# Dynamically Changing Cache Associativity and Size in L1/L2

Author

Prasad Khedekar (G00973357)

**Motivation:**

- Memory latency is very long in today's processors (hundreds of processor clock cycles) and it continues to increase, since DRAM latency is not reducing as fast as processor cycle time.
- To counter the ill-effects faced by this increasing memory latency, the concept of 'Caching' was introduced in 1965 by Wilkes.
- Since its inception, caches have been widely used as they are simple and effective, although it is not completely foolproof because of its inefficiency and passivity.
- One of its important drawbacks is that the amount of wastage of resources and processor cycles on a cache miss is very high.
- Cache misses are a serious issue, especially when not all applications or phases of an application exhibit the same temporal or spatial locality. It might as well happen that a certain phase of an application exhibits excellent temporal/spatial locality leading to low or no cache misses, and a certain other phase of an application exhibits extremely poor temporal/spatial locality leading to numerous cache misses, which translates to energy and power inefficiency.

**Concept:**

- One of the ways to address this problem is to dynamically change the L1 and L2 cache size and associativity to best suit a certain phase of an application, thereby reducing power and resources wastage, and increasing the overall power/energy efficiency of the system.
- Hence, instead of keeping the cache size and associativity static throughout the application, we dynamically adjust cache parameters based on the phase of the application.
- An example to illustrate this concept: Assume we have L2 cache with size as 128KB and associativity at 8. This cache would perform very well in terms of reaching high IPC values for phases which exhibit high spatial/temporal locality, but would be highly power inefficient in phases with low temporal/spatial locality due to its associativity at 8 and size at 128KB. In such a situation, it would be extremely power efficient if the size and associativity of the L2 cache during such a phase is dropped down to maybe 64KB and 4 respectively. This helps the power efficiency of the overall system as the application is anyways not exhibiting high spatial/temporal locality and by reducing the cache size and associativity we are conserving power and energy which would have otherwise gone to waste.

**Project Description:**

- The project exhibits a study on how cache associativity and cache size can affect the processor efficiency or target functions (IPC/power, IPC^2/power) during different phases of an application.

- The setup involves a set of simulations with different L1 and L2 cache configurations. The target functions are then calculated based on the L1 and L2 cache power dissipation values obtained using CACTI.
- The SMTSIM code has been modified to report the IPC values of the application periodically. Using these IPC values and the configuration parameters of the cache such as size and associativity, the target functions (IPC/power, IPC^2/power) are calculated during different phases of an application to then dynamically determine the most power efficient cache configuration suitable for that particular phase of the application.

**Project Implementation:**

The project implementation and methodology can be broadly divided into 3 parts as follows:

A) Modification in SMTSIM code:

- The SMTSIM source code had to be modified to make it report the IPC values of a certain phase of the application periodically, specifically every 10K instructions.
- The modifications were made in 'run.c', which is one of the source code files. The IPC values are calculated based on the number of committed instructions in that sampling interval (based on number of cycles).
- To acquire the number of committed instructions, the 'total_commits' element in 'extra' data structure is accessed.
- The logic for calculating and reporting the IPC values resides in an 'if' control block. The control block ensures that IPC values are reported every 10K instructions. The IPC is calculated within this block based on the number of committed instructions and the number of cycles in that sampling interval and then printed to the log file. The current value of the number of committed instructions and current cycle number are then stored in variables for the calculation of IPC values in the next sampling interval.
- After making these changes in the source code, the SMTSIM tool is built again for these changes to take effect using the command 'make smtsim'.

B) Generating shell scripts, output files and extracting IPC data:

- Once the SMTSIM changes have been done, the tool is run based on the 16 different configurations of L1 and L2 cache parameters for 3 different benchmarks, namely, applu, apsi and crafty. The configurations correspond to:

| Configuration No. | L1 I-cache, D-cache Size | L1 I-cache, D-cache Associativity | L2 cache size | L2 cache associativity |
|---|---|---|---|---|
| 0 | 32768 | 1 | 131072 | 4 |
| 1 | 32768 | 1 | 131072 | 8 |
| 2 | 32768 | 1 | 262144 | 4 |
| 3 | 32768 | 1 | 262144 | 8 |

| | | | | |
|---|---|---|---|---|
| 4 | 32768 | 2 | 131072 | 4 |
| 5 | 32768 | 2 | 131072 | 8 |
| 6 | 32768 | 2 | 262144 | 4 |
| 7 | 32768 | 2 | 262144 | 8 |
| 8 | 65536 | 1 | 131072 | 4 |
| 9 | 65536 | 1 | 131072 | 8 |
| 10 | 65536 | 1 | 262144 | 4 |
| 11 | 65536 | 1 | 262144 | 8 |
| 12 | 65536 | 2 | 131072 | 4 |
| 13 | 65536 | 2 | 131072 | 8 |
| 14 | 65536 | 2 | 262144 | 4 |
| 15 | 65536 | 2 | 262144 | 8 |

- Once the shell scripts are generated by the tool, the 'executeall.sh' shell script is used to generate the output files by running each of the shell scripts.
- The output files contain the periodic reports of the IPC values.
- These IPC values are extracted using a Python script into a separate text file. The extracting of IPC values is implemented using the concept of 'Regular Expressions' or simply 'regex' in Python. It makes use of string expressions search to find and store the data to be extracted in a text file.

C) Creating the table and traversing it to find the most efficient configuration:

- For all the 16 different configurations based on varying the L1 and L2 cache parameters, the power calculations were done based on values retrieved from CACTI.
- This table then served as the feed for the power values in the calculations of the target functions (IPC/Power and IPC^2/Power), achieved using EXCEL formulae.
- The IPC values previously extracted using the Python script into a text file are uploaded into the excel spreadsheet using an EXCEL macro into appropriate columns based on the formulae.
- The table now contains the required information in the form of configuration number, sampling intervals (cycles), IPC value, and the target functions IPC/Power and IPC^2/Power.
- Such a table is created for all the 3 benchmark applications.
- The table is traversed using EXCEL formulae and macros to identify the most efficient cache configuration.
- A cache configuration is considered to be most efficient when it has the highest value for IPC/Power and IPC^2/Power target functions among all the 16 configurations.
- For every sampling interval, the cache configuration with the highest values for the target functions is deduced. Thus, for each of the 1000 phases (sampling intervals) of the

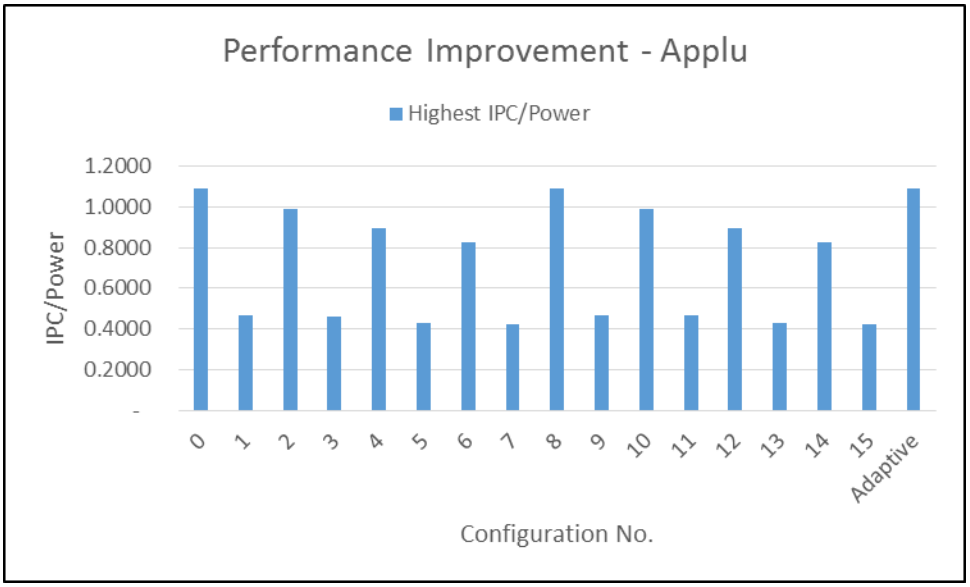
application, we now have the most efficient cache configuration suitable to that specific sampling interval/phase.

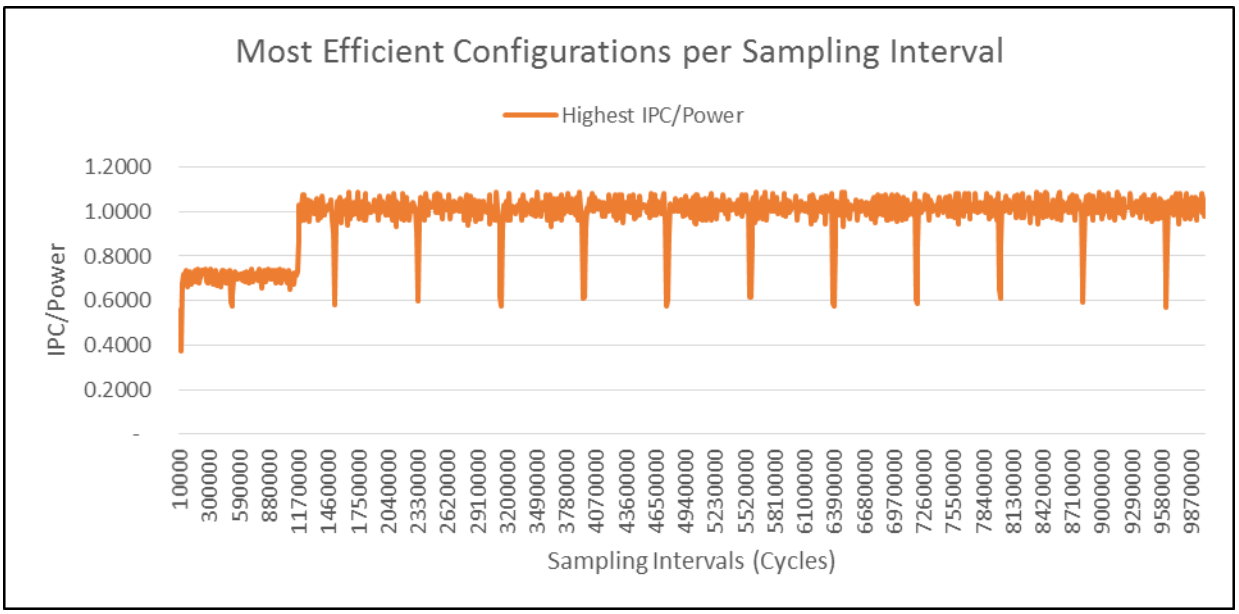
- The most efficient cache configuration for the entire application would simply be that cache configuration which is the most efficient for the maximum number of sampling intervals/phases in that application.
- This is implemented by using a separate table which simply counts the number of sampling intervals/phases for whom that cache configuration is deduced to be the most efficient one. In this way, we dynamically assess which cache configuration suits a particular application most efficiently in terms of power consumption by traversing the table which contains all the calculated values for these target functions.

**Results, Observations and Explanation:**

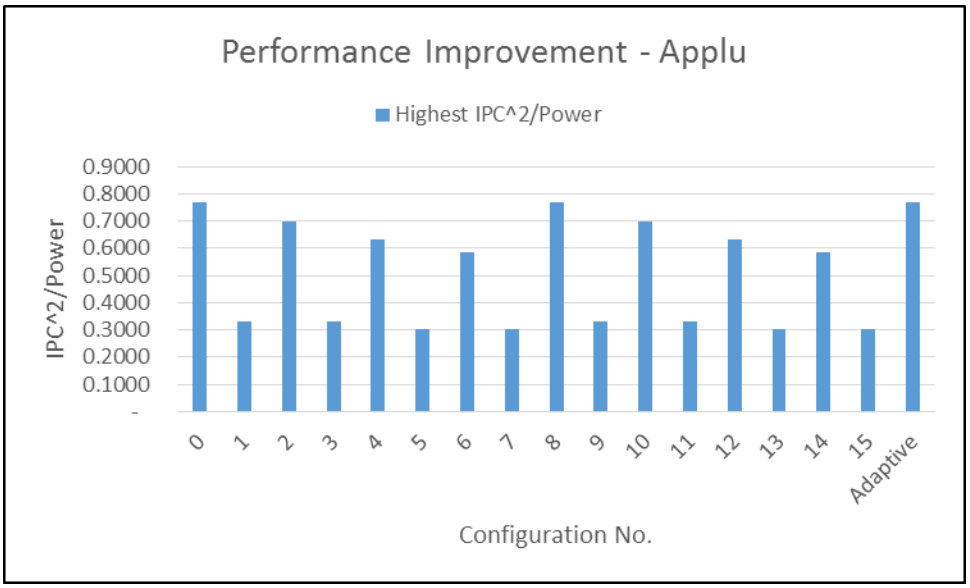
Applu:

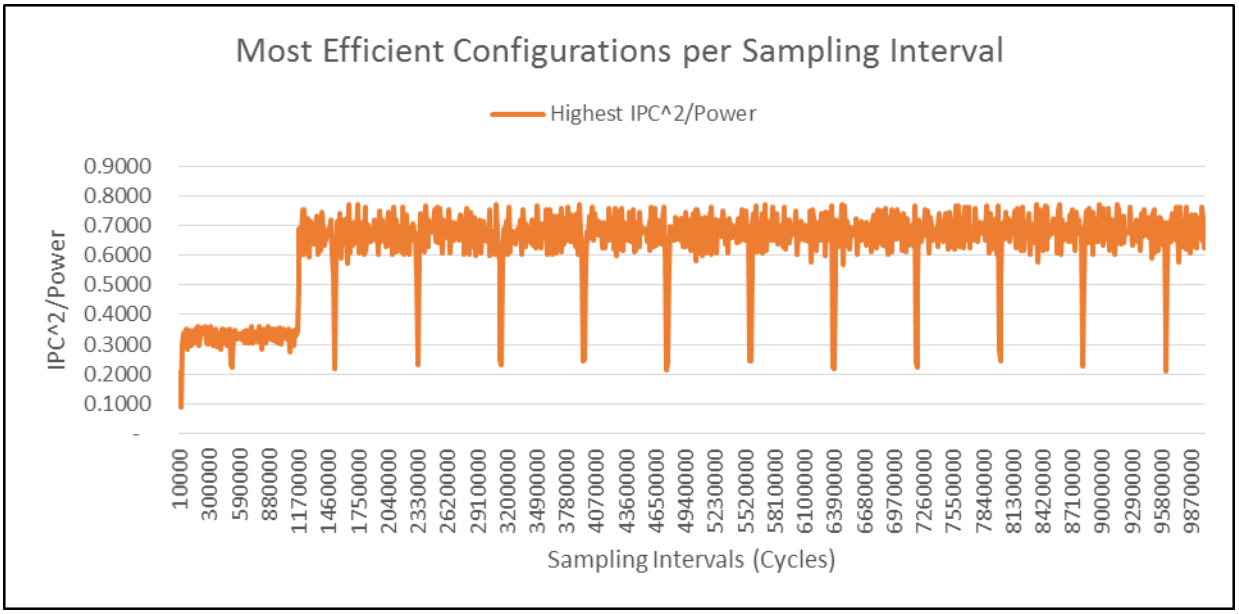
a) IPC/Power Target Function:






b) IPC^2/Power Target Function:






Most Efficient Configuration (Entire Application):

| Config No. | # of phases having this config as most efficient (IPC/Power) | # of phases having this config as most efficient (IPC^2/Power) |
|---|---|---|
| 0 | 533 | 502 |
| 1 | 0 | 0 |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |
| 5 | 0 | 0 |
| 6 | 0 | 0 |
| 7 | 0 | 0 |
| 8 | 458 | 417 |
| 9 | 0 | 0 |
| 10 | 9 | 71 |
| 11 | 0 | 0 |
| 12 | 0 | 10 |
| 13 | 0 | 0 |
| 14 | 0 | 0 |
| 15 | 0 | 0 |

Observations and Explanation:

For Applu benchmark, two of the most efficient configurations in terms of both the target functions are Configuration Nos. 0 and 8. Configuration 0 corresponds to L1 cache size and associativity at 32KB and 1 respectively with L2 cache size and associativity at 128KB and 4

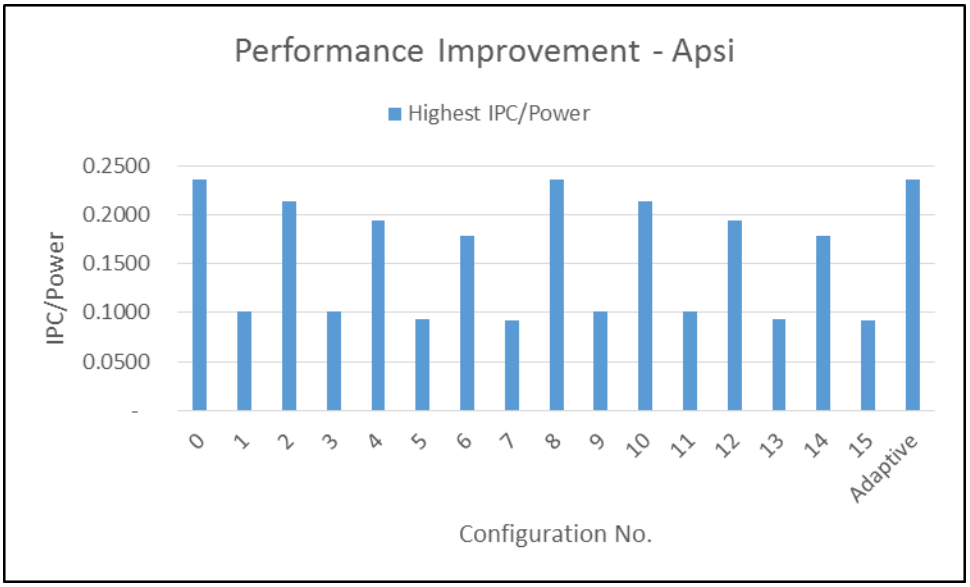
respectively. Configuration 8 corresponds to L1 cache size and associativity at 64KB and 1 respectively with L2 cache size and associativity at 128KB and 4 respectively. The adaptive configuration (most efficient) matches to that of the values of Configuration 0 for both the target functions. The "Performance Improvement" graph clearly shows a drop in efficiency whenever the L2 cache associativity is increased to 8 for both the target functions.

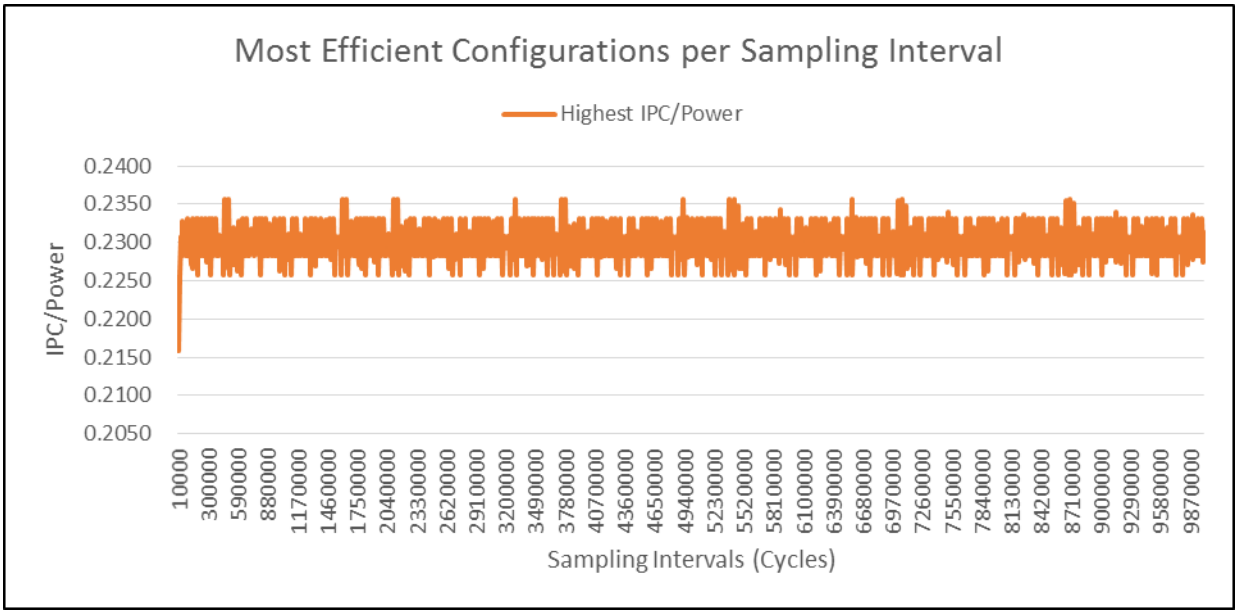
The "Most Efficient Configurations per Sampling Interval" graph depicts the highest values for both the target functions (in separate graphs) at each sampling interval across the 10 million instructions executed. We see that during the initial phase of the application from 0 to 1,180,000 instructions, the highest value of the target functions is capped at approximately 50% compared to the remainder of the application. We can deduce that this phase has lower locality of reference than the rest of the application. After the initial phase, we can see that the rest of the phases in the application are steadily at the highest value for both the target functions, with periodic drops in the values. We can deduce that these phases exhibit the highest locality of reference for this application.

The table shows us for each configuration the number of instances it happens to be the most efficient configuration for that particular phase of the application. From that table, we can see that over 90% of the sampling intervals have either configuration 0 or 8 as the most efficient. Thus, dynamically setting the L1 cache size at either 32KB or 64KB depending on the phase of the application and keeping other parameters like L1 associativity at 1, L2 cache size at 128KB and L2 associativity at 4, we can achieve maximum efficiency in terms of IPC/Power and IPC^2/Power target functions.

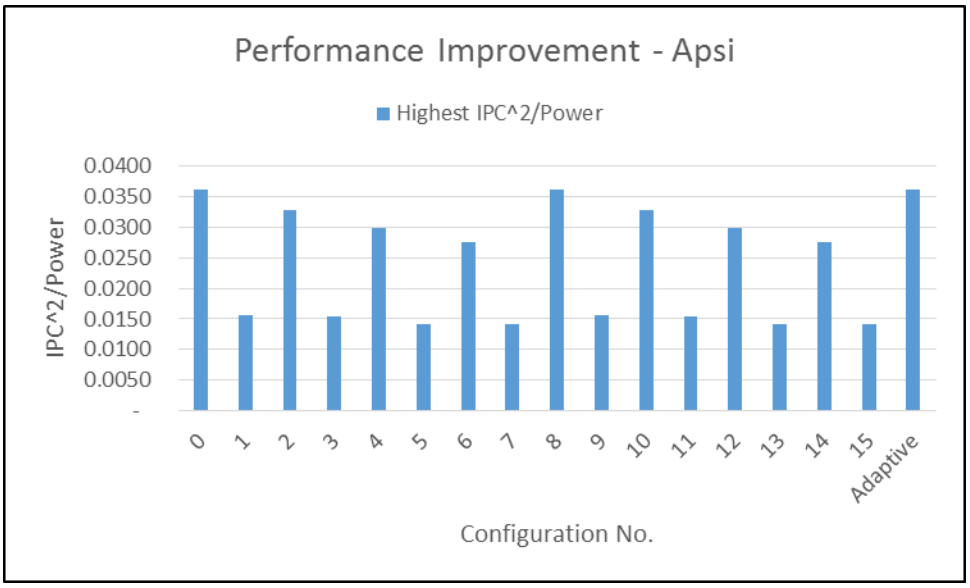
<u>Apsi:</u>

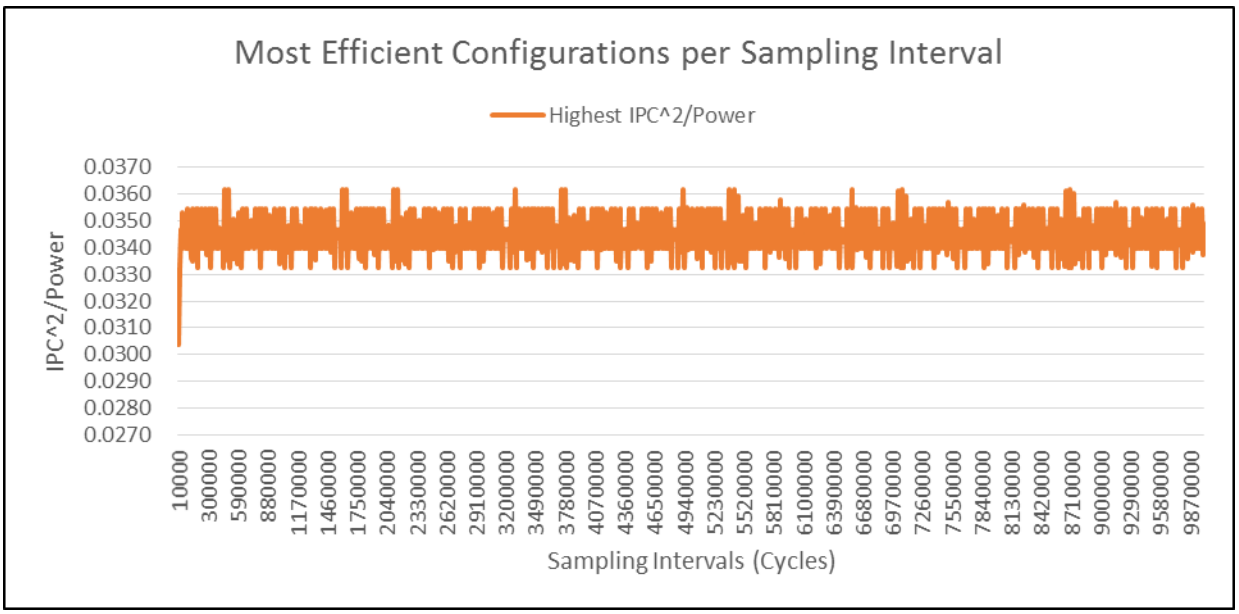
a) IPC/Power Target Function:






b) IPC^2/Power Target Function:






Most Efficient Configuration (Entire Application):

| Config No. | # of phases having this config as most efficient (IPC/Power) | # of phases having this config as most efficient (IPC^2/Power) |
|---|---|---|
| 0 | 1000 | 1000 |
| 1 | 0 | 0 |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |
| 5 | 0 | 0 |
| 6 | 0 | 0 |
| 7 | 0 | 0 |
| 8 | 0 | 0 |
| 9 | 0 | 0 |
| 10 | 0 | 0 |
| 11 | 0 | 0 |
| 12 | 0 | 0 |
| 13 | 0 | 0 |
| 14 | 0 | 0 |
| 15 | 0 | 0 |

Observations and Explanation:

For Apsi benchmark, two of the most efficient configurations in terms of both the target functions are Configuration Nos. 0 and 8. Configuration 0 corresponds to L1 cache size and associativity at 32KB and 1 respectively with L2 cache size and associativity at 128KB and 4

respectively. Configuration 8 corresponds to L1 cache size and associativity at 64KB and 1 respectively with L2 cache size and associativity at 128KB and 4 respectively. The adaptive configuration (most efficient) matches to that of the values of both Configuration 0 as well as Configuration 8 for both the target functions. The "Performance Improvement" graph clearly shows a drop in efficiency whenever the L2 cache associativity is increased to 8 for both the target functions.

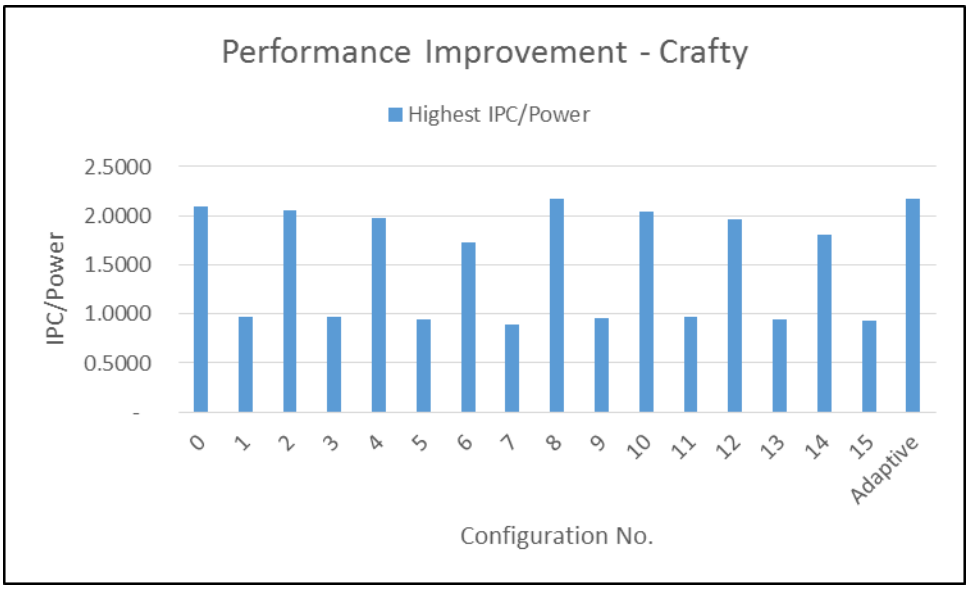
In the "Most Efficient Configurations per Sampling Interval" graphs, we can see that the highest achievable efficiency is constant at a specific value throughout the application for both the target functions. We can also see miniscule drops in efficiency values at regular periods in both the target function graphs. From this, we can deduce that this application features constant locality of reference throughout the application.

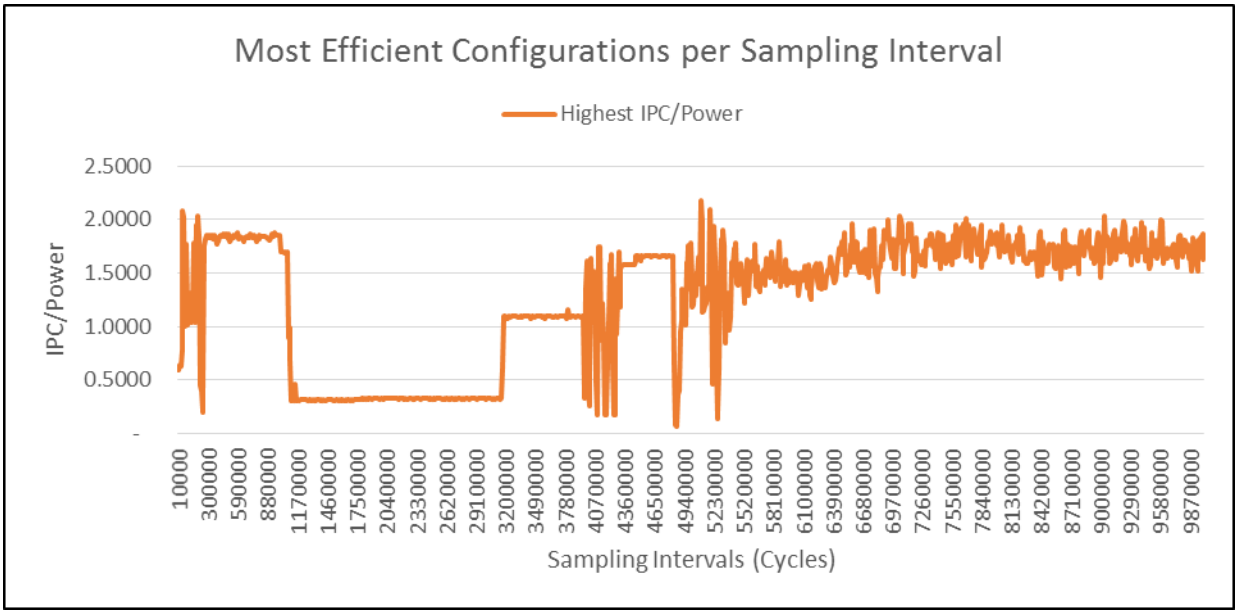
From the table, we can see that for all the sampling intervals throughout the application, only configuration 0 happens to be the most efficient. In reality, it could either be configuration 0 or 8 as they exhibit exactly the same efficiency values for both the target functions (refer "Performance Improvement" graph).

Thus, we can infer that for applications such as "Apsi", our dynamic model for changing cache parameters may prove to be less efficient than setting static configuration for the cache parameters in the long run, due to the overhead involved in dynamically changing the cache parameters. This happens as the two most efficient configurations for the application exhibit exactly same efficiency values and even the application features constant locality of reference. In such cases, it proves to be more efficient just to set the cache parameters to static known configuration which performs most efficiently. We can add functionality to our dynamic model to predict such applications and thus avoid dynamic changes to increase efficiency.

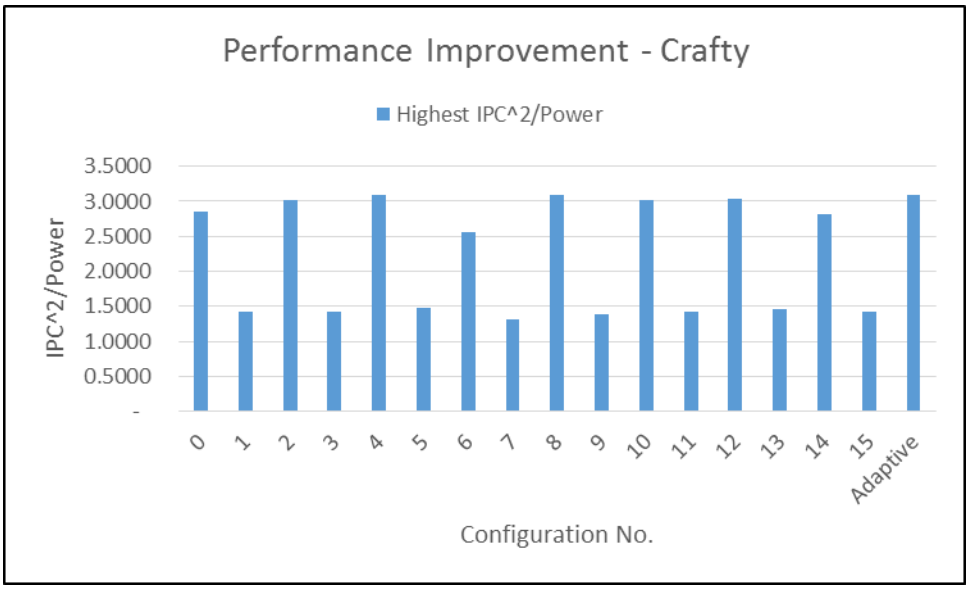
Crafty:

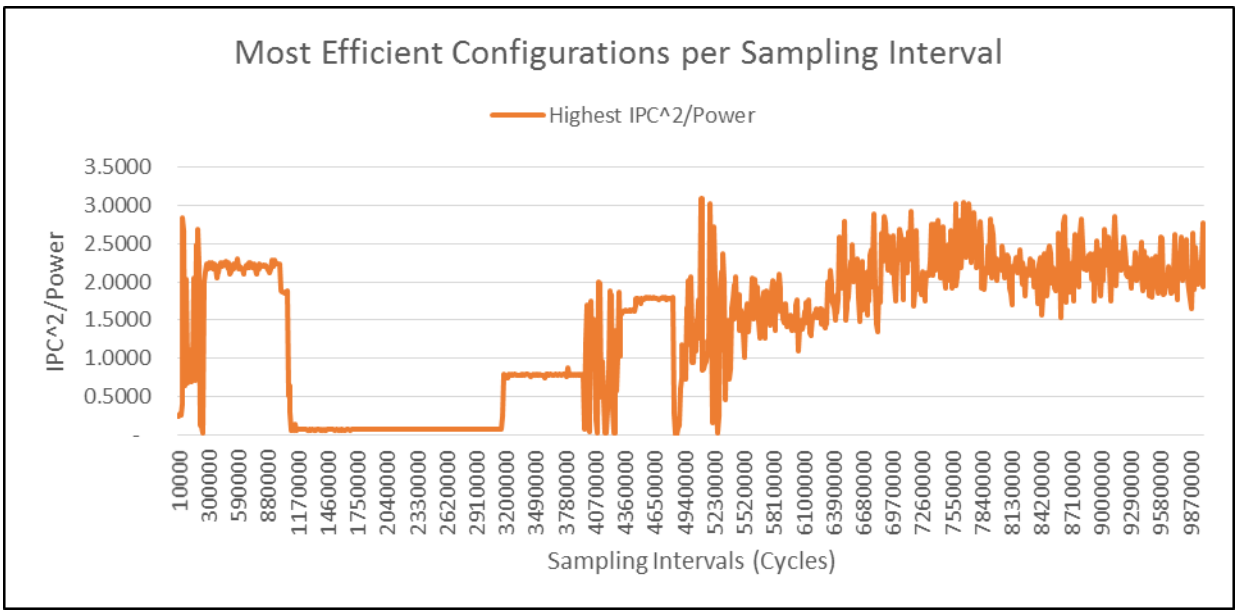
a) IPC/Power Target Function:






b) IPC^2/Power Target Function:






Most Efficient Configuration (Entire Application):

| Config No. | # of phases having this config as most efficient (IPC/Power) | # of phases having this config as most efficient (IPC^2/Power) |
|---|---|---|
| 0 | 277 | 251 |
| 1 | 0 | 0 |
| 2 | 13 | 13 |
| 3 | 0 | 0 |
| 4 | 25 | 37 |
| 5 | 0 | 0 |
| 6 | 8 | 27 |
| 7 | 0 | 0 |
| 8 | 484 | 373 |
| 9 | 0 | 0 |
| 10 | 70 | 67 |
| 11 | 0 | 0 |
| 12 | 87 | 138 |
| 13 | 0 | 0 |
| 14 | 36 | 94 |
| 15 | 0 | 0 |

Observations and Explanation:

From the “Performance Improvement” graph for IPC/Power, the value for the adaptive (most efficient) configuration matches to that of configuration 8, with configurations 0, 2 and 10 exhibiting almost similar performance. From the “Performance Improvement” graph for

IPC^2/Power, the value for the adaptive (most efficient) configuration matches to that of configuration 4, with configurations 2, 8, 10 and 12 exhibiting almost similar performance. The "Performance Improvement" graph clearly shows a drop in efficiency whenever the L2 cache associativity is increased to 8 for both the target functions. These configurations correspond to:

| Configuration No. | L1 I-cache, D-cache Size | L1 I-cache, D-cache Associativity | L2 cache size | L2 cache associativity |
|---|---|---|---|---|
| 0 | 32768 | 1 | 131072 | 4 |
| 1 | 32768 | 1 | 131072 | 8 |
| 2 | 32768 | 1 | 262144 | 4 |
| 3 | 32768 | 1 | 262144 | 8 |
| 4 | 32768 | 2 | 131072 | 4 |
| 5 | 32768 | 2 | 131072 | 8 |
| 6 | 32768 | 2 | 262144 | 4 |
| 7 | 32768 | 2 | 262144 | 8 |
| 8 | 65536 | 1 | 131072 | 4 |
| 9 | 65536 | 1 | 131072 | 8 |
| 10 | 65536 | 1 | 262144 | 4 |
| 11 | 65536 | 1 | 262144 | 8 |
| 12 | 65536 | 2 | 131072 | 4 |
| 13 | 65536 | 2 | 131072 | 8 |
| 14 | 65536 | 2 | 262144 | 4 |
| 15 | 65536 | 2 | 262144 | 8 |

From this table, we can clearly see that the application achieves maximum efficiency when L2 cache associativity is at 4 and L1 associativity is kept at 1 when L2 size is increased to 256KB.

In the "Most Efficient Configurations per Sampling Interval" graphs, we can observe a similar trend in terms of locality of reference for both the target functions. During the phase from 1,100,000 to 3,180,000 instructions, we can observe a sharp fall in the efficiency values for both the target functions. This can be attributed to low levels of locality of reference in this phase of the application. During such phases, our dynamic model would improve power efficiency by adjusting the cache parameters to configurations which consume low power as seen in the graph. The entire application exhibits distinct phases where cache performance is tested and the goal of lowering power consumption is achieved by dynamic changes to parameters.

From the table, we can deduce that over 62% of sampling intervals have either configuration 0 or 8 as the most efficient for both the target functions. But, the application exhibits phases where other configurations apart from these two prove to be more efficient, and these are the instances where our dynamic model would improve efficiency by changing the cache parameters appropriately to reduce power consumption.

Thus, we can infer that for applications such as “Crafty”, the dynamic model of adjusting cache parameters proves to be more power efficient as it senses the application phases of lower locality of reference and appropriately selects the configuration with the lowest power consumption to reduce inefficiency.

## Conclusion:

In this project, the feasibility of dynamically changing cache associativity and size in L1 and L2 to improve power efficiency is studied. The target functions IPC/Power and IPC^2/Power helped us project the power consumption for different cache configurations during different phases/sampling intervals of a specific application. From the findings of the project, we can conclude that the dynamic model for changing cache parameters can help to achieve higher power efficiency than static setting of cache parameters for certain applications which exhibit changes in locality of reference at different phases of the application such as “Crafty”, “Applu”. For applications which do not exhibit changes in their locality of reference throughout the application such as “Apsi”, our dynamic model may prove to be less power efficient than the static model considering the overhead of making dynamic changes to cache configurations with minimal impact on power efficiency. For such applications, the dynamic model can have an added functionality to avoid changing cache parameters when the application has fairly consistent locality of reference.