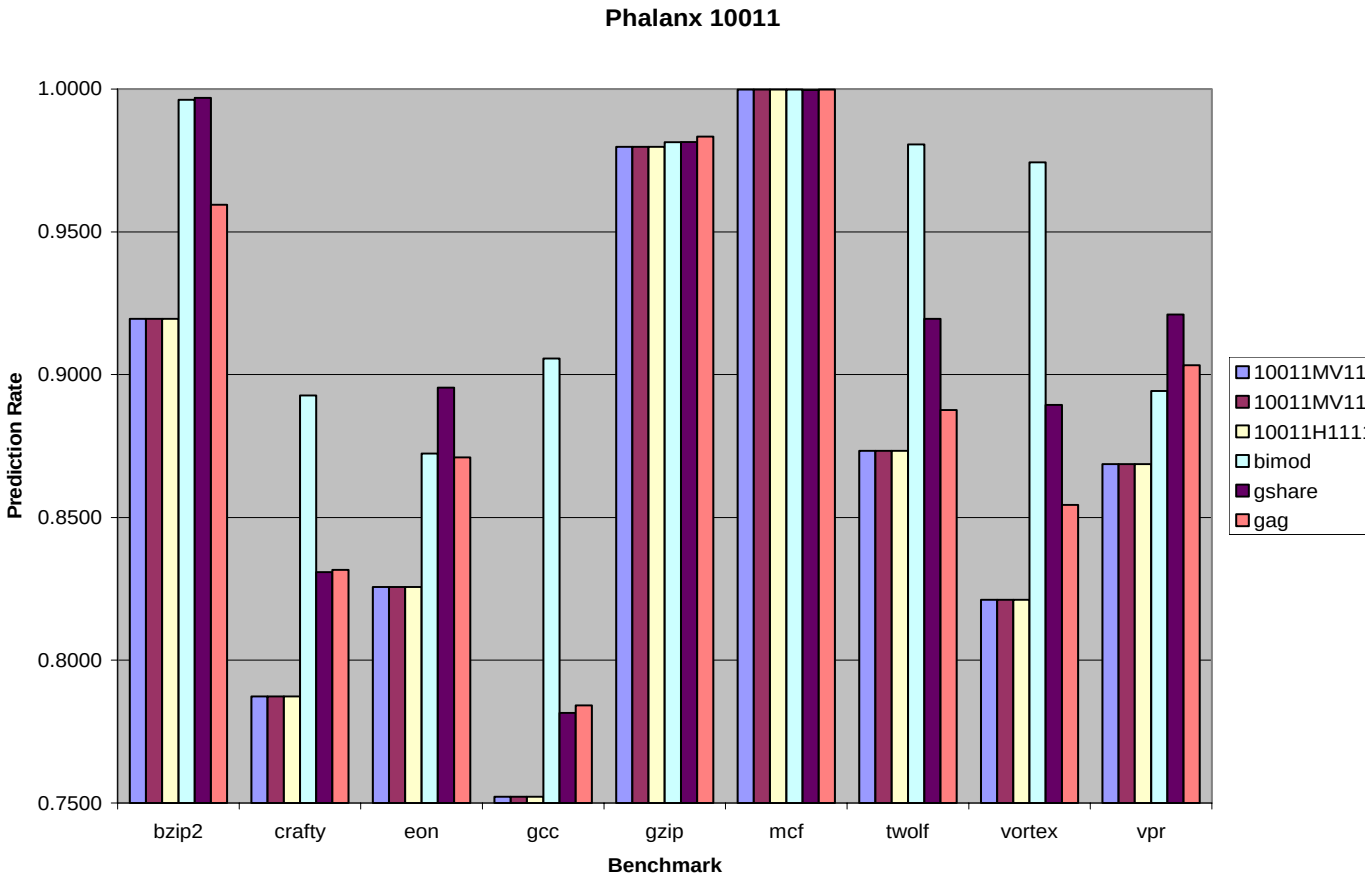

Phalanx 00101
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
00101MV11
00101MV11
00101H111
bimod
pap

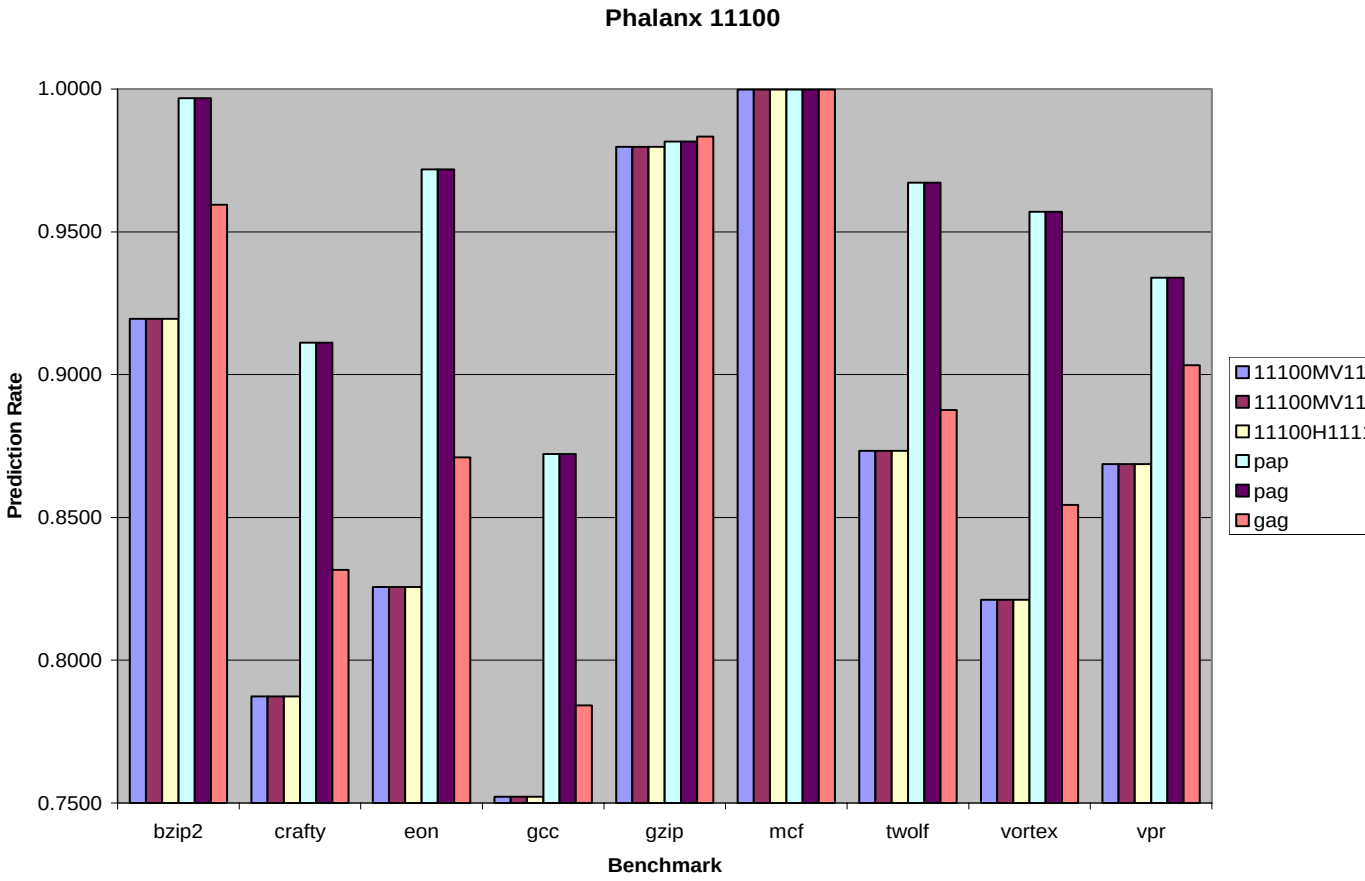

Phalanx 00111
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
00111MV11
00111MV11
00111H111
bimod
gshare
pap

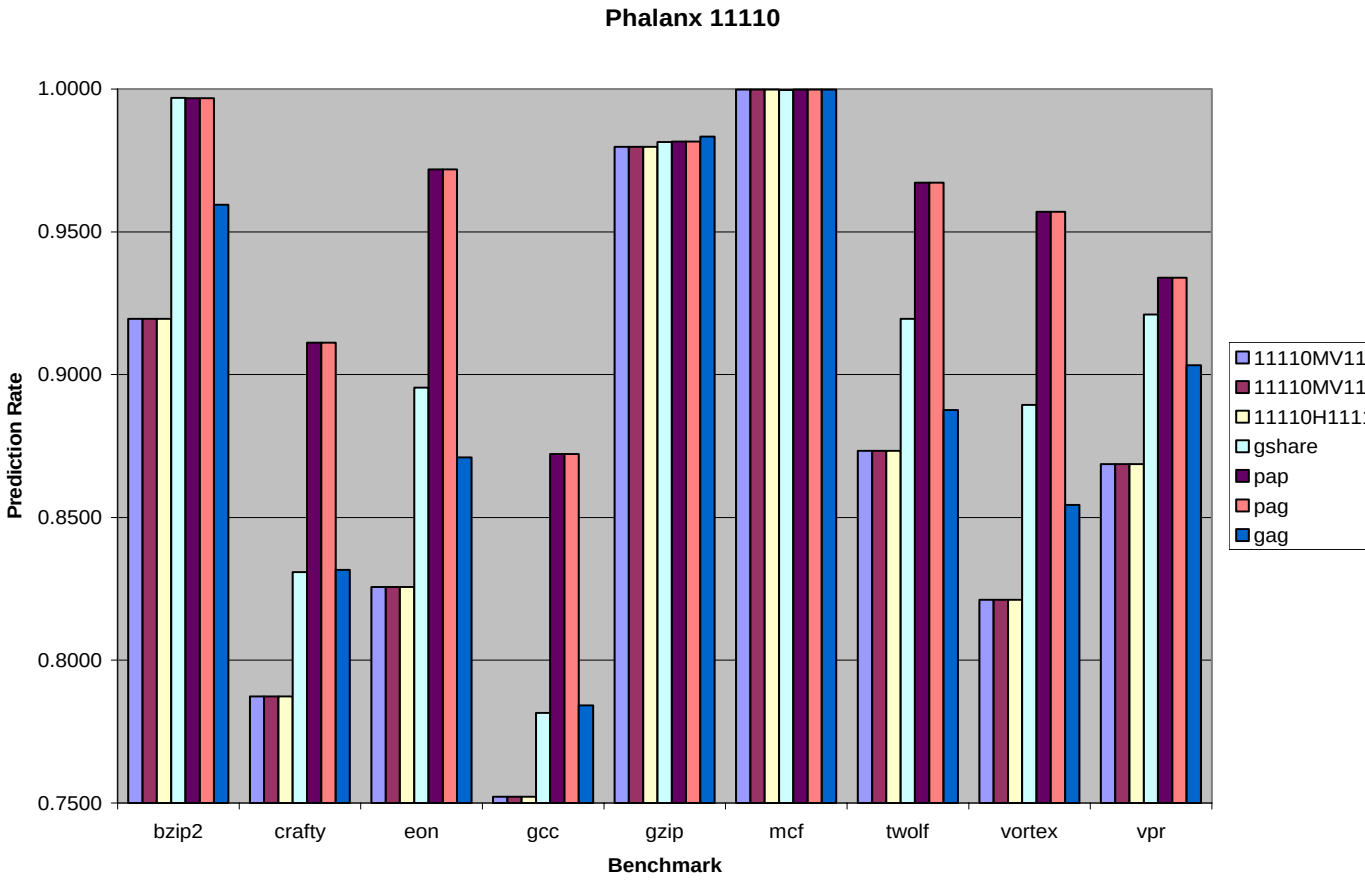

Phalanx 01001
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
01001MV11
01001MV11
01001H111
bimod
pag

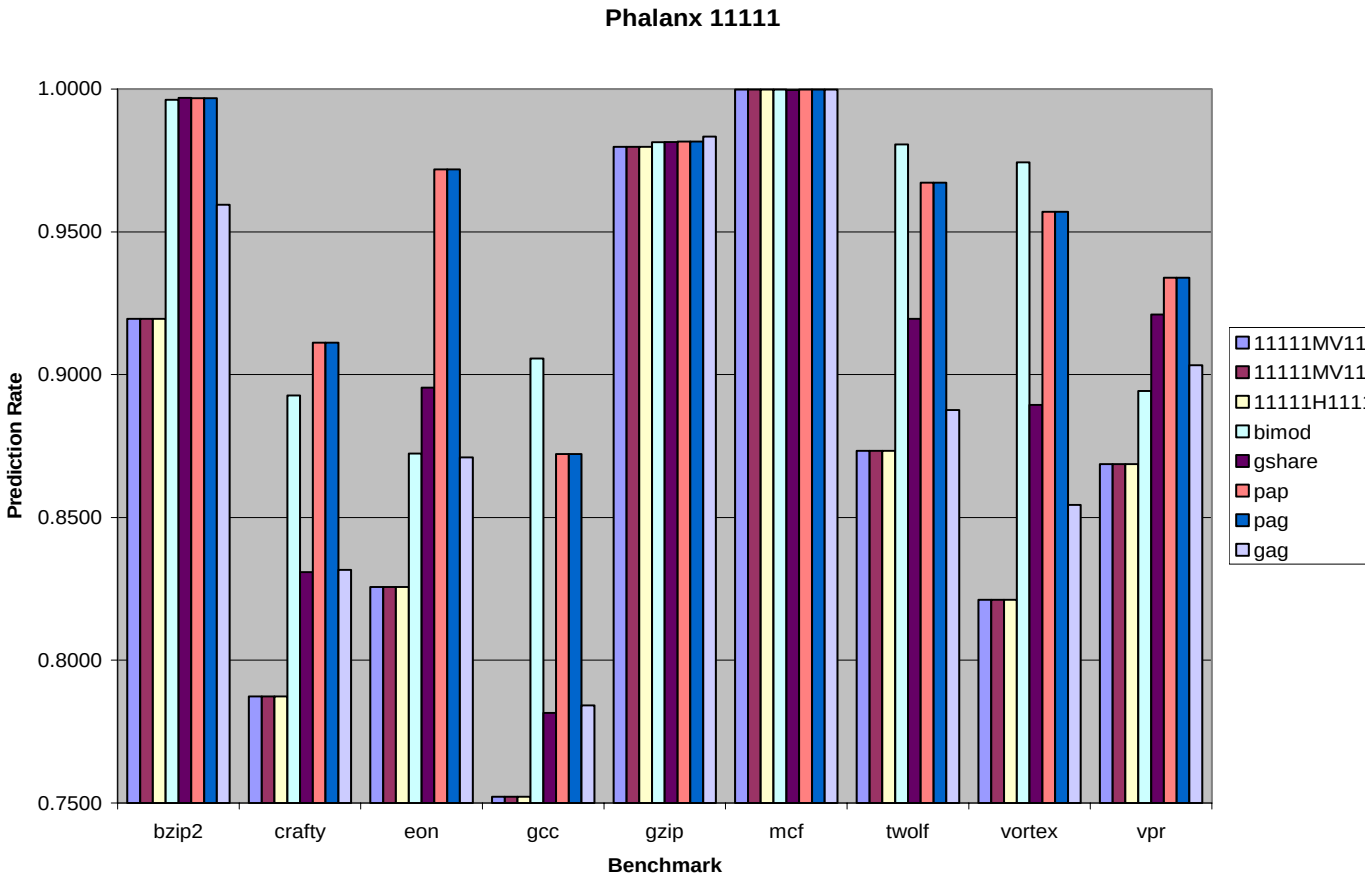

Phalanx 01011
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
01011MV11
01011MV11
01011H111
bimod
gshare
pag


Phalanx 10001
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
10001MV11
10001MV11
10001H111
bimod
gag


Phalanx 10011
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
10011MV11
10011MV11
10011H111
bimod
gshare
gag


Phalanx 11100
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
11100MV11
11100MV11
11100H111
pap
pag
gag


Phalanx 11110
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
11110MV11
11110MV11
11110H111
gshare
pap
pag
gag


Phalanx 11111
1.0000
0.9500
0.9000
0.8500
0.8000
0.7500
Prediction Rate
bzip2
crafty
eon
gcc
gzip
mcf
twolf
vortex
vpr
Benchmark
11111MV11
11111MV11
11111H111
bimod
gshare
pap
pag
gag

	bzip2	crafty	eon	gcc	gzip	mcf	twolf	vortex	vpr
00011MV1111	0.9952	0.7887	0.8265	0.7505	0.9775	0.9997	0.8830	0.8588	0.8805
00011MV1125	0.9952	0.7887	0.8265	0.7505	0.9775	0.9997	0.8830	0.8588	0.8805
00011MV125A	0.9952	0.7887	0.8265	0.7505	0.9775	0.9997	0.8830	0.8588	0.8805
00101MV1111	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
00101MV1125	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
00101MV125A	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
00111MV1111	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
00111MV1125	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
00111MV125A	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
01001MV1111	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01001MV1125	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01001MV125A	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01001MV1111	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01001MV1125	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01001MV125A	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01011MV1111	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01011MV1125	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01011MV125A	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
10001MV1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
10001MV1125	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
10001MV125A	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
10011MV1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
10011MV1125	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
10011MV125A	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11100MV1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11100MV1125	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11100MV125A	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11110MV1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11110MV1125	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11110MV125A	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11111MV1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11111MV1125	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11111MV125A	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
00011H1111	0.9952	0.7887	0.8265	0.7505	0.9775	0.9997	0.8830	0.8588	0.8805
00101H1111	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
00111H1111	0.9948	0.8934	0.9550	0.8458	0.9781	0.9998	0.9541	0.9458	0.9188
01001H1111	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01001H1111	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
01011H1111	0.9948	0.8683	0.9416	0.8137	0.9780	0.9998	0.9437	0.9266	0.9123
10001H1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
10011H1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11100H1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11110H1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
11111H1111	0.9196	0.7873	0.8256	0.7522	0.9798	0.9998	0.8733	0.8212	0.8687
bimod	0.9962	0.8927	0.8724	0.9057	0.9814	0.9998	0.9806	0.9743	0.8943
gshare	0.9969	0.8309	0.8955	0.7816	0.9815	0.9997	0.9196	0.8894	0.9211
pap	0.9968	0.9112	0.9719	0.8722	0.9816	0.9998	0.9672	0.9570	0.9340

pag	0.9968	0.9112	0.9719	0.8722	0.9816	0.9998	0.9672	0.9570	0.9340
gag	0.9595	0.8316	0.8710	0.7842	0.9833	0.9998	0.8876	0.8544	0.9033

CS/ECE 752
Project Report:
Combining Branch Predictors

Dan Gibson
Chuck Tsen
Inge Yuwono

Fall, 2004

Signatures:

Dan Gibson _____

Chuck Tsen _____

Inge Yuwono _____