Branch Prediction Simulation

CS4202 – Practical 1

Jamie Munro, Student Number: 210027910

Abstract

This document discusses the results of a branch prediction simulation experiment. The simulator was implemented in C++ and supports 6 different prediction strategies. The results gathered are analysed and discussed in order to assess the effectiveness of each strategy.

Introduction

Most modern CPUs utilise the so-called "5-stage pipeline" – splitting each instruction into 5 stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), Register Write Back (WB) [1]. The use of this pipeline enables Instruction Level Parallelism (ILP) by performing each of these stages simultaneously on different instructions; i.e. whilst Instruction_i is in the WB-stage, Instruction_{i-1} is in the MEM-stage, Instruction_{i-2} is in the EX-stage and so on. This can offer dramatic performance improvements.

Branch instructions (which can be thought of as low-level if statements) introduce a challenge for the 5-stage pipeline. When a branch instruction is encountered, the CPU must either set the Program Counter (PC) to the target of the branch instruction if the branch is **taken**, or must increment the PC to the next value if the branch is **not taken.** However, the CPU does not know if the branch has been taken or not until the EX-stage, which leads to the potential that the IF-stage fetches the wrong instruction. The simplest way to deal with this is to stall the pipeline until the outcome of the branch is known, but this wastes valuable cycles.

Although many solutions to this problem have been proposed, this document concerns itself with the "branch prediction" method. Under this method, the CPU attempts to guess the outcome of the branch – taken or not taken. It then continues to process the pipeline under the assumption that it's guess was correct, i.e. it fetches the instruction at the branch target if it predicted taken or it fetches the instruction at PC+1 if it predicts not taken. Once the outcome of the branch is known, if the prediction was correct, execution continues as normal. If the prediction was wrong, the CPU incurs a penalty as it has to stop execution and go back to fetch the correct instruction. The performance of such an approach is heavily dependent on the prediction strategy used.

This document discusses 7 predictions strategies: Always Taken, 2-bit (Bimodal), Global History with Index Sharing (Gshare) and 3 different profile-based strategies. These strategies, as well as their implementation details in the simulator, will be discussed later in this document.

The remainder of this document contains a brief discussion on the simulator, an outline of the prediction strategies as well as any relevant implementation details, details on the methodology of the experiments, a presentation of the results, an evaluation of the prediction strategies and finally a conclusion answering the research questions posed in the methodology section. The appendix of this document contains the raw results from the experiments.

Simulator

The simulator on which the experiments in this document were preformed is a C++ program. The program operates on branch trace files which are text documents where each line contains two values separated by a space. The first value is a branch address, and the second value is bit indicating if the branch was taken or not taken. These trace files are considered the ground truth during the simulation, and the predictions of the various strategies are compared against this ground truth. The author had access to 30 branch traces. A python script was also developed to run the simulator using multiple strategies across multiple trace files as a batch and generate a CSV of results.

The simulator follows this procedure:

1. Collect trace file path and prediction strategy (as well as any other options) from the user in the form of command line arguments and display an error and usage message if these are not provided correctly.
2. Load the trace file into a BranchTrace data structure that allows accessing the branch address and branch outcome as well as comparison between BranchTraces. This initial BranchTrace will be referred to as the "ground truth".
3. Instantiate a predictor object of the selected prediction strategy.
4. Feed the branch addresses from the ground truth one by one into the predictor object. Once the predictor object has made a prediction it is recorded in a second BranchTrace object and the predictor is then told the actual outcome of the branch.
5. Once all instructions have been passed into the predictor, the prediction BranchTrace is compared against the ground truth BranchTrace and the results are printed to the console.
6. Any options (such as print to file) are carried out.

The complete source code for the simulator, as well as the trace files used can be viewed here: https://git.jbm.fyi/jbm/Branch_Prediction_Simulator.

Prediction Strategies

Always Taken

Always taken is the simplest of the prediction strategies. Under this strategy, the CPU will always predict that a branch is taken, without taking into account any dynamic information. This will mean that on average, the prediction will be correct around half the time, although this statistic can be improved if the compiler is aware of this strategy and optimises the code to take advantage of it.

2-bit (Bimodal)

The 2-bit strategy responds dynamically to the outcomes of previous branches. Under this strategy, the CPU maintains a prediction table of size n. The table is indexed by the last $\log_2(n)$ bits of the branch instruction address, e.g. if the table has 512 entries, then the last 9-bits of the branch address will be used to index the table. Each entry is a 2-bit counter. Every time a branch address is taken, its respective counter will be incremented and every time it is not taken, the counter will be decremented. The predictor then uses the upper bit of the relevant counter to make its prediction:

Decimal = 0, Binary = 00, **Predict: not taken**

Decimal = 1, Binary = 01, **Predict: not taken**

Decimal = 2, Binary = 10, **Predict: taken**

Decimal = 3, Binary = 11, **Predict: taken**

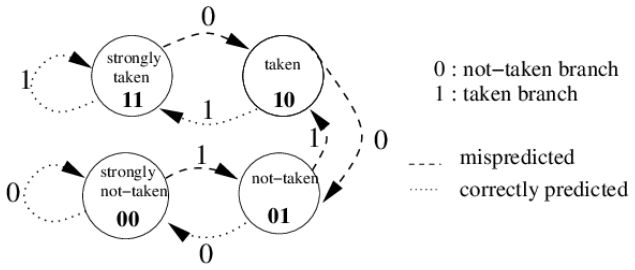


Figure 1 State diagram of 2-bit predictor [2]

The upper bit of the counter is known as the prediction bit, and the lower bit as the conviction or hysteresis bit. This 2-bit scheme means that the predictor needs to be wrong twice in a row in order to change its prediction, which helps to stabilise the strategy when compared to a 1-bit counter.

As only a subset of the address is used to index the table, there will be multiple address that map to the same entry in the table. In the simulator, this prediction table has been implemented as an array of chars. Every time a value *v* in the table is updated, it is constrained such that *v* >= 0 and *v* <= 3.

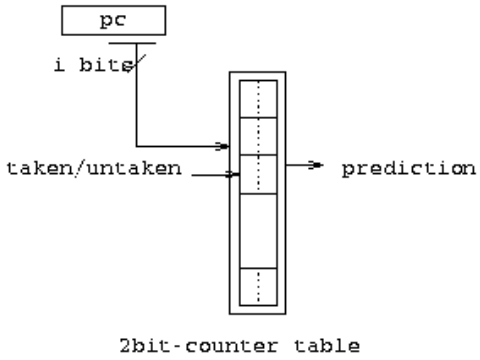


Figure 2 2-bit predictor architecture diagram [3]

GShare

The GShare strategy is similar to the 2-bit strategy but also takes into account that previous branches may affect the outcome of future branches, **even if those branches do not share an address**. Gshare maintains the same *n*-sized prediction table with 2-bit counters as the 2-bit strategy, but also maintains a global history register (GHR). This register is shifted left each time a branch occurs, and the last bit of the GHR is set to 1 if the branch was taken and 0 if it was not. The strategy then proceeds in the same way as the 2-bit strategy, only the table is indexed by the last *n*-bits of the branch address XOR the last *m*-bits of the GHR. Note that *n* and *m* do not need to be equal, but in these experiments they will be.

In the simulator, the prediction table is implemented the same as in the 2-bit strategy. The GHR is implemented as a class with a *history()* and *update(bool taken)* method. The class contains an unsigned long representing the register. The history method simply returns this long, whilst the update function uses bitwise operations to shift the register to the left and set the final bit to the value of the taken Boolean variable.

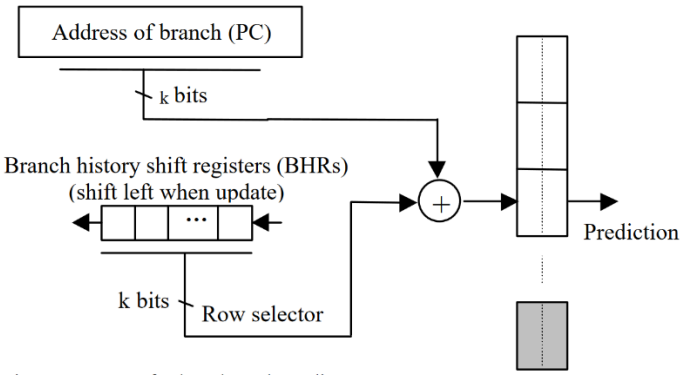


Figure 3 Gshare architecture diagram [4]

Profile-based approaches

Profile-based approaches allow the predictor to view the code in a "profile" run before the "real" run. The predictor must come up with some way of representing the insights of the profile run which can be utilised in the real run. Three different profile strategies were experimented with.

Section Profile

In the "section profile" strategy, the predictor maintains an *n*-entry Boolean prediction table. During the profile run, the branch trace is split into *n* equal (or nearly equal if the total instructions are not divisible by n) sections. For each section a "mini-profile" is generated which records the total number of branch instructions, the number of taken branches and the number of not taken branches. Each mini-profile is mapped to an entry in the prediction table which is set to 0 if there were more not taken branches then taken branches, and 1 if there were more taken branches.

During the real run, the prediction strategy then acts similarly to the always taken strategy, only it changes between always taken and always not taken based on which outcome is more likely for the current section. The outcome to use is determined by using the instruction index / (total instructions / *n*) as an index for the prediction table.

In the simulator, this was implemented using a dedicated Profiler class which traverses the ground truth table and generates *n* mini-profiles. These mini-profiles are then passed to the SectionProfile class which populates a Boolean prediction table with the values determined by comparing the taken to not taken values from each profile. Due to poor performance, and the reasons discussed in the next paragraph, this strategy was removed from the simulator, however the code remains in the repository.

This strategy is unlikely to be easy to implement in real hardware, as it requires the predictor to be aware of the instruction index of the current instruction within the program, instead of the branch address as the other predictors use.

Predetermined

In the "predetermined" strategy, the predictor maintains also an *n*-entry Boolean prediction table. The prediction table is addressed in the same way as in the 2-bit strategy, using the last log2(*n*) bits of the branch address as the index for the prediction table.

However, unlike in the 2-bit strategy, the table is prepopulated with either 0 or 1, based on which outcome is more plentiful for each address, during the profile run of the code. During the real run of the code, the relevant prediction is retrieved from the prediction table and no updates occur during the runtime. The lack of an update step means that in theory, there will be less runtime overhead.

However, this fact is of course counter-balanced by the need to run a profile of the code prior to real execution of the code.

In the simulator, this strategy is implemented by creating an integer table of counters, using the same addressing scheme as the Boolean table discussed above. The ground truth is then iterated through incrementing the relevant counter for each branch that has a taken outcome, and decrementing for each branch that is not taken. Once this is done, the integer table is transferred over to the Boolean table by setting the relevant address to 0 if the corresponding integer value is negative and 1 if the corresponding integer value is positive.

If implemented in real hardware, the same method as described above could be used, storing the integer values either in the CPU cache or in main memory. If the performance of the profile run is not relevant, there is no need for dedicated hardware to perform profile step of the strategy. A strategy similar to this could be useful if the runtime performance of the code is vital, but there is time to prepare a profile prior to the real run.

Predetermined Global

The "Predetermined Global" strategy is very similar to the Predetermined strategy, only it also makes use of the GHR as in the Gshare strategy. This means that the strategy incorporates some dynamic information as well as making use of the predetermined static values.

The profile run of the code is performed in the same way as the Predetermined strategy, but also tracks the state of the GHR during the profile. The prediction table is indexed using the last *n*-bits of the branch address XOR the last *m*-bits of the GHR. The real run of the strategy is also performed in the same way as the Predetermined strategy, but the predictor must also update the GHR in real time, which reintroduces some of the overhead done away with in the Predetermined strategy.

The implementation in the simulator and in real hardware would be done in the same way as described for the Predetermined strategy but also incorporating the GHR.

Methodology

Performance statistics for each of the strategies across each of the 30 trace files were gathered by running the simulation across each combination of prediction strategy and trace file. This process was automated using a python script, resulting in a single CSV of data. This raw data can be seen in the appendix of this document.

With this data gathered two research questions were formulated:

- **Question 1:** Which prediction strategies yield the best performance?
- **Question 2:** What is the effect of table size on performance?

These questions were answered using statistical analysis of the results and these results are presented in a number of graphs, shown in the results section below.

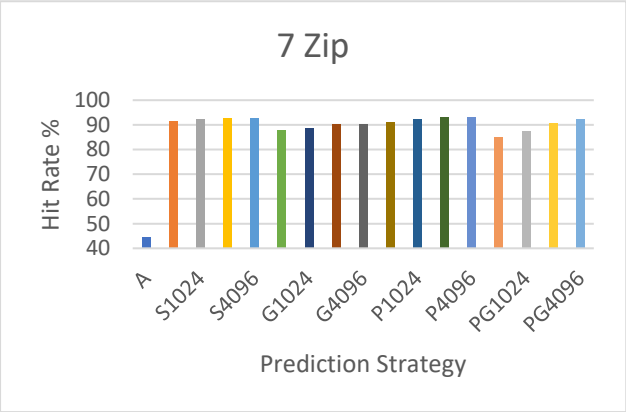
Results

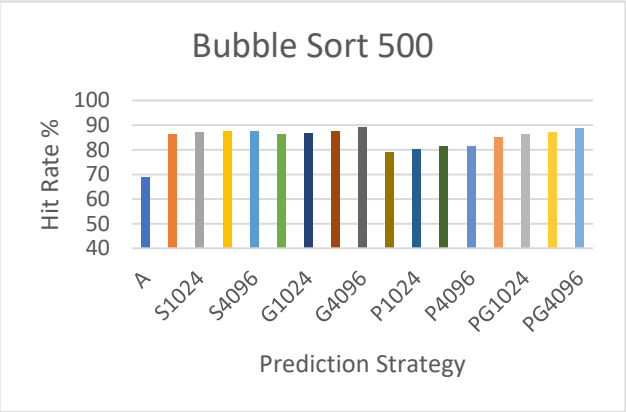
On the following charts, the strategy names have been abbreviated. See the table below for a description of what each of the abbreviations stand for, as well as the mean, median and range of each of the strategies (these will be plotted later).

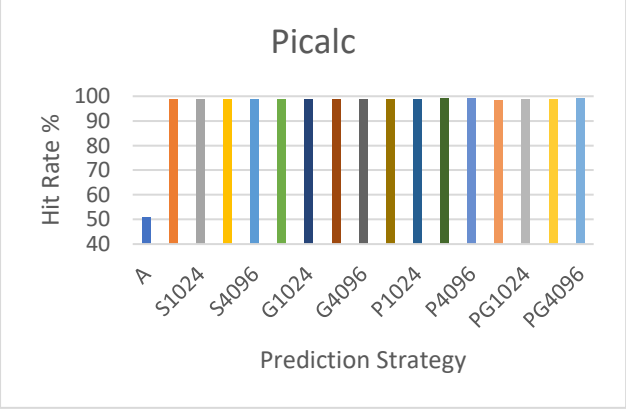
Table 1

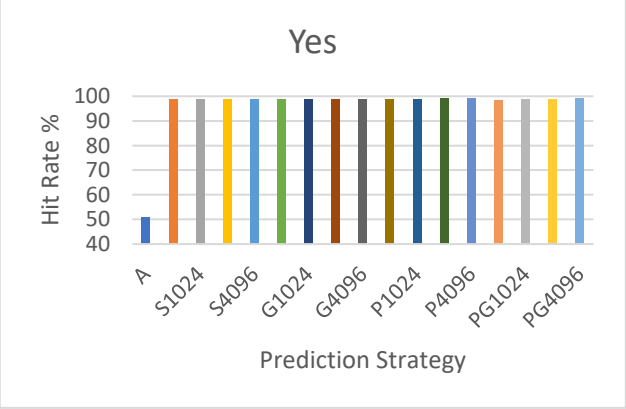
Strategy	Description	MEAN	MEDIAN	RANGE
A	Always Taken	55.93027667	52.30975	38.4204
S512	2-Bit with table size 512	90.57706667	89.42695	20.925
S1024	2-Bit with table size 1024	91.54628	91.31485	20.925
S2048	2-Bit with table size 2048	92.11433333	92.05795	20.925
S4096	2-Bit with table size 4096	92.35294	92.60495	20.925
G512	Gshare with table size 512	89.48314	87.81435	16.1721
G1024	Gshare with table size 1024	90.81912667	88.5667	14.0014
G2048	Gshare with table size 2048	91.82763667	90.29995	12.983
G4096	Gshare with table size 4096	92.37427	91.5656	12.8358
P512	Predetermined with table size 512	88.97668667	88.90865	23.5955
P1024	Predetermined with table size 1024	90.93857333	90.8176	19.0689
P2048	Predetermined with table size 2048	92.28933333	92.3603	18.1857
P4096	Predetermined with table size 4096	93.08119	93.08655	18.0499
PG512	Predetermined Global with table size 512	86.02312333	84.9818	30.7332
PG1024	Predetermined Global with table size 1024	88.73319333	88.1564	27.0943
PG2048	Predetermined Global with table size 2048	91.04037667	91.5198	22.8168
PG4096	Predetermined Global with table size 4096	92.69009667	93.04445	18.9531

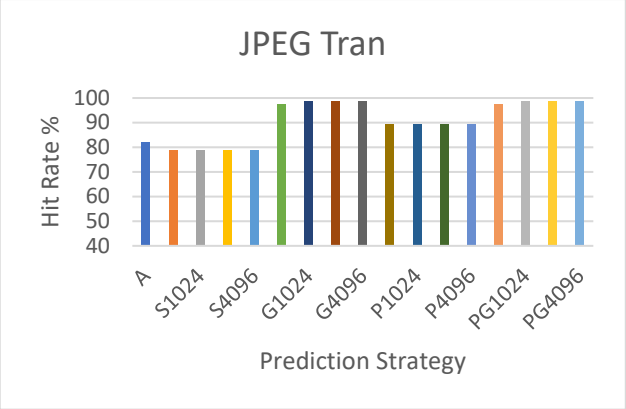
Six of the sample branch traces (7Zip, Bubble Sort 500, Picalc, Yes, JPEG Tran and Google Chrome) were selected to compare the performance of each strategy on that trace:

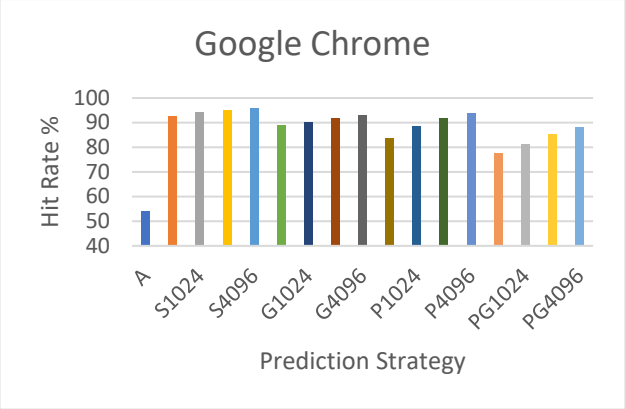






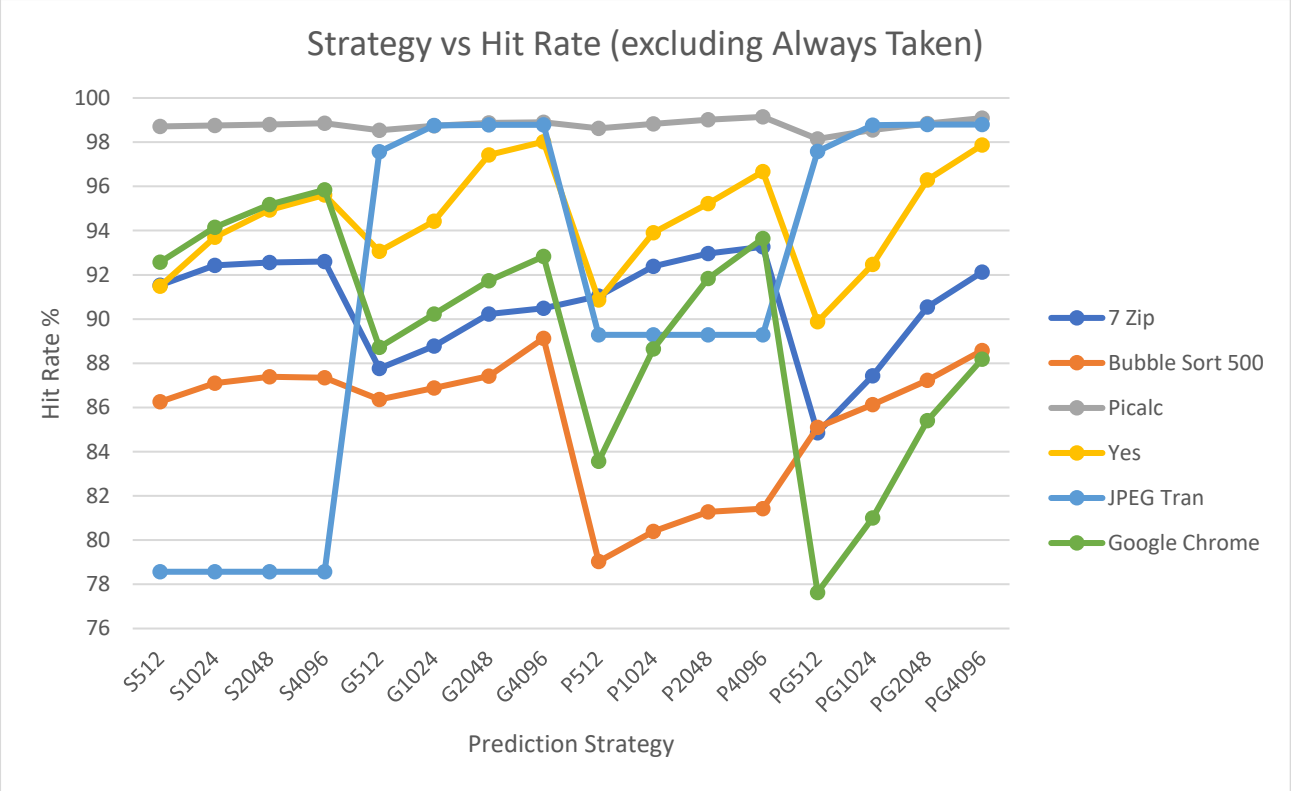






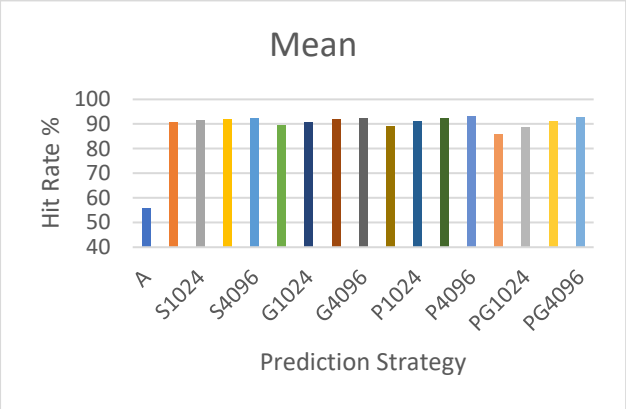
Graphs 1-6

Next all six of these traces are plotted on the same line graph with the A strategy excluded to offer a higher resolution look at the performance differences between the more complex strategies.

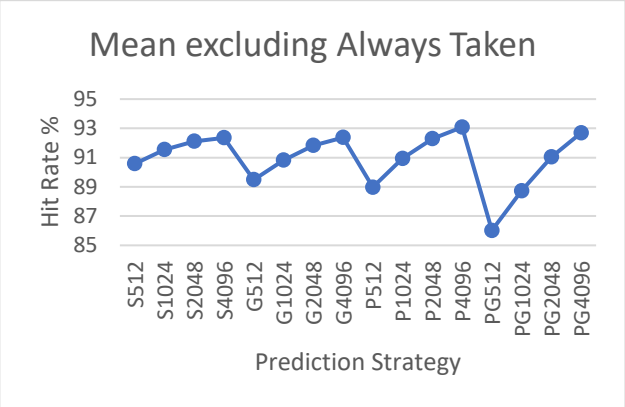


Graph 7

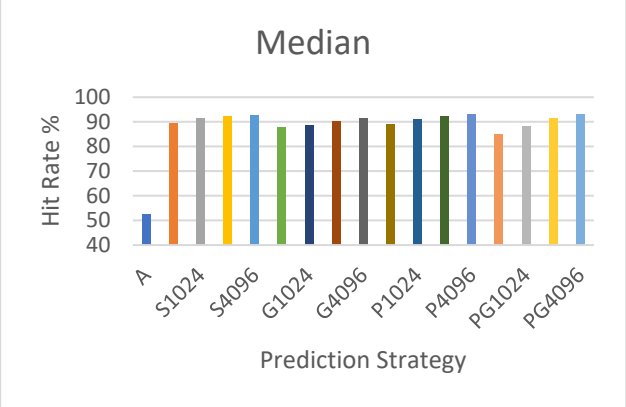
The next graphs show the mean, median and range of each of the strategies across all 30 sample branch traces plotted as bar charts and line graphs. The line graphs on the right exclude the A strategy to offer a higher resolution look at the more complex strategies.



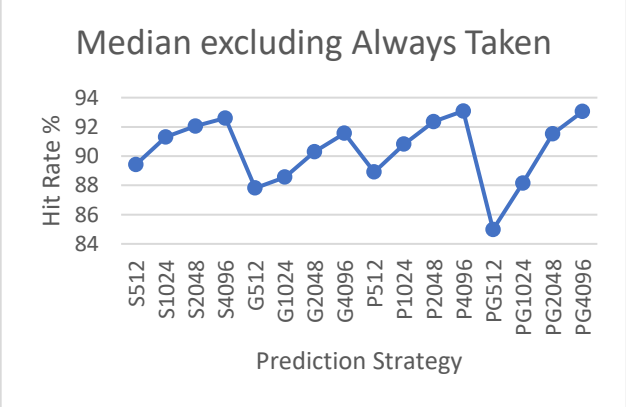
Graph 8



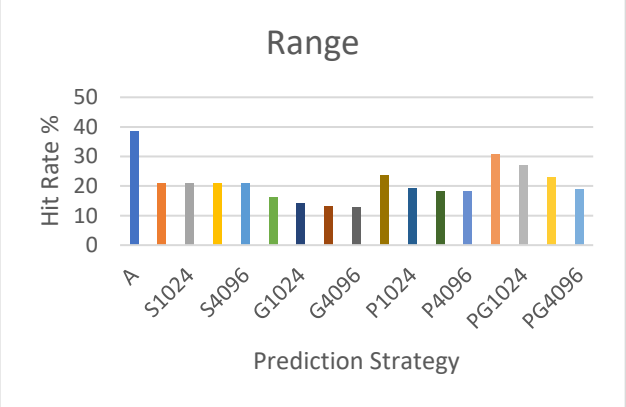
Graph 9



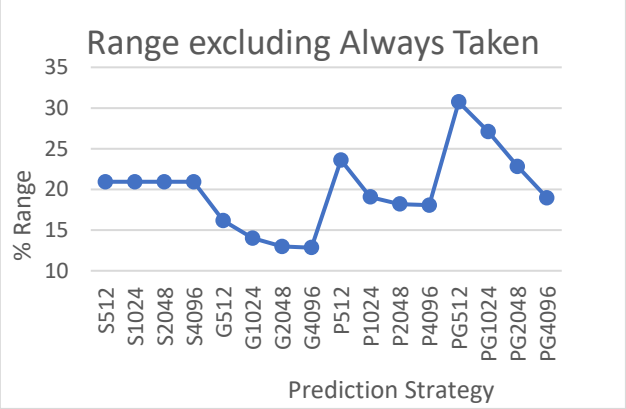
Graph 10



Graph 11

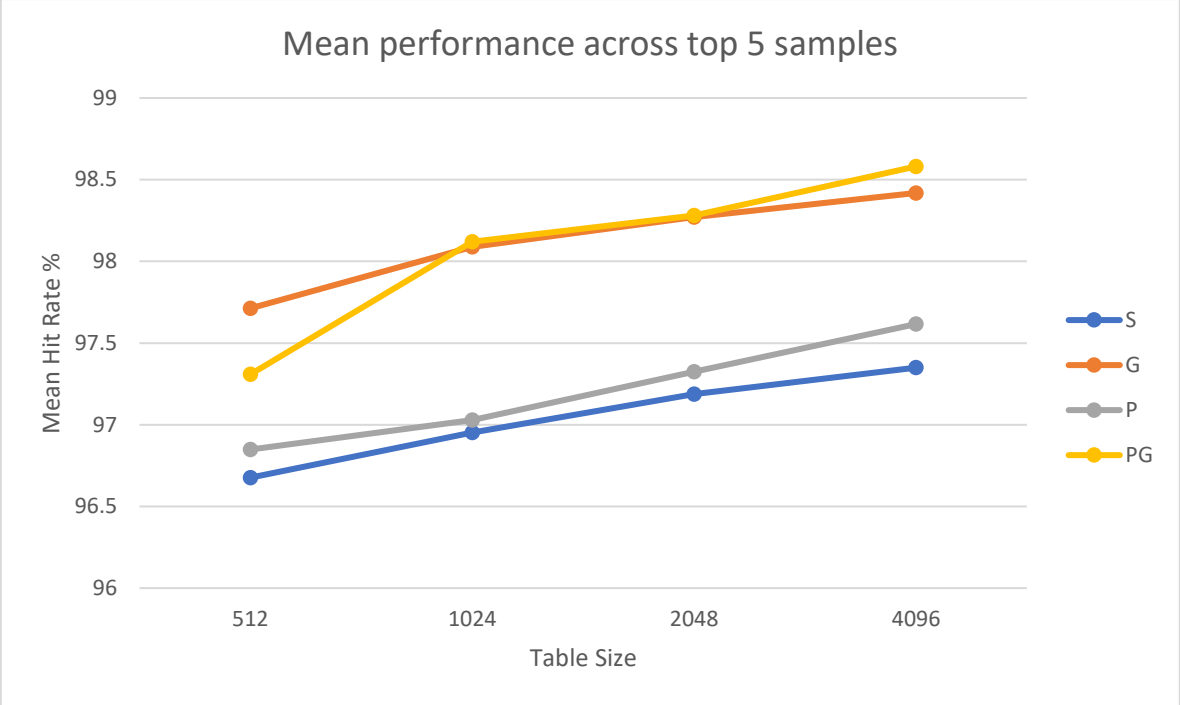


Graph 12

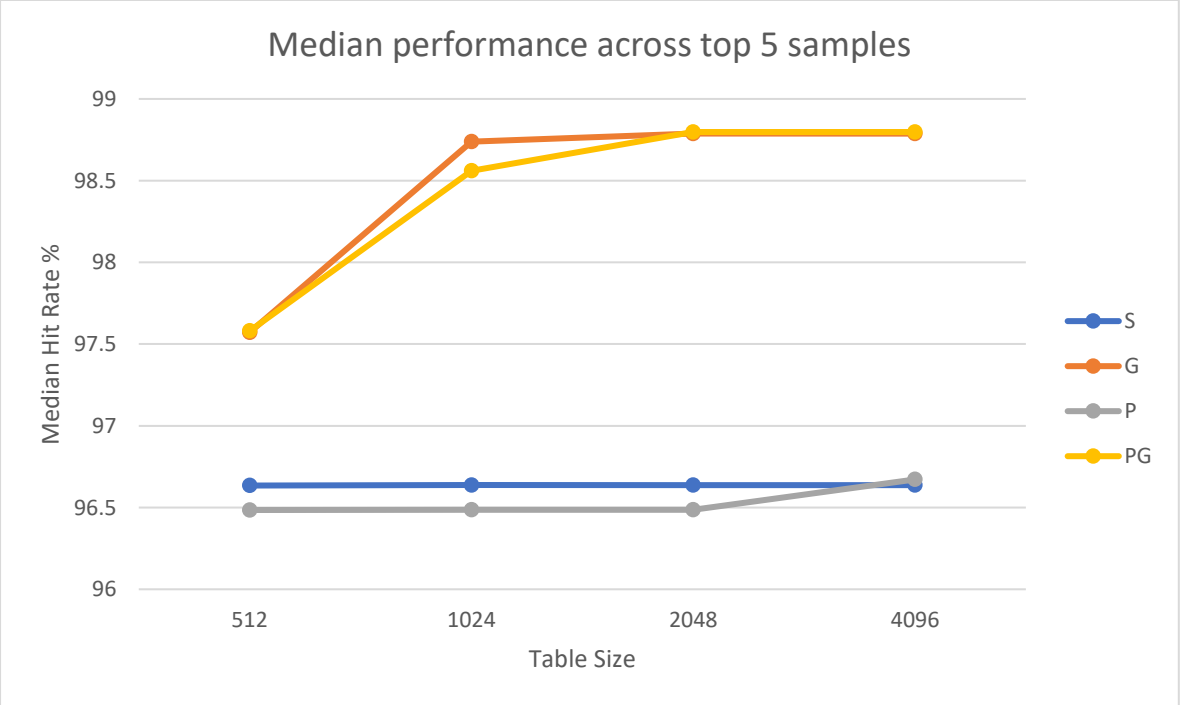


Graph 13

Next, the mean and median performance of the more complex strategies is shown across the top five performing samples for the respective strategy. This allows us to see the performance of the strategies in their best (or near best) case scenarios, and allows a tertiary look at performance across table size.

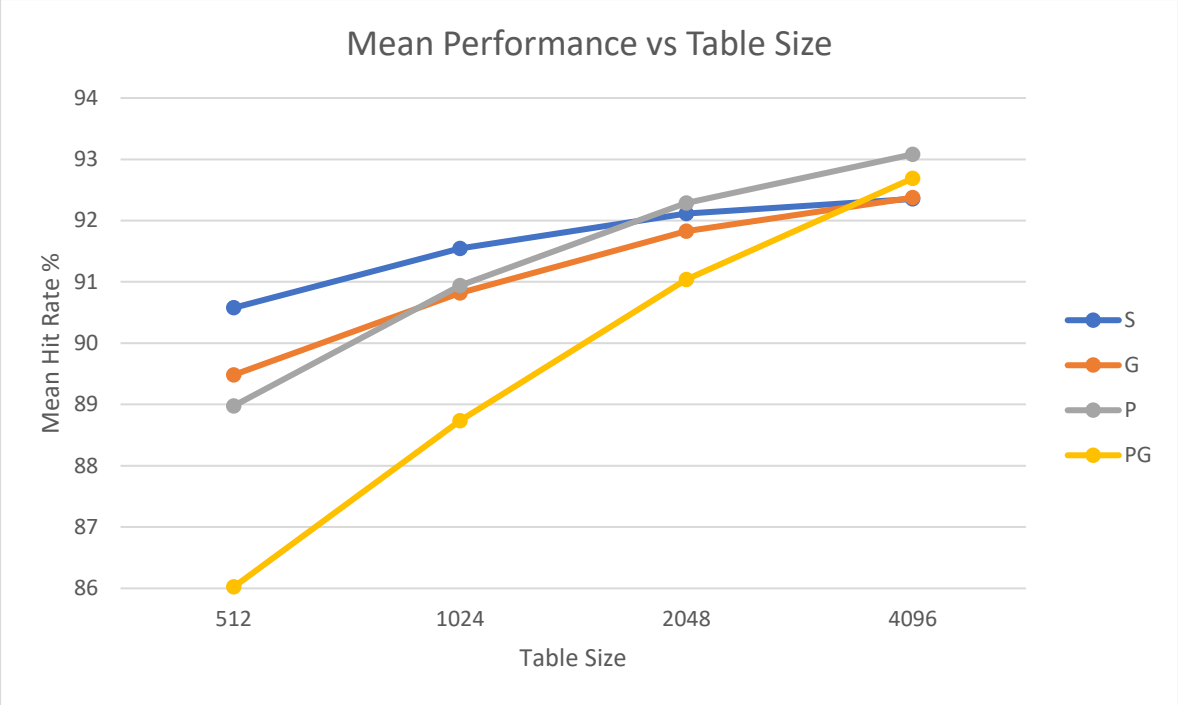


Graph 14

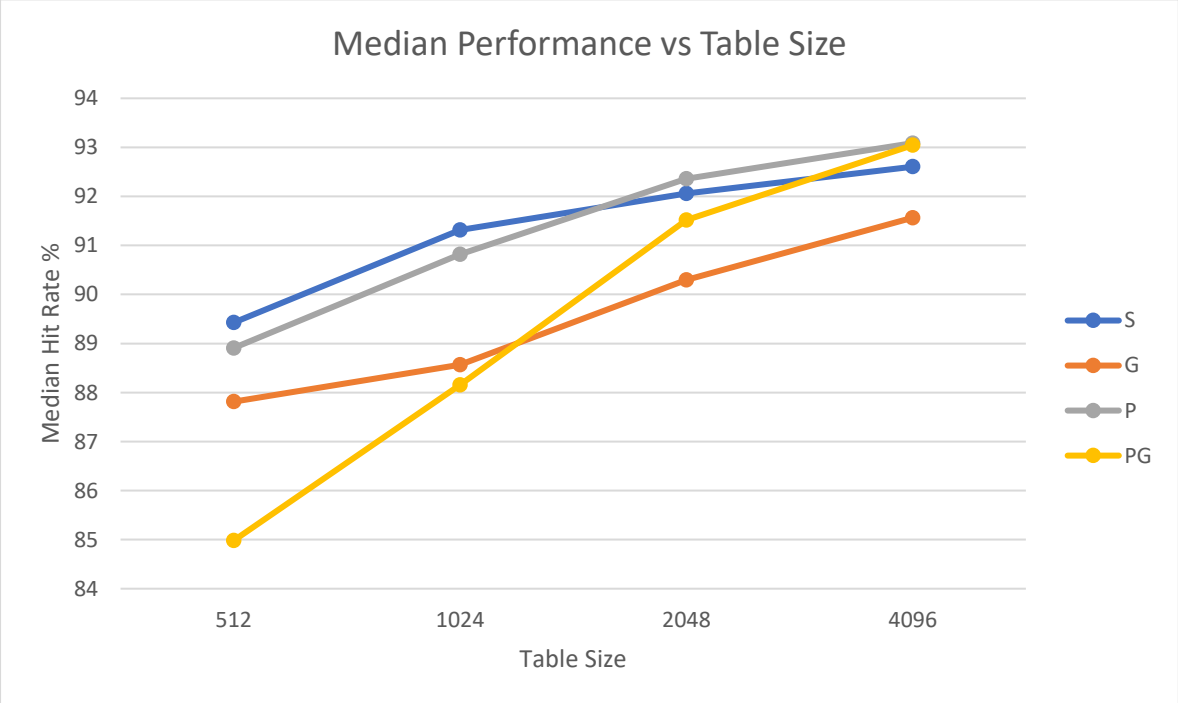


Graph 15

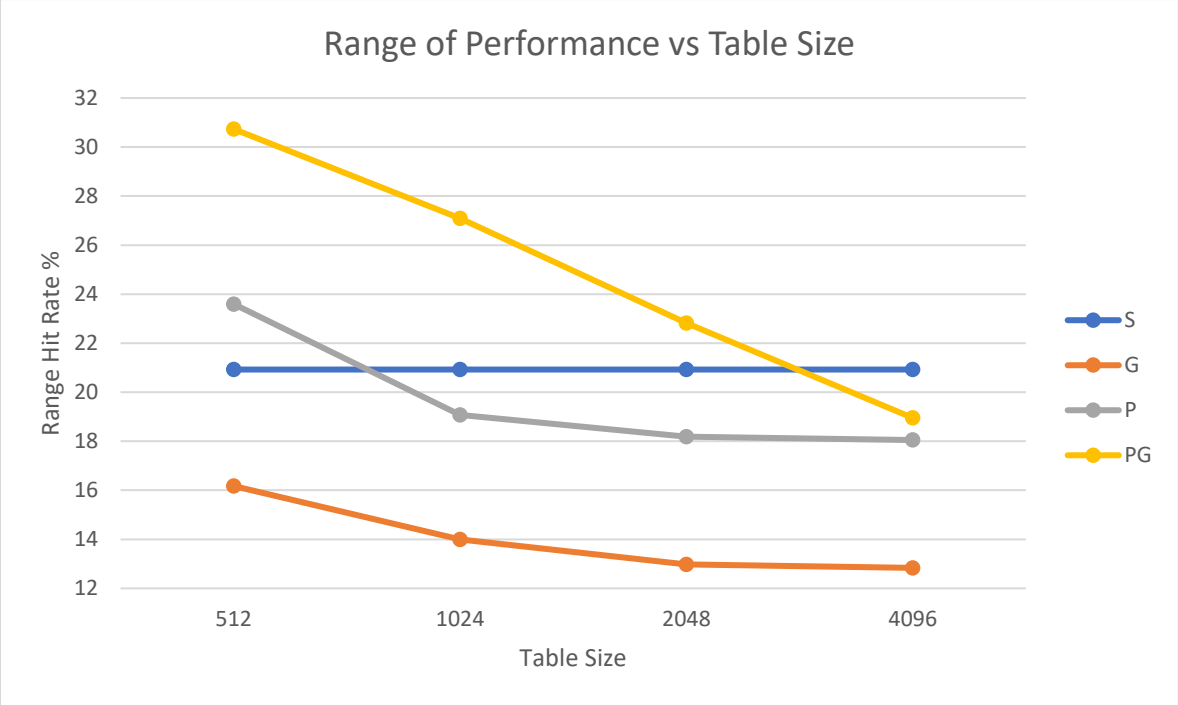
Finally, the impact of table size of on performance is directly analysed. The charts below show the performance in terms of mean hit rate, median hit rate and hit rate range for each of the strategies across all of the samples vs table size.



Graph 16



Graph 17



Graph 18

Evaluation

By looking at the first six graphs, which compare the performance of the different strategies on the same branch trace, we can see that the Always Taken strategy consistently performs much worse than the dynamic strategies. This is not surprising as dynamic strategies generally perform better then static strategies as they make use of more up to date information. However, it should be noted that the compiler is unlikely to have optimised for the Always Taken strategy in this case and therefore it is likely that better performance could have been achieved had the compiler been aware that this strategy was in use.

Graph 7 shows the same data at the first 6 graphs but on the same chart and excluding the Always Taken strategy. This gives us a higher resolution look at the dynamic strategies. This chart shows that the performance of all the strategies is highly correlated with the sample itself, i.e. if one strategy performs well it is likely that all strategies will perform well. This is clearest on the Picalc series, but one can note that all the other series follow roughly the same pattern. This suggests that the biggest impact on performance is not the prediction strategy used, but the predictability of the branches within a sample. This may lead to average metrics, such as median and mean being misleading when applied across the whole sample set.

Graphs 8 – 11 shows the average performance of each strategy across the entire sample set. Charts 9 and 11 clearly show that performance improves as the table size increases, which is again unsurprising as these strategies have access to more information. The best overall performance across all the samples seems to be achieved by the Predetermined Strategy with a table size of 4096. However, it has only narrowly beaten the standard 2-bit strategy and the Gshare strategy with the same table sizes, neither of which require a profile run of the code.

When looking at graphs 12 and 13, which shows the range in performance achieved by the various strategies, lower values indicate a more consistent performance. Here, Gshare is the clear winner with none of the other strategies able to offer more or even close to the consistency of Gshare.

Graphs 14 and 15 show the mean and median performances respectively of the various dynamic strategies across the top 5 samples for each respective strategy. This allows us to observe the strategies in their best-case scenarios across the sample set. Here we can see that the Standard 2-bit strategy and the Predetermined strategy offer similar performance to each other, whilst the Gshare and Predetermined Global strategies offer similar performance to each other. This is not surprising as these pairs of strategies are similar to each other. Generally, the Predetermined strategies offer slightly higher performance, however this is counter-balanced by their need to perform a profile run of the code before the real execution can begin. Gshare is able to show a performance advantage against Predetermined Global when the table size is 512 and when looking median graph, the Standard 2-bit prediction beats the Predetermined strategy for all table sizes lower than 4096. These facts further show that the Predetermined strategies are unlikely to be worthwhile.

Graphs 16 – 18 show a clear trend of improved performance and consistency as table sizes increase. At lower table sizes it seems that the standard 2-bit strategy has the best performance whilst at larger table sizes the Predetermined strategies begin to show better performance. At higher table sizes Gshare and the Standard 2-bit strategies offer similar performance to each other, but Gshare is substantially more consistent.

Conclusion

The results gathered in this project are able to offer a clear answer to question 2: increased table sizes offer increased performance. It would be interesting to repeat the experiments for more table sizes to see if the performance continues to increase with table size, or if it begins to plateau.

The answer to question 1 is less clear cut. The Gshare strategy is able to offer better performance than the Standard 2-bit strategy when only looking at the strategies in their best-case scenarios, but seem to offer similar performances across the entire sample set. Gshare is also is also able to offer a more consistent performance, so it's probably fair to say that Gshare outperforms the Standard 2-bit strategy and does so without incurring significantly higher overhead. It would be interesting to repeat this experiment across more samples to see if these trends hold out.

The Predetermined Global strategy was able to offer similar or better performance to the Gshare strategy, but only at larger table sizes. As the Predetermined Global strategy requires a profile run of the code before any "real" execution can begin, this overhead is unlikely to be worthwhile for many applications. However, if there is time to prepare a profile but real execution must be highly optimised, it may make sense to use this strategy in some settings.

References

[1] J. L. Hennessy, David A. Patterson 2012: "Computer Architecture: A Quantitative Approach", 5th Edition, Appendix C.
[2] C. Maiza, C. Rochange, 2006: "History-based Schemes and Implicit Path Enumeration".
[3] Z. Su, M. Zhou, 1995: "A Comparative Analysis of Branch Prediction Schemes".
[4] N. Ismail, 2002: "Performance study of dynamic branch prediction schemes for modern ILP processors".

Appendix: Raw Data

Table 2

Trace File	Strategy	Total	Hits	Misses	Hit %	Miss %
7zz	A	50000	22334	27666	44.668	55.332
7zz	S512	50000	45765	4235	91.53	8.47
7zz	S1024	50000	46211	3789	92.422	7.578
7zz	S2048	50000	46276	3724	92.552	7.448
7zz	S4096	50000	46300	3700	92.6	7.4
7zz	G512	50000	43883	6117	87.766	12.234
7zz	G1024	50000	44387	5613	88.774	11.226
7zz	G2048	50000	45113	4887	90.226	9.774
7zz	G4096	50000	45241	4759	90.482	9.518
7zz	P512	50000	45516	4484	91.032	8.968
7zz	P1024	50000	46195	3805	92.39	7.61
7zz	P2048	50000	46481	3519	92.962	7.038
7zz	P4096	50000	46635	3365	93.27	6.73
7zz	PG512	50000	42429	7571	84.858	15.142
7zz	PG1024	50000	43719	6281	87.438	12.562
7zz	PG2048	50000	45275	4725	90.55	9.45
7zz	PG4096	50000	46059	3941	92.118	7.882
Echo	A	48361	24740	23621	51.1569	48.8431
Echo	S512	48361	42946	5415	88.803	11.197
Echo	S1024	48361	43242	5119	89.415	10.585
Echo	S2048	48361	43360	5001	89.659	10.341
Echo	S4096	48361	43392	4969	89.7252	10.2748
Echo	G512	48361	42133	6228	87.1219	12.8781
Echo	G1024	48361	42367	5994	87.6057	12.3943
Echo	G2048	48361	42499	5862	87.8787	12.1213
Echo	G4096	48361	42430	5931	87.736	12.264
Echo	P512	48361	42637	5724	88.164	11.836
Echo	P1024	48361	43839	4522	90.6495	9.35051
Echo	P2048	48361	44467	3894	91.9481	8.05194
Echo	P4096	48361	44786	3575	92.6077	7.39232
Echo	PG512	48361	40003	8358	82.7175	17.2825
Echo	PG1024	48361	41745	6616	86.3196	13.6804
Echo	PG2048	48361	42920	5441	88.7492	11.2508
Echo	PG4096	48361	44028	4333	91.0403	8.9597
LoopCondition	A	66006	35282	30724	53.4527	46.5473
LoopCondition	S512	66006	58170	7836	88.1283	11.8716
LoopCondition	S1024	66006	59505	6501	90.1509	9.8491
LoopCondition	S2048	66006	59900	6106	90.7493	9.25067
LoopCondition	S4096	66006	59766	6240	90.5463	9.45369
LoopCondition	G512	66006	58646	7360	88.8495	11.1505

LoopCondition	G1024	66006	59256	6750	89.7737	10.2263
LoopCondition	G2048	66006	59569	6437	90.2479	9.75214
LoopCondition	G4096	66006	59774	6232	90.5584	9.44157
LoopCondition	P512	66006	56542	9464	85.6619	14.3381
LoopCondition	P1024	66006	59290	6716	89.8252	10.1748
LoopCondition	P2048	66006	60915	5091	92.2871	7.71293
LoopCondition	P4096	66006	61385	4621	92.9991	7.00088
LoopCondition	PG512	66006	55483	10523	84.0575	15.9425
LoopCondition	PG1024	66006	57971	8035	87.8269	12.1731
LoopCondition	PG2048	66006	59592	6414	90.2827	9.7173
LoopCondition	PG4096	66006	61089	4917	92.5507	7.44932
MonteCarlo20000	A	321653	263479	58174	81.914	18.0859
MonteCarlo20000	S512	321653	283650	38003	88.1851	11.8149
MonteCarlo20000	S1024	321653	283996	37657	88.2927	11.7073
MonteCarlo20000	S2048	321653	284076	37577	88.3175	11.6825
MonteCarlo20000	S4096	321653	284090	37563	88.3219	11.6781
MonteCarlo20000	G512	321653	282613	39040	87.8627	12.1373
MonteCarlo20000	G1024	321653	282782	38871	87.9152	12.0848
MonteCarlo20000	G2048	321653	282591	39062	87.8559	12.1441
MonteCarlo20000	G4096	321653	284399	37254	88.418	11.582
MonteCarlo20000	P512	321653	287536	34117	89.3932	10.6068
MonteCarlo20000	P1024	321653	288678	32975	89.7483	10.2517
MonteCarlo20000	P2048	321653	289228	32425	89.9193	10.0807
MonteCarlo20000	P4096	321653	289478	32175	89.997	10.003
MonteCarlo20000	PG512	321653	279398	42255	86.8632	13.1368
MonteCarlo20000	PG1024	321653	281633	40020	87.558	12.442
MonteCarlo20000	PG2048	321653	283420	38233	88.1136	11.8864
MonteCarlo20000	PG4096	321653	287241	34412	89.3015	10.6985
ack	A	50000	33552	16448	67.104	32.896
ack	S512	50000	46716	3284	93.432	6.568
ack	S1024	50000	46796	3204	93.592	6.408
ack	S2048	50000	46909	3091	93.818	6.182
ack	S4096	50000	46930	3070	93.86	6.14
ack	G512	50000	46047	3953	92.094	7.906
ack	G1024	50000	46241	3759	92.482	7.518
ack	G2048	50000	46665	3335	93.33	6.67
ack	G4096	50000	46686	3314	93.372	6.628
ack	P512	50000	44168	5832	88.336	11.664
ack	P1024	50000	44709	5291	89.418	10.582
ack	P2048	50000	44939	5061	89.878	10.122
ack	P4096	50000	45062	4938	90.124	9.876
ack	PG512	50000	46084	3916	92.168	7.832
ack	PG1024	50000	46756	3244	93.512	6.488
ack	PG2048	50000	47538	2462	95.076	4.924
ack	PG4096	50000	48029	1971	96.058	3.942

bubbleSort500	A	369764	254761	115003	68.8983	31.1017
bubbleSort500	S512	369764	318934	50830	86.2534	13.7466
bubbleSort500	S1024	369764	322039	47725	87.0931	12.9069
bubbleSort500	S2048	369764	323139	46625	87.3906	12.6094
bubbleSort500	S4096	369764	322951	46813	87.3398	12.6602
bubbleSort500	G512	369764	319304	50460	86.3535	13.6465
bubbleSort500	G1024	369764	321270	48494	86.8851	13.1149
bubbleSort500	G2048	369764	323226	46538	87.4141	12.5859
bubbleSort500	G4096	369764	329577	40187	89.1317	10.8683
bubbleSort500	P512	369764	292229	77535	79.0312	20.9688
bubbleSort500	P1024	369764	297292	72472	80.4005	19.5995
bubbleSort500	P2048	369764	300558	69206	81.2837	18.7163
bubbleSort500	P4096	369764	301060	68704	81.4195	18.5805
bubbleSort500	PG512	369764	314690	55074	85.1056	14.8944
bubbleSort500	PG1024	369764	318494	51270	86.1344	13.8656
bubbleSort500	PG2048	369764	322545	47219	87.23	12.77
bubbleSort500	PG4096	369764	327499	42265	88.5697	11.4303
coremark	A	59199	29572	29627	49.9535	50.0465
coremark	S512	59199	51662	7537	87.2684	12.7316
coremark	S1024	59199	52295	6904	88.3376	11.6624
coremark	S2048	59199	52827	6372	89.2363	10.7637
coremark	S4096	59199	52946	6253	89.4373	10.5627
coremark	G512	59199	50808	8391	85.8258	14.1742
coremark	G1024	59199	51440	7759	86.8934	13.1066
coremark	G2048	59199	51528	7671	87.042	12.958
coremark	G4096	59199	51627	7572	87.2092	12.7908
coremark	P512	59199	51344	7855	86.7312	13.2688
coremark	P1024	59199	53043	6156	89.6012	10.3988
coremark	P2048	59199	54166	5033	91.4982	8.50183
coremark	P4096	59199	54656	4543	92.3259	7.67412
coremark	PG512	59199	48479	10720	81.8916	18.1084
coremark	PG1024	59199	50405	8794	85.145	14.855
coremark	PG2048	59199	52043	7156	87.912	12.088
coremark	PG4096	59199	53285	5914	90.01	9.99003
curl	A	500000	252211	247789	50.4422	49.5578
curl	S512	500000	444263	55737	88.8526	11.1474
curl	S1024	500000	456601	43399	91.3202	8.6798
curl	S2048	500000	470242	29758	94.0484	5.9516
curl	S4096	500000	474309	25691	94.8618	5.1382
curl	G512	500000	429533	70467	85.9066	14.0934
curl	G1024	500000	441797	58203	88.3594	11.6406
curl	G2048	500000	452769	47231	90.5538	9.4462
curl	G4096	500000	462795	37205	92.559	7.441
curl	P512	500000	418973	81027	83.7946	16.2054
curl	P1024	500000	440883	59117	88.1766	11.8234

curl	P2048	500000	461131	38869	92.2262	7.7738
curl	P4096	500000	471459	28541	94.2918	5.7082
curl	PG512	500000	398630	101370	79.726	20.274
curl	PG1024	500000	416857	83143	83.3714	16.6286
curl	PG2048	500000	439905	60095	87.981	12.019
curl	PG4096	500000	457320	42680	91.464	8.536
echo_hello	A	62985	31836	31149	50.5454	49.4546
echo_hello	S512	62985	55548	7437	88.1924	11.8076
echo_hello	S1024	62985	55984	7001	88.8847	11.1153
echo_hello	S2048	62985	56314	6671	89.4086	10.5914
echo_hello	S4096	62985	56347	6638	89.461	10.539
echo_hello	G512	62985	54451	8534	86.4507	13.5493
echo_hello	G1024	62985	55245	7740	87.7114	12.2886
echo_hello	G2048	62985	55712	7273	88.4528	11.5472
echo_hello	G4096	62985	55653	7332	88.3591	11.6409
echo_hello	P512	62985	55277	7708	87.7622	12.2378
echo_hello	P1024	62985	56429	6556	89.5912	10.4088
echo_hello	P2048	62985	57396	5589	91.1265	8.87354
echo_hello	P4096	62985	57794	5191	91.7584	8.24164
echo_hello	PG512	62985	52699	10286	83.6691	16.3309
echo_hello	PG1024	62985	54577	8408	86.6508	13.3492
echo_hello	PG2048	62985	56203	6782	89.2324	10.7676
echo_hello	PG4096	62985	57248	5737	90.8915	9.10852
ffmpegenc	A	50000	24165	25835	48.33	51.67
ffmpegenc	S512	50000	44327	5673	88.654	11.346
ffmpegenc	S1024	50000	45135	4865	90.27	9.73
ffmpegenc	S2048	50000	45453	4547	90.906	9.094
ffmpegenc	S4096	50000	45564	4436	91.128	8.872
ffmpegenc	G512	50000	42187	7813	84.374	15.626
ffmpegenc	G1024	50000	43649	6351	87.298	12.702
ffmpegenc	G2048	50000	44959	5041	89.918	10.082
ffmpegenc	G4096	50000	45554	4446	91.108	8.892
ffmpegenc	P512	50000	44912	5088	89.824	10.176
ffmpegenc	P1024	50000	45692	4308	91.384	8.616
ffmpegenc	P2048	50000	46046	3954	92.092	7.908
ffmpegenc	P4096	50000	46194	3806	92.388	7.612
ffmpegenc	PG512	50000	42123	7877	84.246	15.754
ffmpegenc	PG1024	50000	44280	5720	88.56	11.44
ffmpegenc	PG2048	50000	45750	4250	91.5	8.5
ffmpegenc	PG4096	50000	46731	3269	93.462	6.538
ffmpegplay	A	50000	24055	25945	48.11	51.89
ffmpegplay	S512	50000	44446	5554	88.892	11.108
ffmpegplay	S1024	50000	45226	4774	90.452	9.548
ffmpegplay	S2048	50000	45570	4430	91.14	8.86
ffmpegplay	S4096	50000	45688	4312	91.376	8.624

ffmpegplay	G512	50000	42597	7403	85.194	14.806
ffmpegplay	G1024	50000	44013	5987	88.026	11.974
ffmpegplay	G2048	50000	45176	4824	90.352	9.648
ffmpegplay	G4096	50000	45813	4187	91.626	8.374
ffmpegplay	P512	50000	45079	4921	90.158	9.842
ffmpegplay	P1024	50000	45872	4128	91.744	8.256
ffmpegplay	P2048	50000	46240	3760	92.48	7.52
ffmpegplay	P4096	50000	46443	3557	92.886	7.114
ffmpegplay	PG512	50000	42640	7360	85.28	14.72
ffmpegplay	PG1024	50000	44717	5283	89.434	10.566
ffmpegplay	PG2048	50000	45966	4034	91.932	8.068
ffmpegplay	PG4096	50000	46922	3078	93.844	6.156
fft	A	893688	534259	359429	59.7814	40.2186
fft	S512	893688	835883	57805	93.5319	6.46814
fft	S1024	893688	850832	42856	95.2046	4.79541
fft	S2048	893688	853121	40567	95.4607	4.53928
fft	S4096	893688	857212	36476	95.9185	4.08151
fft	G512	893688	808419	85269	90.4587	9.54125
fft	G1024	893688	831815	61873	93.0767	6.92333
fft	G2048	893688	844190	49498	94.4614	5.53862
fft	G4096	893688	852954	40734	95.442	4.55797
fft	P512	893688	800178	93510	89.5366	10.4634
fft	P1024	893688	826076	67612	92.4345	7.5655
fft	P2048	893688	839398	54290	93.9252	6.07483
fft	P4096	893688	852114	41574	95.348	4.65196
fft	PG512	893688	752192	141496	84.1672	15.8328
fft	PG1024	893688	790788	102900	88.4859	11.5141
fft	PG2048	893688	819226	74462	91.668	8.33199
fft	PG4096	893688	839278	54410	93.9117	6.08825
fib100	A	154838	72242	82596	46.6565	53.3435
fib100	S512	154838	141408	13430	91.3264	8.67358
fib100	S1024	154838	144053	10785	93.0347	6.96534
fib100	S2048	154838	145479	9359	93.9556	6.04438
fib100	S4096	154838	146497	8341	94.6131	5.38692
fib100	G512	154838	139907	14931	90.357	9.64298
fib100	G1024	154838	142233	12605	91.8592	8.14077
fib100	G2048	154838	144222	10616	93.1438	6.8562
fib100	G4096	154838	145009	9829	93.6521	6.34793
fib100	P512	154838	134921	19917	87.1369	12.8631
fib100	P1024	154838	138989	15849	89.7641	10.2359
fib100	P2048	154838	144010	10828	93.0069	6.99312
fib100	P4096	154838	146277	8561	94.471	5.529
fib100	PG512	154838	130213	24625	84.0963	15.9037
fib100	PG1024	154838	135471	19367	87.4921	12.5079
fib100	PG2048	154838	142121	12717	91.7869	8.2131

fib100	PG4096	154838	145053	9785	93.6805	6.31951
g++	A	720330	441575	278755	61.3018	38.6982
g++	S512	720330	630669	89661	87.5528	12.4472
g++	S1024	720330	648034	72296	89.9635	10.0365
g++	S2048	720330	659562	60768	91.5639	8.43613
g++	S4096	720330	667097	53233	92.6099	7.39008
g++	G512	720330	600220	120110	83.3257	16.6743
g++	G1024	720330	625414	94916	86.8233	13.1767
g++	G2048	720330	643414	76916	89.3221	10.6779
g++	G4096	720330	657813	62517	91.3211	8.67894
g++	P512	720330	602067	118263	83.5821	16.4179
g++	P1024	720330	632019	88311	87.7402	12.2598
g++	P2048	720330	651231	69099	90.4073	9.59269
g++	P4096	720330	663799	56531	92.1521	7.84793
g++	PG512	720330	562016	158314	78.022	21.978
g++	PG1024	720330	591955	128375	82.1783	17.8217
g++	PG2048	720330	617030	103300	85.6593	14.3406
g++	PG4096	720330	639602	80728	88.7929	11.2071
gcc	A	50000	24584	25416	49.168	50.832
gcc	S512	50000	43875	6125	87.75	12.25
gcc	S1024	50000	44114	5886	88.228	11.772
gcc	S2048	50000	44145	5855	88.29	11.71
gcc	S4096	50000	44179	5821	88.358	11.642
gcc	G512	50000	42270	7730	84.54	15.46
gcc	G1024	50000	42747	7253	85.494	14.506
gcc	G2048	50000	43255	6745	86.51	13.49
gcc	G4096	50000	43327	6673	86.654	13.346
gcc	P512	50000	44127	5873	88.254	11.746
gcc	P1024	50000	44665	5335	89.33	10.67
gcc	P2048	50000	45027	4973	90.054	9.946
gcc	P4096	50000	45447	4553	90.894	9.106
gcc	PG512	50000	41071	8929	82.142	17.858
gcc	PG1024	50000	42643	7357	85.286	14.714
gcc	PG2048	50000	44099	5901	88.198	11.802
gcc	PG4096	50000	44928	5072	89.856	10.144
google_chrome	A	2750516	1486855	1263661	54.0573	45.9427
google_chrome	S512	2750516	2546356	204160	92.5774	7.42261
google_chrome	S1024	2750516	2589735	160781	94.1545	5.84548
google_chrome	S2048	2750516	2617984	132532	95.1816	4.81844
google_chrome	S4096	2750516	2636221	114295	95.8446	4.1554
google_chrome	G512	2750516	2440408	310108	88.7255	11.2745
google_chrome	G1024	2750516	2481710	268806	90.2271	9.77293
google_chrome	G2048	2750516	2522978	227538	91.7274	8.27256
google_chrome	G4096	2750516	2553413	197103	92.834	7.16604
google_chrome	P512	2750516	2298448	452068	83.5642	16.4358

google_chrome	P1024	2750516	2438132	312384	88.6427	11.3573
google_chrome	P2048	2750516	2525899	224617	91.8336	8.16636
google_chrome	P4096	2750516	2575636	174880	93.6419	6.35808
google_chrome	PG512	2750516	2135265	615251	77.6314	22.3686
google_chrome	PG1024	2750516	2227828	522688	80.9967	19.0033
google_chrome	PG2048	2750516	2348903	401613	85.3986	14.6014
google_chrome	PG4096	2750516	2425498	325018	88.1834	11.8166
gpg	A	300000	212863	87137	70.9543	29.0457
gpg	S512	300000	282995	17005	94.3317	5.66833
gpg	S1024	300000	282995	17005	94.3317	5.66833
gpg	S2048	300000	282995	17005	94.3317	5.66833
gpg	S4096	300000	282995	17005	94.3317	5.66833
gpg	G512	300000	290025	9975	96.675	3.325
gpg	G1024	300000	290602	9398	96.8673	3.13267
gpg	G2048	300000	289791	10209	96.597	3.403
gpg	G4096	300000	289772	10228	96.5907	3.40933
gpg	P512	300000	288921	11079	96.307	3.693
gpg	P1024	300000	288921	11079	96.307	3.693
gpg	P2048	300000	288921	11079	96.307	3.693
gpg	P4096	300000	288921	11079	96.307	3.693
gpg	PG512	300000	292167	7833	97.389	2.611
gpg	PG1024	300000	292832	7168	97.6107	2.38933
gpg	PG2048	300000	293232	6768	97.744	2.256
gpg	PG4096	300000	293232	6768	97.744	2.256
grep	A	50000	23317	26683	46.634	53.366
grep	S512	50000	45243	4757	90.486	9.514
grep	S1024	50000	45663	4337	91.326	8.674
grep	S2048	50000	45723	4277	91.446	8.554
grep	S4096	50000	45769	4231	91.538	8.462
grep	G512	50000	42745	7255	85.49	14.51
grep	G1024	50000	43603	6397	87.206	12.794
grep	G2048	50000	44315	5685	88.63	11.37
grep	G4096	50000	44551	5449	89.102	10.898
grep	P512	50000	45259	4741	90.518	9.482
grep	P1024	50000	45913	4087	91.826	8.174
grep	P2048	50000	46226	3774	92.452	7.548
grep	P4096	50000	46587	3413	93.174	6.826
grep	PG512	50000	41804	8196	83.608	16.392
grep	PG1024	50000	43433	6567	86.866	13.134
grep	PG2048	50000	44950	5050	89.9	10.1
grep	PG4096	50000	45844	4156	91.688	8.312
gzip	A	300000	182510	117490	60.8367	39.1633
gzip	S512	300000	289904	10096	96.6347	3.36533
gzip	S1024	300000	289914	10086	96.638	3.362
gzip	S2048	300000	289913	10087	96.6377	3.36233

gzip	S4096	300000	289911	10089	96.637	3.363
gzip	G512	300000	288831	11169	96.277	3.723
gzip	G1024	300000	289747	10253	96.5823	3.41767
gzip	G2048	300000	290314	9686	96.7713	3.22867
gzip	G4096	300000	290687	9313	96.8957	3.10433
gzip	P512	300000	289455	10545	96.485	3.515
gzip	P1024	300000	289457	10543	96.4857	3.51433
gzip	P2048	300000	289458	10542	96.486	3.514
gzip	P4096	300000	289458	10542	96.486	3.514
gzip	PG512	300000	282077	17923	94.0257	5.97433
gzip	PG1024	300000	288782	11218	96.2607	3.73933
gzip	PG2048	300000	289843	10157	96.6143	3.38567
gzip	PG4096	300000	290664	9336	96.888	3.112
jpegtran	A	500000	410334	89666	82.0668	17.9332
jpegtran	S512	500000	392839	107161	78.5678	21.4322
jpegtran	S1024	500000	392839	107161	78.5678	21.4322
jpegtran	S2048	500000	392839	107161	78.5678	21.4322
jpegtran	S4096	500000	392839	107161	78.5678	21.4322
jpegtran	G512	500000	487859	12141	97.5718	2.4282
jpegtran	G1024	500000	493796	6204	98.7592	1.2408
jpegtran	G2048	500000	493941	6059	98.7882	1.2118
jpegtran	G4096	500000	493938	6062	98.7876	1.2124
jpegtran	P512	500000	446408	53592	89.2816	10.7184
jpegtran	P1024	500000	446408	53592	89.2816	10.7184
jpegtran	P2048	500000	446408	53592	89.2816	10.7184
jpegtran	P4096	500000	446408	53592	89.2816	10.7184
jpegtran	PG512	500000	487901	12099	97.5802	2.4198
jpegtran	PG1024	500000	493840	6160	98.768	1.232
jpegtran	PG2048	500000	493988	6012	98.7976	1.2024
jpegtran	PG4096	500000	493988	6012	98.7976	1.2024
jpegtran2	A	867406	453840	413566	52.3215	47.6785
jpegtran2	S512	867406	809420	57986	93.315	6.68499
jpegtran2	S1024	867406	812141	55265	93.6287	6.3713
jpegtran2	S2048	867406	812804	54602	93.7051	6.29486
jpegtran2	S4096	867406	813053	54353	93.7338	6.26615
jpegtran2	G512	867406	805111	62295	92.8182	7.18176
jpegtran2	G1024	867406	810759	56647	93.4694	6.53062
jpegtran2	G2048	867406	815799	51607	94.0504	5.94958
jpegtran2	G4096	867406	817488	49918	94.2451	5.75486
jpegtran2	P512	867406	809732	57674	93.351	6.64902
jpegtran2	P1024	867406	815817	51589	94.0525	5.9475
jpegtran2	P2048	867406	820871	46535	94.6352	5.36485
jpegtran2	P4096	867406	821841	45565	94.747	5.25302
jpegtran2	PG512	867406	770570	96836	88.8361	11.1639
jpegtran2	PG1024	867406	795075	72331	91.6612	8.33877

jpegtran2	PG2048	867406	806010	61396	92.9219	7.07812
jpegtran2	PG4096	867406	812396	55010	93.6581	6.3419
linpack	A	500000	252666	247334	50.5332	49.4668
linpack	S512	500000	497464	2536	99.4928	0.5072
linpack	S1024	500000	497464	2536	99.4928	0.5072
linpack	S2048	500000	497464	2536	99.4928	0.5072
linpack	S4096	500000	497464	2536	99.4928	0.5072
linpack	G512	500000	497489	2511	99.4978	0.5022
linpack	G1024	500000	497477	2523	99.4954	0.5046
linpack	G2048	500000	497465	2535	99.493	0.507
linpack	G4096	500000	497449	2551	99.4898	0.5102
linpack	P512	500000	497347	2653	99.4694	0.5306
linpack	P1024	500000	497347	2653	99.4694	0.5306
linpack	P2048	500000	497347	2653	99.4694	0.5306
linpack	P4096	500000	497347	2653	99.4694	0.5306
linpack	PG512	500000	497009	2991	99.4018	0.5982
linpack	PG1024	500000	497010	2990	99.402	0.598
linpack	PG2048	500000	497011	2989	99.4022	0.5978
linpack	PG4096	500000	497013	2987	99.4026	0.5974
matrix_mult	A	34662	17004	17658	49.0566	50.9434
matrix_mult	S512	34662	30671	3991	88.4859	11.514
matrix_mult	S1024	34662	30965	3697	89.3341	10.6659
matrix_mult	S2048	34662	31124	3538	89.7929	10.2071
matrix_mult	S4096	34662	31131	3531	89.8131	10.1869
matrix_mult	G512	34662	29879	4783	86.201	13.799
matrix_mult	G1024	34662	30390	4272	87.6753	12.3247
matrix_mult	G2048	34662	30816	3846	88.9043	11.0957
matrix_mult	G4096	34662	30779	3883	88.7975	11.2025
matrix_mult	P512	34662	30678	3984	88.5061	11.4939
matrix_mult	P1024	34662	31663	2999	91.3479	8.65213
matrix_mult	P2048	34662	32167	2495	92.8019	7.19808
matrix_mult	P4096	34662	32376	2286	93.4049	6.59512
matrix_mult	PG512	34662	29813	4849	86.0106	13.9894
matrix_mult	PG1024	34662	30889	3773	89.1149	10.8851
matrix_mult	PG2048	34662	31810	2852	91.772	8.22803
matrix_mult	PG4096	34662	32311	2351	93.2174	6.78264
mergesort	A	407099	249102	157997	61.1895	38.8105
mergesort	S512	407099	383509	23590	94.2053	5.79466
mergesort	S1024	407099	385401	21698	94.6701	5.32991
mergesort	S2048	407099	388982	18117	95.5497	4.45027
mergesort	S4096	407099	389119	17980	95.5834	4.41662
mergesort	G512	407099	377845	29254	92.814	7.18597
mergesort	G1024	407099	385716	21383	94.7475	5.25253
mergesort	G2048	407099	387174	19925	95.1056	4.89439
mergesort	G4096	407099	388260	18839	95.3724	4.62762

mergesort	P512	407099	376664	30435	92.5239	7.47607
mergesort	P1024	407099	381361	25738	93.6777	6.3223
mergesort	P2048	407099	388148	18951	95.3449	4.65513
mergesort	P4096	407099	389740	17359	95.7359	4.26407
mergesort	PG512	407099	363116	43983	89.196	10.804
mergesort	PG1024	407099	374159	32940	91.9086	8.0914
mergesort	PG2048	407099	380954	26145	93.5777	6.42227
mergesort	PG4096	407099	386225	20874	94.8725	5.1275
mxm	A	350281	199883	150398	57.0636	42.9364
mxm	S512	350281	326064	24217	93.0864	6.91359
mxm	S1024	350281	328206	22075	93.6979	6.30208
mxm	S2048	350281	332014	18267	94.785	5.21496
mxm	S4096	350281	332418	17863	94.9004	5.09962
mxm	G512	350281	319683	30598	91.2647	8.73527
mxm	G1024	350281	328125	22156	93.6748	6.32521
mxm	G2048	350281	329431	20850	94.0476	5.95236
mxm	G4096	350281	330759	19522	94.4268	5.57324
mxm	P512	350281	321727	28554	91.8483	8.15174
mxm	P1024	350281	325542	24739	92.9374	7.06262
mxm	P2048	350281	332027	18254	94.7888	5.21124
mxm	P4096	350281	333975	16306	95.3449	4.65512
mxm	PG512	350281	309420	40861	88.3348	11.6652
mxm	PG1024	350281	318891	31390	91.0386	8.96138
mxm	PG2048	350281	324661	25620	92.6859	7.31413
mxm	PG4096	350281	329652	20629	94.1107	5.88927
picalc	A	637334	322835	314499	50.654	49.346
picalc	S512	637334	629154	8180	98.7165	1.28347
picalc	S1024	637334	629406	7928	98.7561	1.24393
picalc	S2048	637334	629643	7691	98.7933	1.20675
picalc	S4096	637334	630031	7303	98.8541	1.14587
picalc	G512	637334	628040	9294	98.5417	1.45826
picalc	G1024	637334	629300	8034	98.7394	1.26056
picalc	G2048	637334	630189	7145	98.8789	1.12108
picalc	G4096	637334	630311	7023	98.8981	1.10193
picalc	P512	637334	628601	8733	98.6298	1.37024
picalc	P1024	637334	629866	7468	98.8282	1.17176
picalc	P2048	637334	631080	6254	99.0187	0.981275
picalc	P4096	637334	631890	5444	99.1458	0.854183
picalc	PG512	637334	625518	11816	98.146	1.85397
picalc	PG1024	637334	628160	9174	98.5606	1.43943
picalc	PG2048	637334	629955	7379	98.8422	1.15779
picalc	PG4096	637334	631518	5816	99.0874	0.912551
pythonsort	A	7653016	3340263	4312753	43.6464	56.3536
pythonsort	S512	7653016	6820391	832625	89.1203	10.8797
pythonsort	S1024	7653016	6987930	665086	91.3095	8.69051

pythonsort	S2048	7653016	7092568	560448	92.6768	7.32323
pythonsort	S4096	7653016	7194533	458483	94.0091	5.99088
pythonsort	G512	7653016	6556165	1096851	85.6677	14.3323
pythonsort	G1024	7653016	6731421	921595	87.9577	12.0422
pythonsort	G2048	7653016	6880151	772865	89.9012	10.0988
pythonsort	G4096	7653016	7002908	650108	91.5052	8.49479
pythonsort	P512	7653016	5806645	1846371	75.8739	24.1261
pythonsort	P1024	7653016	6180082	1472934	80.7535	19.2465
pythonsort	P2048	7653016	6490198	1162818	84.8058	15.1942
pythonsort	P4096	7653016	6759419	893597	88.3236	11.6764
pythonsort	PG512	7653016	5255221	2397795	68.6686	31.3314
pythonsort	PG1024	7653016	5533721	2119295	72.3077	27.6923
pythonsort	PG2048	7653016	5861090	1791926	76.5854	23.4146
pythonsort	PG4096	7653016	6156815	1496201	80.4495	19.5505
quicksort	A	38816	20300	18516	52.298	47.702
quicksort	S512	38816	34440	4376	88.7263	11.2737
quicksort	S1024	38816	34731	4085	89.476	10.524
quicksort	S2048	38816	34896	3920	89.9011	10.0989
quicksort	S4096	38816	34904	3912	89.9217	10.0783
quicksort	G512	38816	33619	5197	86.6112	13.3888
quicksort	G1024	38816	34126	4690	87.9174	12.0826
quicksort	G2048	38816	34546	4270	88.9994	11.0006
quicksort	G4096	38816	34483	4333	88.8371	11.1629
quicksort	P512	38816	34366	4450	88.5357	11.4643
quicksort	P1024	38816	35317	3499	90.9857	9.01432
quicksort	P2048	38816	35879	2937	92.4335	7.56647
quicksort	P4096	38816	36075	2741	92.9385	7.06152
quicksort	PG512	38816	33356	5460	85.9336	14.0664
quicksort	PG1024	38816	34594	4222	89.123	10.877
quicksort	PG2048	38816	35532	3284	91.5396	8.46043
quicksort	PG4096	38816	36049	2767	92.8715	7.1285
yes	A	693297	427601	265696	61.6765	38.3235
yes	S512	693297	634228	59069	91.48	8.52001
yes	S1024	693297	649678	43619	93.7085	6.29153
yes	S2048	693297	658158	35139	94.9316	5.06839
yes	S4096	693297	662892	30405	95.6144	4.38557
yes	G512	693297	645179	48118	93.0595	6.94046
yes	G1024	693297	654652	38645	94.4259	5.57409
yes	G2048	693297	675411	17886	97.4202	2.57985
yes	G4096	693297	679583	13714	98.0219	1.97808
yes	P512	693297	629939	63358	90.8614	9.13865
yes	P1024	693297	651073	42224	93.9097	6.09032
yes	P2048	693297	660140	33157	95.2175	4.78251
yes	P4096	693297	670223	23074	96.6718	3.32816
yes	PG512	693297	623109	70188	89.8762	10.1238

yes	PG1024	693297	641147	52150	92.478	7.52203
yes	PG2048	693297	667624	25673	96.297	3.70303
yes	PG4096	693297	678546	14751	97.8723	2.12766
zip	A	950641	507996	442645	53.4372	46.5628
zip	S512	950641	853044	97597	89.7336	10.2664
zip	S1024	950641	861620	89021	90.6357	9.36431
zip	S2048	950641	866424	84217	91.141	8.85897
zip	S4096	950641	870687	79954	91.5895	8.41054
zip	G512	950641	825147	125494	86.799	13.201
zip	G1024	950641	835157	115484	87.852	12.148
zip	G2048	950641	844227	106414	88.8061	11.1939
zip	G4096	950641	853634	97007	89.7956	10.2044
zip	P512	950641	809446	141195	85.1474	14.8526
zip	P1024	950641	831382	119259	87.4549	12.5451
zip	P2048	950641	843310	107331	88.7096	11.2904
zip	P4096	950641	863476	87165	90.8309	9.16908
zip	PG512	950641	732428	218213	77.0457	22.9543
zip	PG1024	950641	765330	185311	80.5067	19.4933
zip	PG2048	950641	791521	159120	83.2618	16.7382
zip	PG4096	950641	820490	130151	86.3091	13.6909