ECE-611

Project Report on

Dynamically Changing Cache Associativity and Size in L1/L2

By

Onkar Mahadev Randive

Introduction

One of the main limitations in improving the performance of computers is the memory access time required by the processor to access the memory locations to perform computations and store the results of these computations. There is an inherent trade-off between size and speed (given that a larger resource implies greater physical distances) but also a trade-off between expensive, premium technologies (such as SRAM) VS cheaper, easily mass-produced commodities (such as DRAM or hard disks). Cache is a trade-off between these two types of memories. The use of a cache also allows for higher throughput from the underlying resource, by assembling multiple fine grain transfers into larger, more efficient requests.

Cache misses can be a serious issue, as not all applications or phases of an application execute in the similar fashion or require the same hardware resources. It might be the case that a certain phase of an application exhibits excellent locality leading to low or no cache misses, and a certain other phase of an application exhibits extremely locality leading to numerous cache misses, which translates to energy and power inefficiency.

Hence, there might be a need to dynamically change the associativity and size of the cache as per the need of the application. This ensures or at least helps in achieving higher performance and efficiency. The size and associativity of L2 and L3 can be changed or varied effectively to increase performance as well as achieve better efficiency.

Methodology

In this project we will implement how cache associativity and cache size can affect th e processor efficiency or target functions (IPC/power, IPC^2/power) during different phases of an application. A set of simulations with different L1 and L2 configurations are simulated on SMTSIM for a single configuration. Three set of workloads are used so as to have a holistic approach and not be constrained to a specific benchmark or type of application. We change the python scripts so as to different configurations of caches with varying cache size and associativity. IPC values are calculated during the execution of the application instead of calculating it at the end of the completion. This enables us to study the performance of the application during various stages of the application's execution. The intermediate IPC are obtained every 10000 cycle i.e they are sampled every 10000 instructions. We are taking 10 million instructions for our simulation. Hence we will have 10000 intermediate IPC values. Further we obtain static and dynamic power for these configurations using CACTI tool. These helps us to understand the power requirements of the application during the phases of its execution. We try to obtain the IPC/Power and IPC^2/Power functions which can also be called as efficiency. Thus, we can change the size and associativity of caches to obtain higher performance and better efficiency.

Implementation

- Firstly, to obtain the intermediate IPC values during the execution of the application, we have to modify the source code of SMTSIM.
- We change the 'run.c' file present in the source code of SMTSIM such that we we can get IPC values after every 10000 instructions for 10 million instructions.
- We find the IPC during the sampling interval of 10000 using 'total_commits' which gives us the total number of commited instructions. we print the IPC only upto the 10 million instructions.
- After changing the source code for the SMTSIM, we again build the SMTSIM so that the changes are reflected onto the SMTSIM executable file.
- Next, we change the python script for a single core so that we have the required 16 configurations.

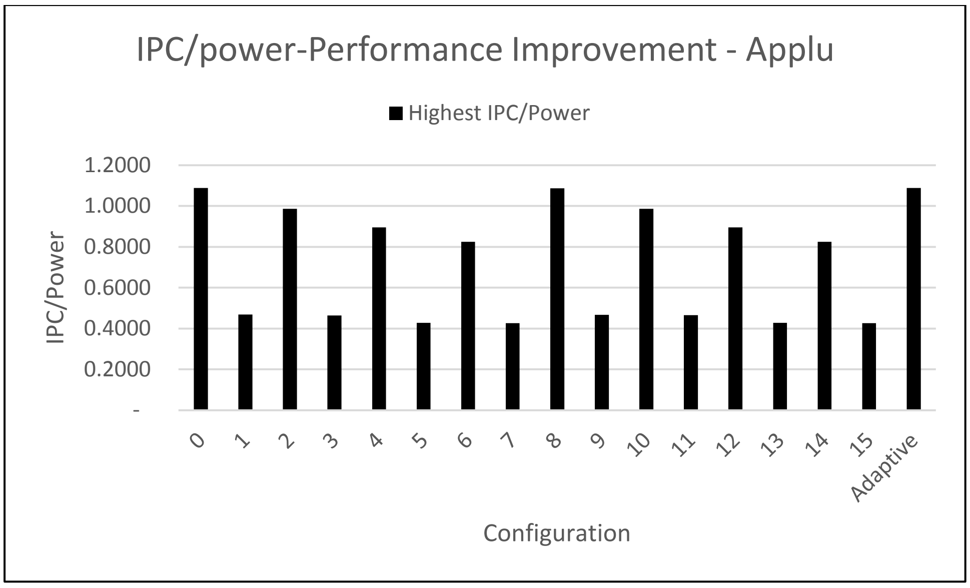
Configuration	L1 size	L1 associativity	L2 size	L2 associativity
0	32kb	1	128kb	4
1	32kb	1	128kb	8
2	32kb	1	256kb	4
3	32kb	1	256kb	8
4	32kb	2	128kb	4
5	32kb	2	128kb	8
6	32kb	2	256kb	4
7	32kb	2	256kb	8
8	64kb	1	128kb	4
9	64kb	1	128kb	8
10	64kb	1	256kb	4
11	64kb	1	256kb	8
12	64kb	2	128kb	4
13	64kb	2	128kb	8
14	64kb	2	256kb	4
15	64kb	2	256kb	8

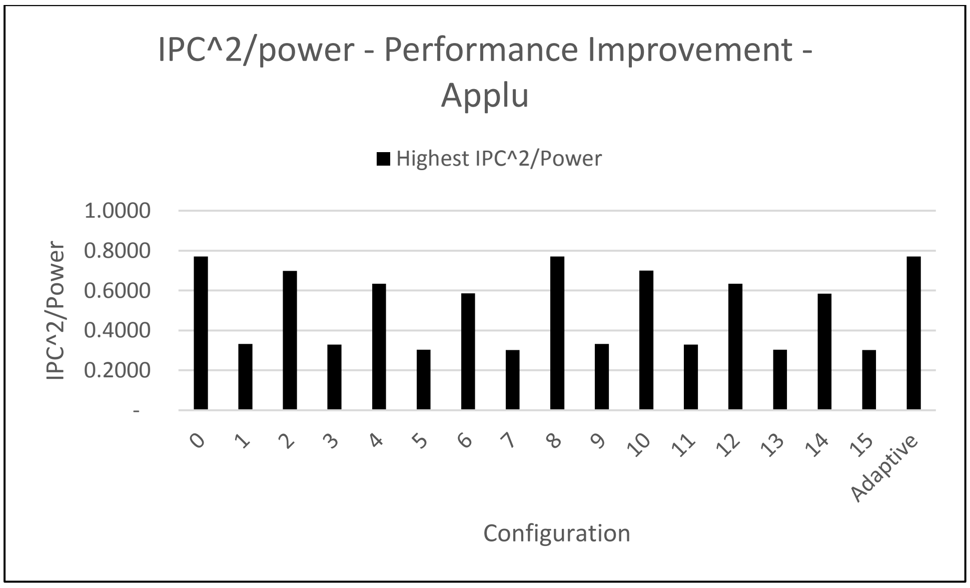
- Once the python scripts are ready we generate shell scripts files from it. The shell scripts are further executed to obtain the output files.
- I have used the grep command to get the IPC values from the output files.
  'grep Instructions applu0 > result_applu0'
  This above line gets the line containing the word Instruction from the file applu0 and stores the lines in to the file result_applu0.
- Thus, I have generated 16 such text files for every workload containing only the IPC values for the workload.

- Further, all these values are copied onto a excel file with separate sheet for every benchmark.
- Next, we find dynamic and static power for all the configurations and have a separate coloumn for IPC/power and IPC^2/power.
- Further, we use implicit excel functions to transverse the table and find the highest IPC/power values and generate the graphs for it.

Results:

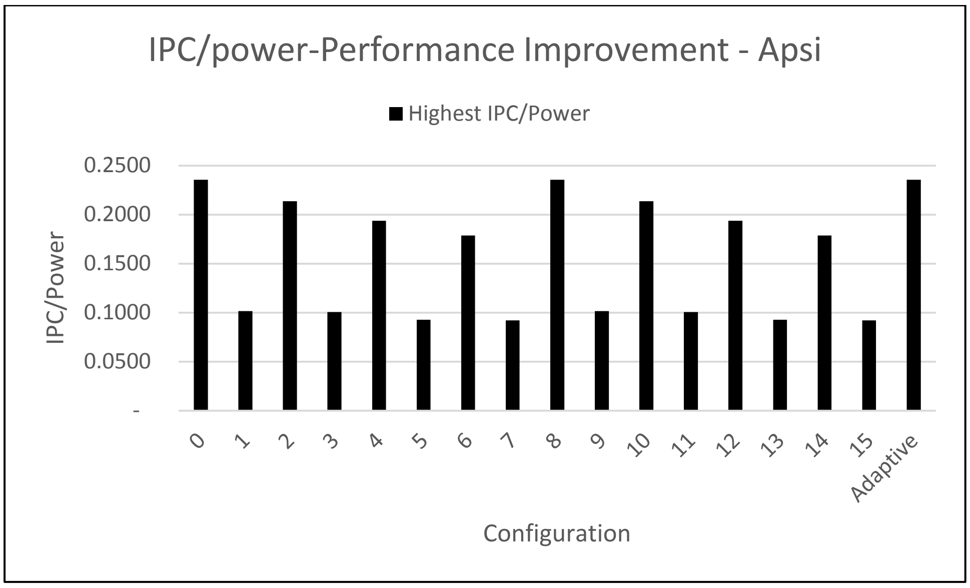
applu-

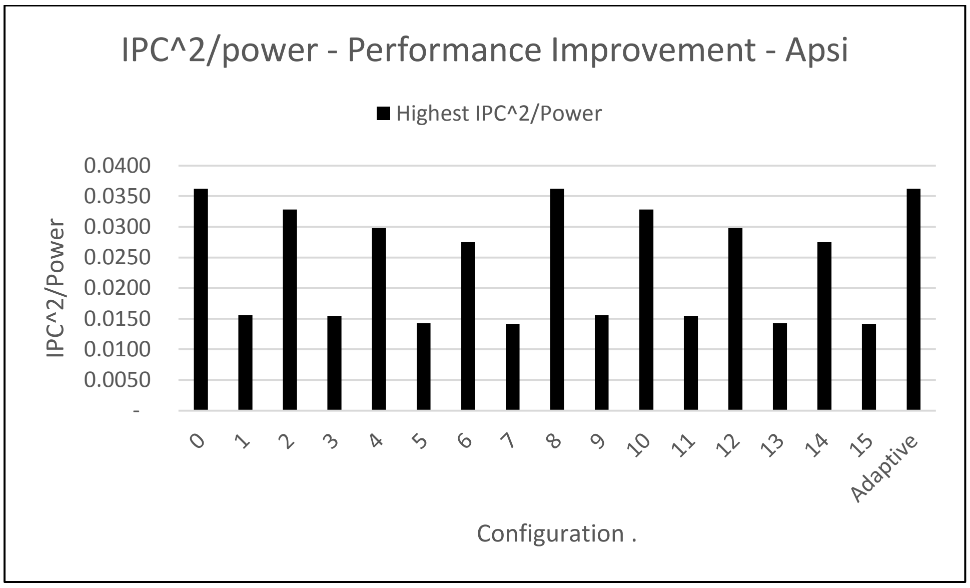




For Applu benchmark, it can be observed that the efficient configurations for the IPC/power performance comparison the best configurations are 0 and 8 . Configuration '0' has L1 size of 32 kb and associativity of 1 while configuration 8 has L1 size as 64 kb and associativity of 4.. the configuration 0 mathces the values of the adaptive configuration for tboth the target functions. It can be also observed that the performance function decreases when the L2 associativity is made 8.

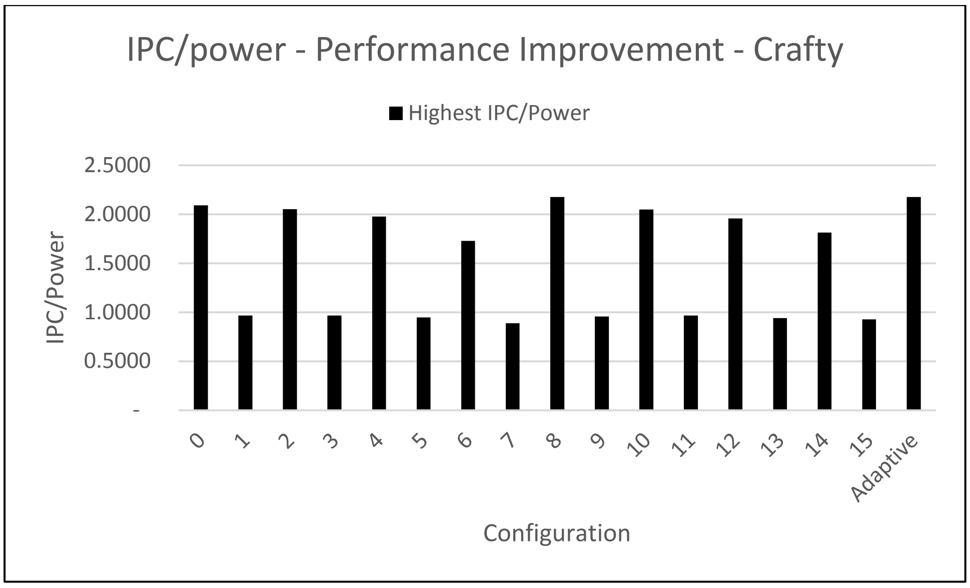
Apsi:

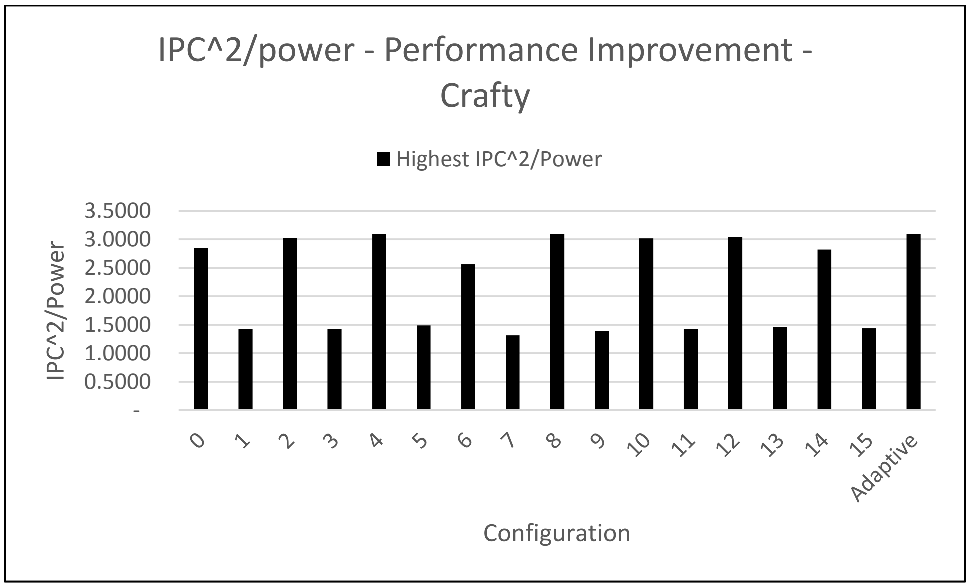




From the above two graphs showing performance improvement for the apsi workload, it can be observed that the most efficient configurations can be said to be configuration 0 and configuration 8 . Also the adaptive configuration matches the value of both the target functions to the value of configuration 8 and configuration 0. An increase in associativity of L2 cache can be said to inversely proportional to increase in Efficiency as we can observe that whenever we increase associativity of L2 there is a significant drop in the target functions.

Crafty:





For the crafty workload, the configuration number that has the best performance ratio and the better efficiency is 4 and 8. This configuration correspond to L1 size of 32kb and associativity 2 and L2 size 128kb and associativity as 4 . Configuration 8 corresponds to L1 size as 64kb and associativity 1 and L2 size as 128kb and associativity as 4. It can also be observed that there are more number of configurations with values of target functions closer to that of the adaptive configuration than there were for applu and apsi workloads.

Conclusion:

This project explains the importance of dynamically varying cache size and associativity of L1 and L2 cache during various phases of execution of an application to obtain increased performance and better efficiency. The project also gave a better insight into workings of a microprrocessor. The SMTSIM was better understood owing to the fact that we had to make changes to source code of the SMTSIM and had to make the SMTSIM again. The target functions give us the power requirements of the application during different stages of its execution. Also, dynamically changing the cache size and cache associativity might not always be a good solution as it might be prove to be less power efficient for some types of workloads.

Note:

- I have not used a python script to generate the configurations, hence the python to generate configuration has not been included. I did the 16 configurations manually

- Also, grep command was used to get the IPC values from output files. Hence the python scripts for the same have also not been included. Grep command used has been discussed in the implementation.