Optimal Cache Design

Project I – Computer Architecture CE-6304

Gunjan Munjal - 2021366660
Sandeep Govindaraj - 2021378846
Tanya Tukade - 2021388622

Introduction:
A cache is a component that stores data so future requests for that data can be served faster. The data present in the cache might be the result of an earlier computation, or the duplicate of data stored elsewhere. To enable efficient use of data and be cost-effective, caches are relatively small.

In this project, we fine-tune the cache hierarchy on X86 architecture based on the gem5 simulator for 5 different benchmarks namely 401.bzip2, 429.mcf, 456.hmmer, 458.sjeng, 470.lbm considering multiple parameter tradeoffs and performance characteristics.

The report consists of 5 parts according to the project description.

Aim
The project aims to find the optimal cache design configuration whilst running it on multiple benchmark by simultaneously varying different design specifications.

Goal
The goal is to not only find the minimum CPI but also to design a cost function which minimizes the cost the of the design.

Part-1:
For our project, we accessed the Gem5 on server. In this part, the functionality of the gem5 simulator is examined and the following experiment is run.

Part-2:
We calculated the Cycles per Instruction (CPI) for a set of 5 benchmarks with the command line provided in the project description, and modifications needed for each benchmark.

Our baseline X86 configuration is setup as follows: -
CPU Models: Timing Simple CPU (timing).
Cache levels: Two levels.
Size: 128KB L1 data cache, 128KB L1 instruction cache, 1MB unified L2 cache.
Associativity: Two-way set-associative L1 caches, Direct-mapped L2 cache.
Block size: 64 bytes (applied to all caches).

*Given an L1 miss penalty of 6 cycles, L2 miss penalty of 50 cycles, and one cycle cache hit/instruction execution. The following is the formula used for the calculation of the CPI.

CPI = 1 + ((IL1.miss_num+DL1.miss_num)×6+(L2.miss_num×50))/(Total _ Inst _ num)

The values of the calculated CPI is tabulated below

Benchmark	**CPI**
401.bzip2	1.32266
429.mcf	1.1098
456.hmmer	1.003092
458.sjeng	1.93
470.lbm	1.8012789

Part-3:

This part involves exploring the design space and trying different configurations to find an optimal configuration of cache which provides the best performance. The optimization of the CPI based on various observations like cache size, associativity, etc. By providing different configurations the most suitable, optimized cache design is found for all 5 benchmarks with CPU model as timing and other trade off features like associativity, cache levels etc.

The CPI values for each of the L1 and L2 cache are calculated based on the following parameters:

1. L1 Associativity - 1, 2, 4, 8
2. L2 Associativity – 1
3. CPU Models – timing
4. Block Size – 64
5. L1 Size – 64kB, 128kB, 256kB
6. L2 Size – 2MB, 4MB

The formula used for the calculation of the CPI is as follows

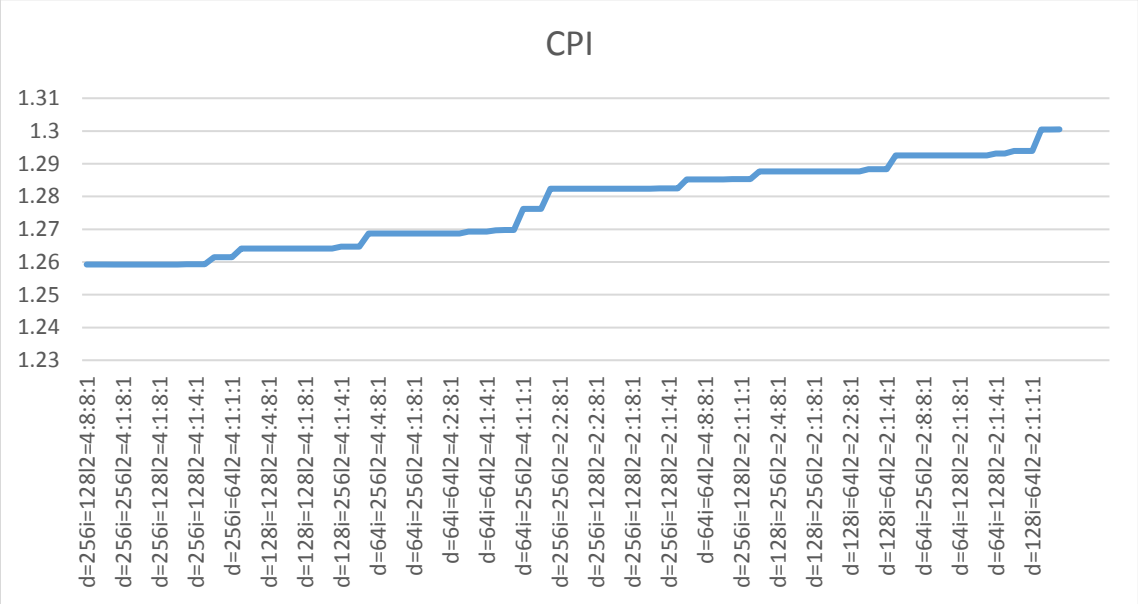
CPI = 1 + ((IL1.miss_num+DL1.miss_num)×6+(L2.miss_num×50))/(Total _ Inst _ num)

The CPI for various cache design combinations for each benchmark have been tabulated and plotted as follows.

Benchmark	Max CPI	Min CPI
401.bzip2	1.30048932	1.259233736
429.mcf	1.73233	1.53109
456.hmmr	1.010496	1.000806
458.sjeng	1.965650168	1.93742112
470.lbm	1.802204	1.800779

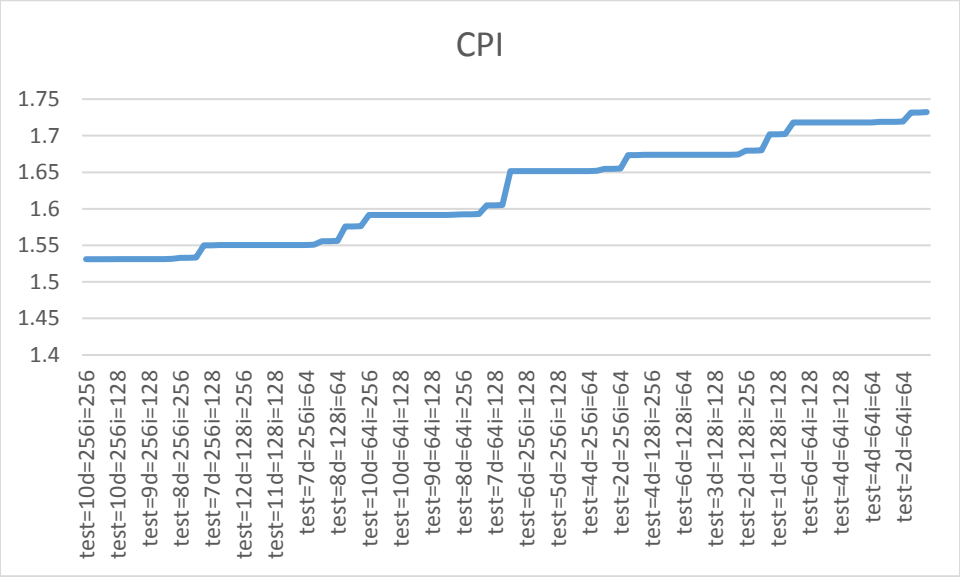
The plot is a CPI vs Iterations. The plot showcases the variation of CPI per Iteration.

401.bzip2-timing:
Graph:



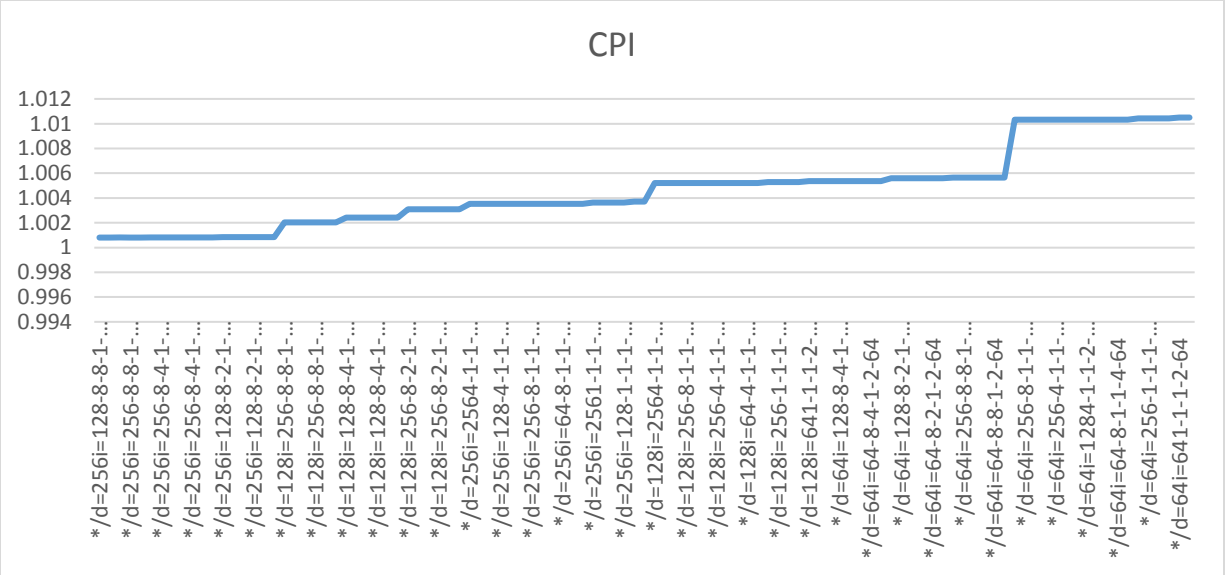
Configuration of optimal CPI:
CPU type: Timing
L1 D Cache size = 256KB
L1 I Cache size = 128KB
L2 Cache size = 4MB
Block size = 64KB
L1 I Associativity = 8
L1 D Associativity = 8
L2 Associativity = 1
Optimal CPI = 1.259233
Justification for optimal configuration:
The configuration is chosen optimum because it uses high L1i and L1d configurations and the higher associativity of L1 cache and higher L2 size reduces the CPI of this benchmark to the least value.

429.mcf-timing:
Graph:



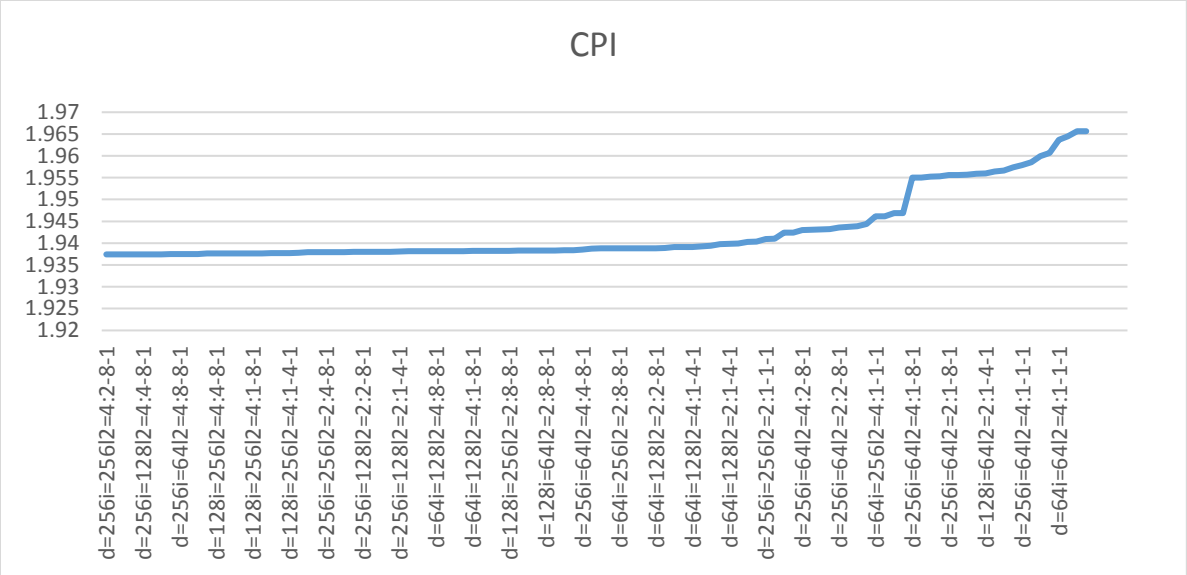
Configuration of optimal CPI:
CPU type: Timing
L1 D Cache size = 256KB
L1 I Cache size = 256KB
L2 Cache size = 4MB
Block size = 64KB
L1 I Associativity = 2
L1 D Associativity = 8
L2 Associativity = 1
Optimal CPI = 1.5310907
Justification for optimal configuration:
The configuration is chosen optimum because it uses equal L1i and L1d configurations and the highest associativity of l1 and higher size of l2 reduces the CPI of this benchmark to the least value.

456.hmmer-Timing:
Graph:



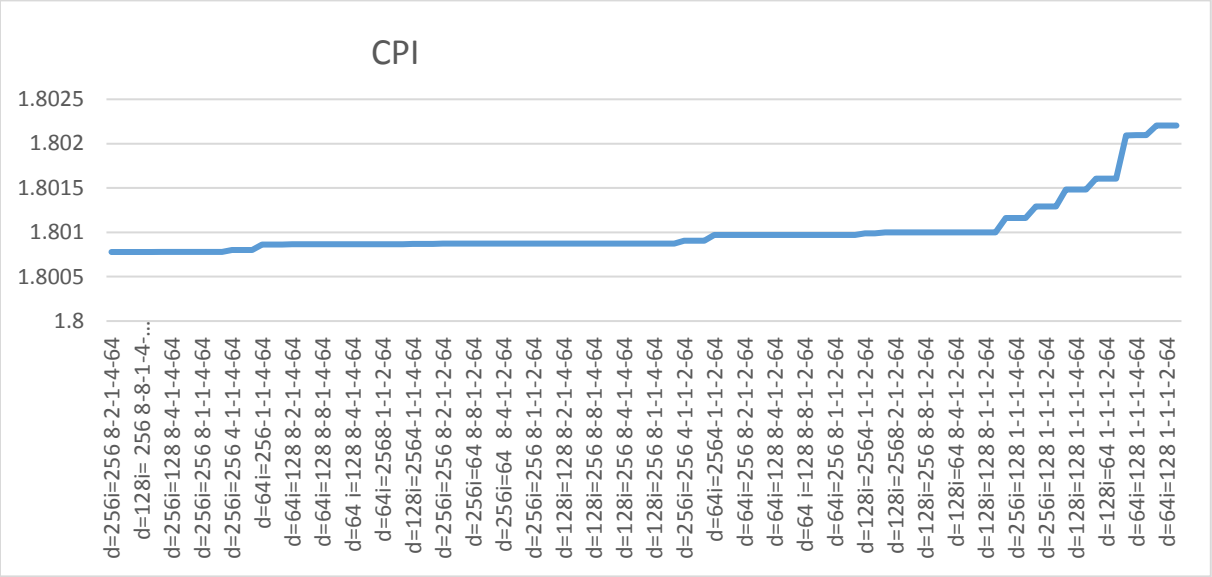
Configuration of optimal CPI:
CPU type: Timing
L1 D Cache size = 256KB
L1 I Cache size = 128KB
L2 Cache size = 2MB
Block size = 64KB
L1 I Associativity = 8
L1 D Associativity = 8
L2 Associativity = 1
Optimal CPI = 1.000805524
Justification for optimal configuration:
The configuration is chosen optimum because it uses high L1i and L1d configurations and the higher and equal associativity of L1 cache reduces the CPI of this benchmark to the least value.

458.Sjeng-Timing:
Graph:



Configuration of optimal CPI:
CPU type: Timing
L1 D Cache size = 256KB
L1 I Cache size = 256KB
L2 Cache size = 4MB
Block size = 64KB
L1 I Associativity = 8
L1 D Associativity = 2
L2 Associativity = 1
Optimal CPI = 1.93742112
Justification for optimal configuration:
The configuration is chosen optimum because it uses equal L1i and L1d configurations and the higher value of l2 size and higher associativity of l1 cache reduces the CPI of this benchmark to the least value.

470.lbm-timing:
Graph:



Configuration of optimal CPI:
CPU type: Timing
L1 D Cache size = 256KB
L1 I Cache size = 256KB
L2 Cache size = 4MB
Block size = 64KB
L1 I Associativity = 8
L1 D Associativity = 2
L2 Associativity = 1
Optimal CPI = 1.8007787
Justification for optimal configuration:
The configuration is chosen optimum because it uses equal L1i and L1d configurations and the highest l2 cache size and higher associativity of l1cache reduces the CPI of this benchmark to the least value.

Part 4: Cost Function:

Cost Table:

Particular	Values	Cost Assigned
L1_d Associativity	1	1
	2	2
	4	4
	8	8
L1_i Associativity	1	1
	2	2
	4	4
	8	8
L2 Associativity	1	1
L1_d Cache Size	64 kB	10
	128 kB	20
	256 kB	40
L1_i Cache Size	64 kB	10
	128 kB	20
	256 kB	40
L2 Cache Size	2 MB	80
	4 MB	160
Cache Line	64	10

Assumptions:

1. Cost assigned are purely fictitious and do not reflect any dependency on the market.
2. A thumb Rule of cache cost: The cost doubles when size increases for both L1 cache size, L2 cache size and associativity.
3. L2 associativity has same cost as L1 associativity.
4. Considering that the cache is responsible for 70% of the machine cost, we have formulated the cost function for just the cache and not the entire machine.

Weighted cost for total cost of cache

Particular	Weighted average
L2 Associativity	20%
L1_I Associativity	20%
L1_D Associativity	20%
L1_I cache size	30%
L1_ D cache size	30%
L2 cache size	30%
Cacheline	50%

Total Cost = 0.2 * (L2_ Associativity + L1_I Associativity +L1_D Associativity)+0.3*(L1_I cache size + L1_D cache size + L2 cachesize)+0.5*Cacheline size

Equation Used to normalize cost:

$$\mathbf{cost_{norm}} = \frac{\mathbf{X(i) - Cost_{min}}}{\mathbf{cost_{max} - cost_{min}}}$$

Equation Used to Normalize CPI:

$$\mathbf{CPI_{norm}} = \frac{\mathbf{X(i) - CPI_{min}}}{\mathbf{CPI_{max} - CPI_{min}}}$$

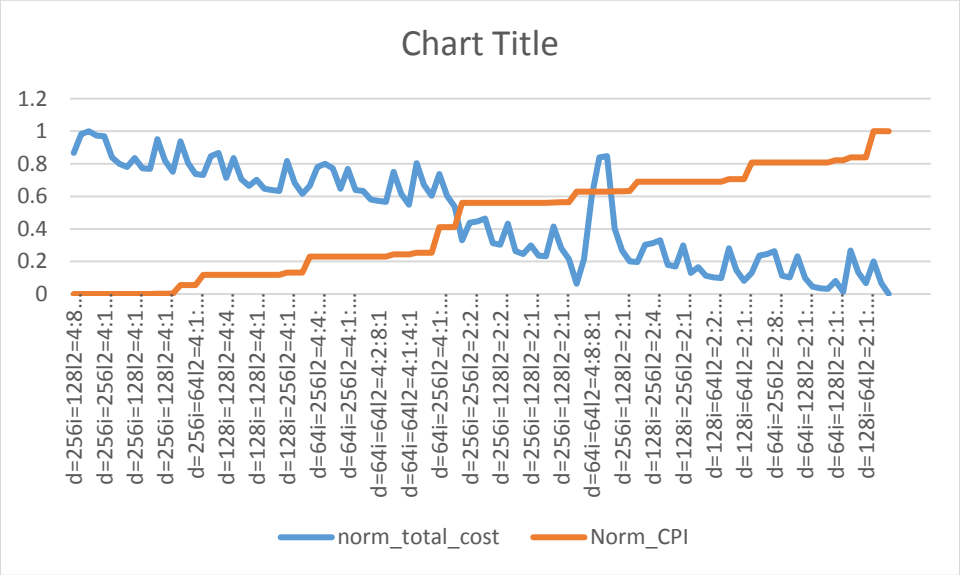
Part 5:

Optimize caches for performance/cost

- ***For each benchmark***

The normalized cost and normalized CPI are plotted against the iterations. The Value with minimum cost for minimum CPI is chosen for all Benchmarks.

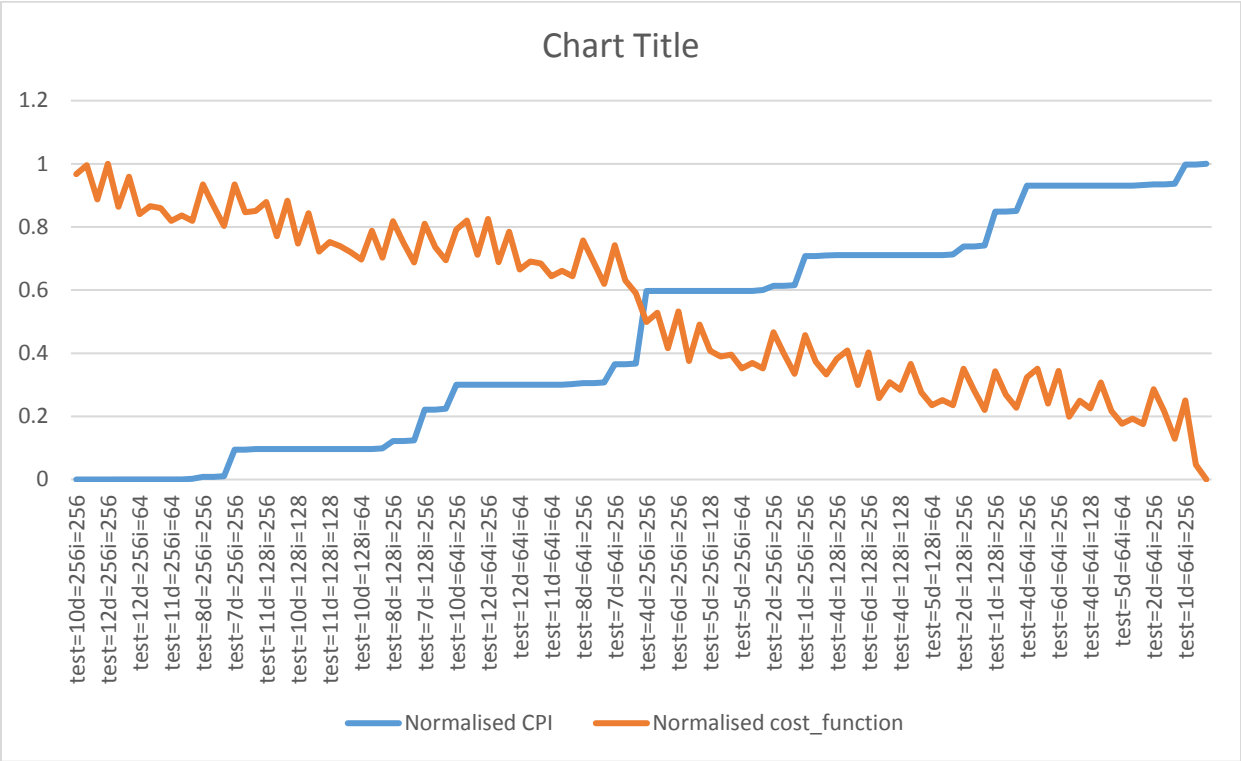
1. 401.bzip2



The corresponding values are:

CPU	CPI	L2 Assoc	L1_D Size	L1_I Size	L1_D Assoc	L1_I Assoc	L2 Size	CacheLine	Cost	Cost Normalized	CPI Normalized
Timing	1.27621692	1	64kB	64kB	1	1	4MB	64	86.6	0.535714286	0.411657826

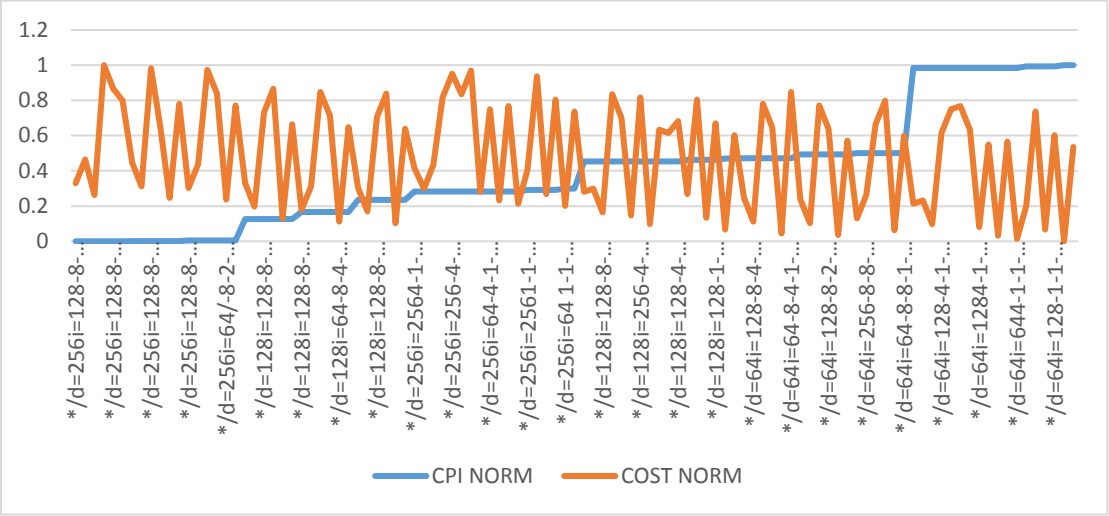
2. 429.mcf



The corresponding values are:

CPU	CPI	L2 Assoc	L1_D Size	L1_I Size	L1_D Assoc	L1_I Assoc	L2 Size	CacheLine	Cost	Cost Normalized	CPI Normalized
Timing	1.6513	1	256kb	256kb	8	2	4 MB	64	67.420	0.4987	0.5976

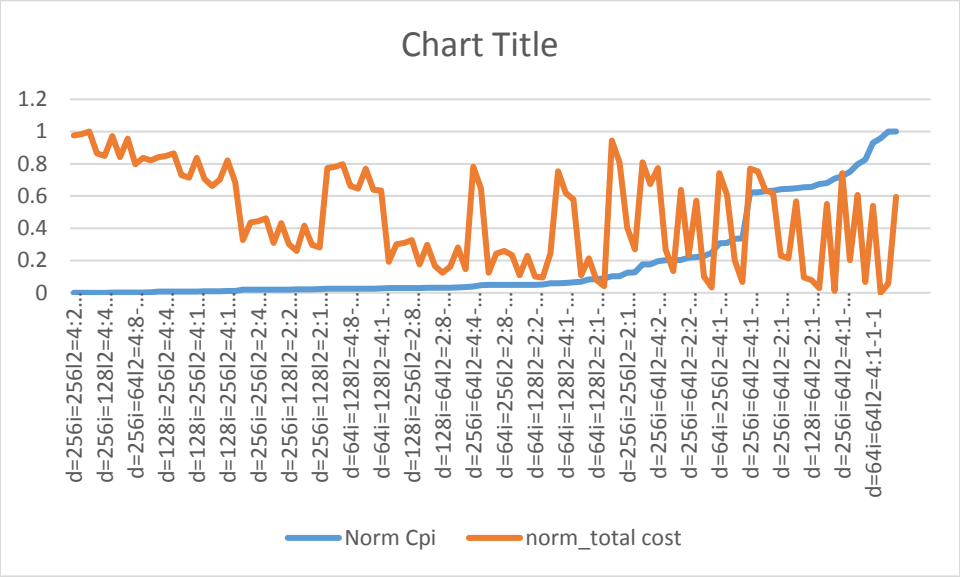
3. 458.hmmr



The corresponding values are:

CPU	CPI	L2 Assoc	L1_D Size	L1_I Size	L1_D Assoc	L1_I Assoc	L2 Size	CacheLine	Cost	Cost Normalized	Cpi normalized
Timing	1.002032	1	128kB	6kB	8	8	2MB	64	41.4	0.872730491	0.126537916

4. 456.sjeng

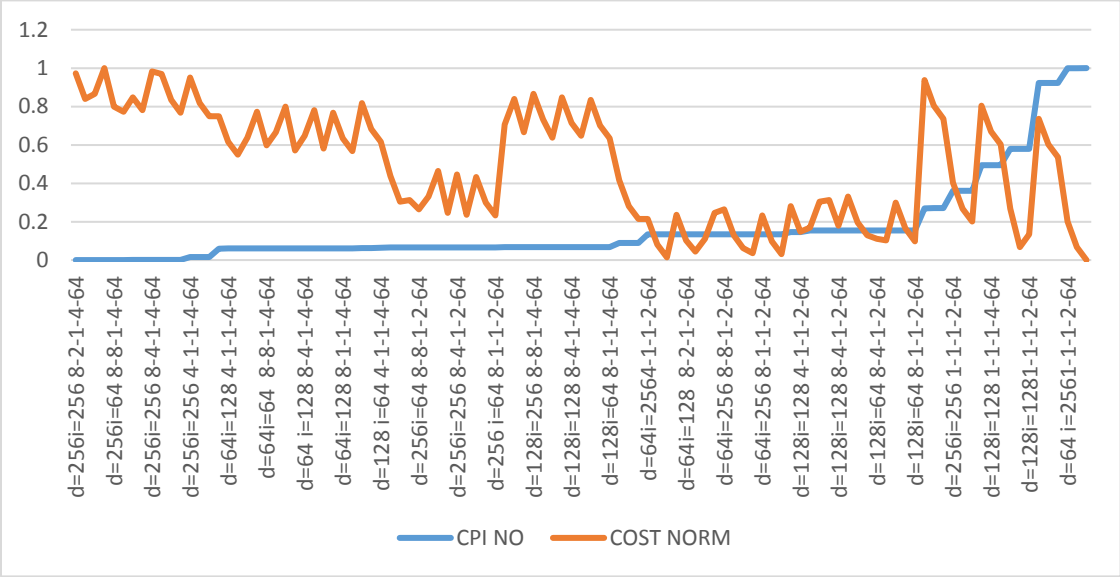


The corresponding values are :

CPU	CPI	L2 Assoc	L1_D Size	L1_I Size	L1_D Assoc	L1_I Assoc	L2 Size	CacheLine	Cost	Cost Normalized	CPI Normalized
Timing	1.939377476	1	128kB	64kB	4	8	2MB	64	32.45	0.107816712	0.069302939

5. 470.lbm

i) Timing



Iteration Choosen For timing is 62 nd iteration

The corresponding values are :

CPU	CPI	L2 Assoc	L1_D Size	L1_I Size	L1_D Assoc	L1_I Assoc	L2 Size	CacheLine	Cost	Cost Normalized	CPI Normalized
Timing	1.800969	1	64kb	128kb	4	1	2MB	64	39.2	0.080357143	0.133599

Cost Function Working:

How does the cost function work? And what does the graphs represent?

The values obtained from the iteration are arranged in ascending order of normalized CPI. The normalized values of CPI and Cost are plotted w.r.t to iterations, thus we obtain a graph in the range of 0 and 1.
Thus, for a given iteration we can see the normalized cost and normalized CPI for a particular configuration. Thus, the lower the CPI the higher the cost and vice versa. With the help of these graphs and after careful consideration of the cost and CPI parameters we decide a design configuration for each benchmark.

- ***Combined for all benchmarks***

From the graphs above we locate the optimal configuration which provides maximum performance. This is done by minimizing the cost and increasing CPI by a small amount.

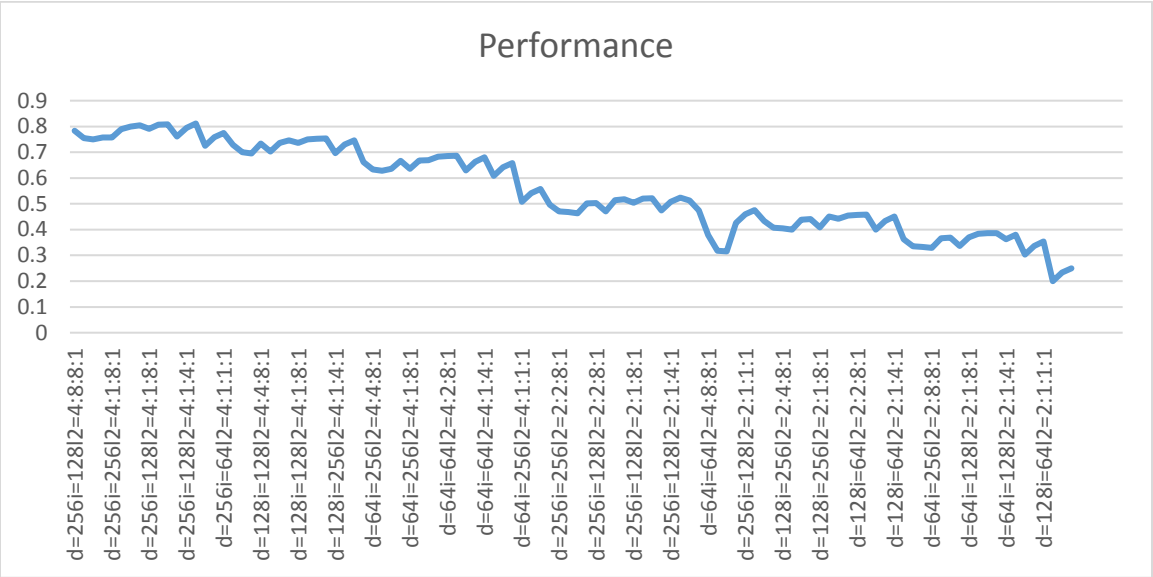
Performance is measure by providing a weighted measure of 75% of Normalized CPI and 25% of Normalized Cost.

Performance =1- (0.75% *Norm_CPI +0.25*Norm_Cost)

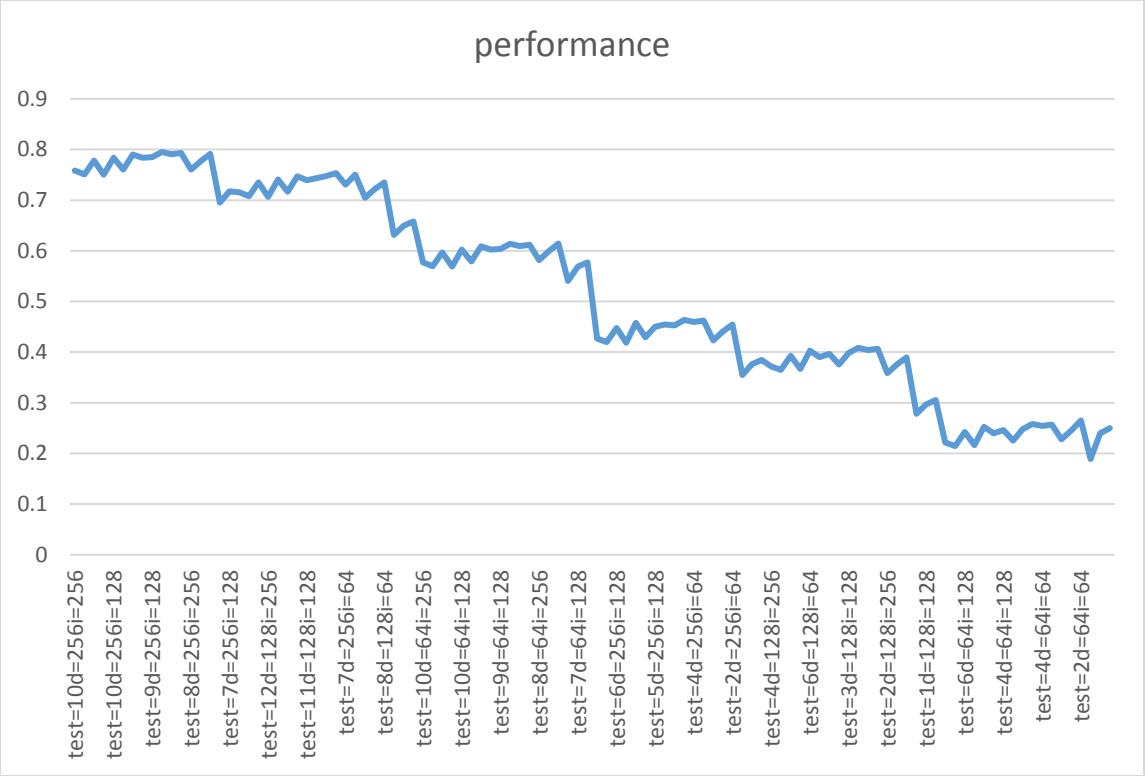
Note: Sum is subtracted from 1 because the CPI with highest performance is at 0. Thus the graph is inverted for laymen interpretation.

The performance graphs of all the benchmarks are as follows:

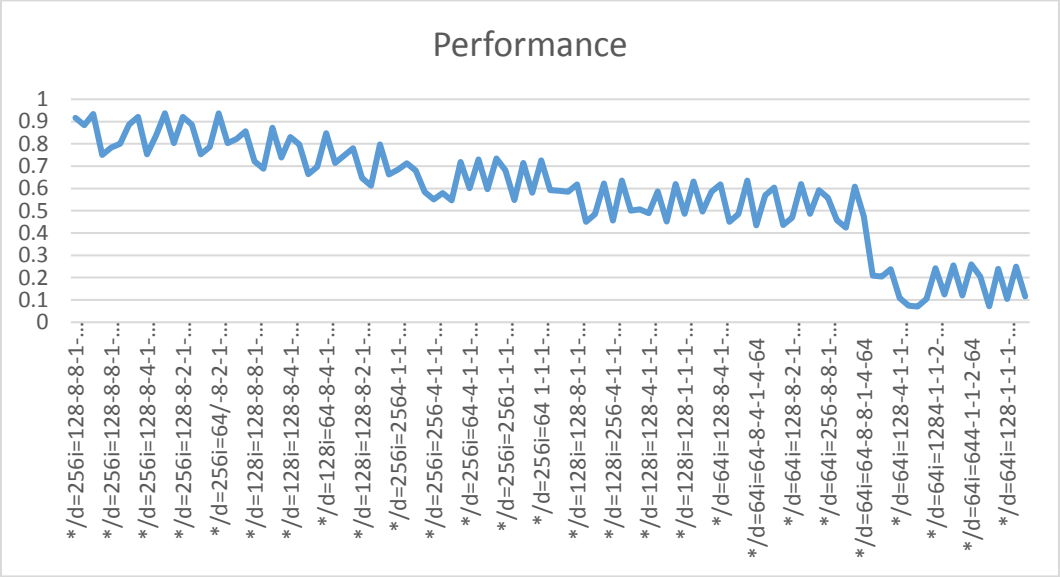
401.Bzip2



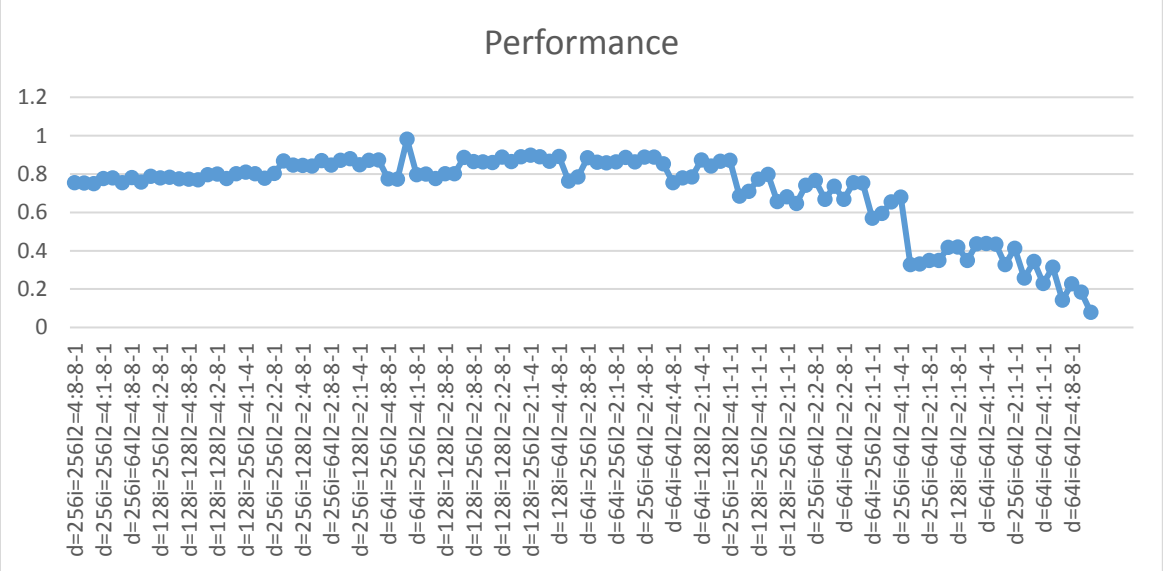
429.mcf



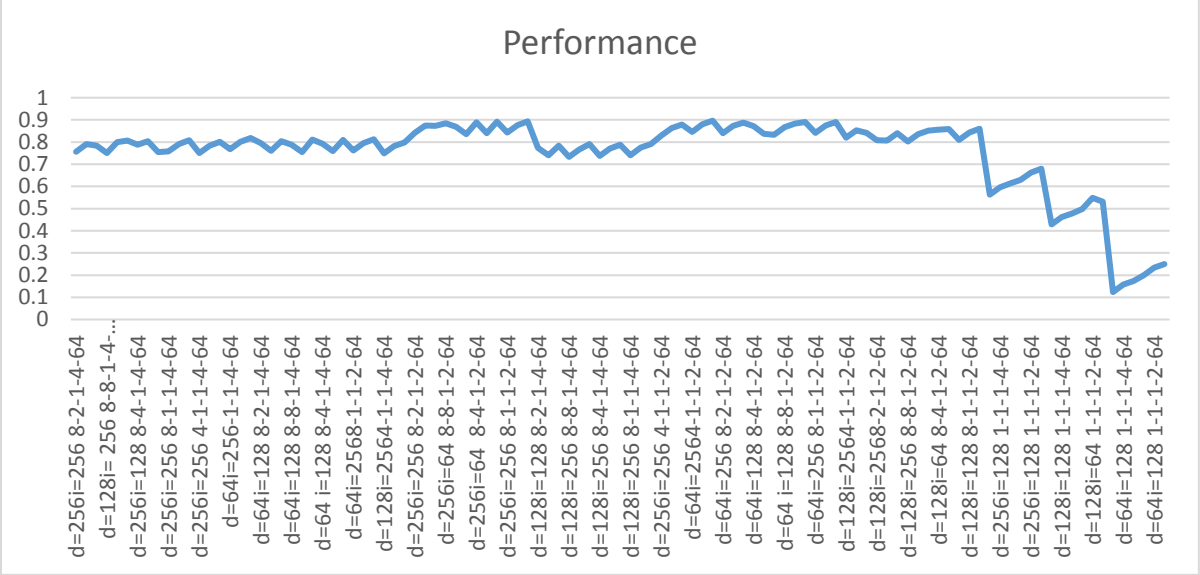
456.hmmr



458.sjeng

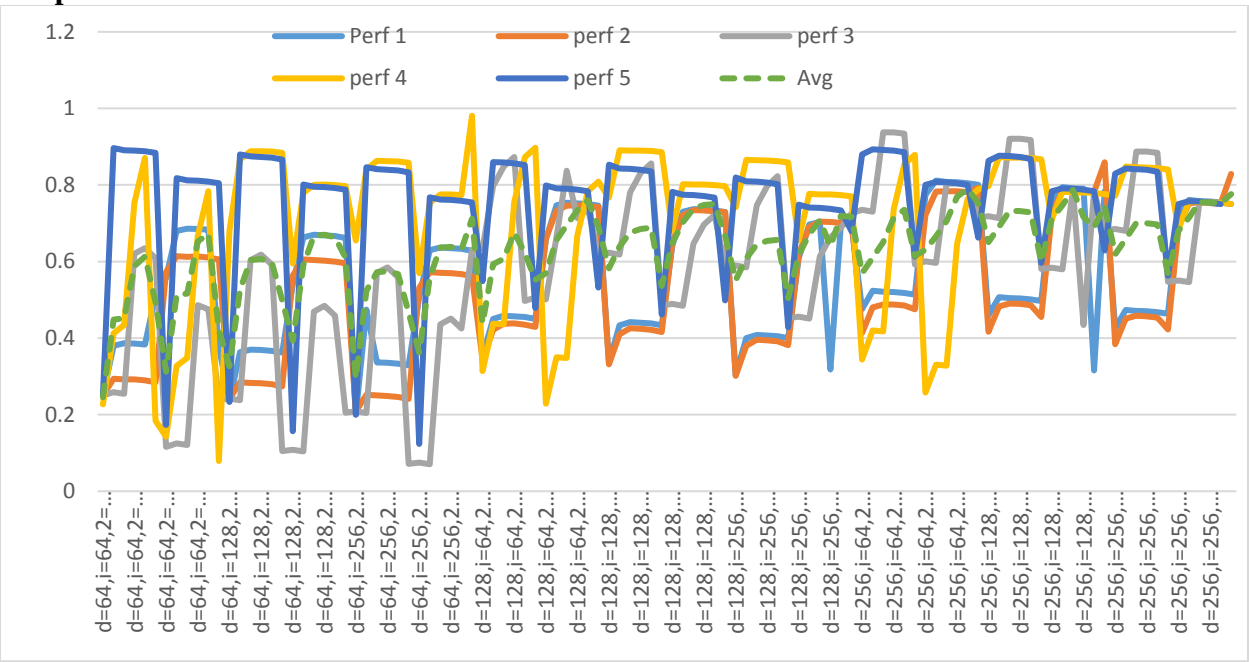


470.lbm:


Performance
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
d=256i=256 8-2-1-4-64
d=128i= 256 8-8-1-4-...
d=256i=128 8-4-1-4-64
d=256i=256 8-1-1-4-64
d=256i=256 4-1-1-4-64
d=64i=256-1-1-4-64
d=64i=128 8-2-1-4-64
d=64i=128 8-8-1-4-64
d=64 i=128 8-4-1-4-64
d=64i=2568-1-1-2-64
d=128i=2564-1-1-4-64
d=256i=256 8-2-1-2-64
d=256i=64 8-8-1-2-64
d=256i=64 8-4-1-2-64
d=256i=256 8-1-1-2-64
d=128i=128 8-2-1-4-64
d=128i=256 8-8-1-4-64
d=128i=256 8-4-1-4-64
d=128i=256 8-1-1-4-64
d=256i=256 4-1-1-2-64
d=64i=2564-1-1-2-64
d=64i=256 8-2-1-2-64
d=64i=128 8-4-1-2-64
d=64 i=128 8-8-1-2-64
d=64i=256 8-1-1-2-64
d=128i=2564-1-1-2-64
d=128i=2568-2-1-2-64
d=128i=256 8-8-1-2-64
d=128i=64 8-4-1-2-64
d=128i=128 8-1-1-2-64
d=256i=128 1-1-1-4-64
d=256i=128 1-1-1-2-64
d=128i=128 1-1-1-4-64
d=128i=64 1-1-1-2-64
d=64i=128 1-1-1-4-64
d=64i=128 1-1-1-2-64

Tradeoff between Designs for different Benchmarks (In terms of Performance)

Graph:



The above graph plots all the 5 benchmarks. From the graph, it is clear that the highest average performance is for the configuration given below. The performance is calculated using the equation discussed in the part 5 introduction. The performance here indicates that the benchmark offers low CPI and low cost at that particular iteration. Each iteration offers a unique cache design configuration.

CPU	CPI average	L2 Size	L1_D	L1_i	L1_d Assoc	L1_I Assoc	L2 Assoc	Cacheline	Cost average
Timing	0.789991752	4	256kB	64kb	4	8	1	64	72.37

Tradeoff: It is very clear from the graphs of the part 3 that CPI has a tradeoff with cost. For a normalized CPI of 0 , the normalized cost is mostly 1. This is because the lowest CPI is attained with high design specifications. We may not need that high specifications or lowering the specifications of some design parameters may not affect the CPI. Understanding this behavior of the CPI vs Cost will lead us to the optimal Design configuration.

Conclusion:

The cost of the design variables are assigned and weighted accordingly to calculate the total cost. With the help weighted data, we calculate the performance of the design for a benchmark and later is compared for different benchmarks for a CPU to make a conclusion of stable Design configuration. This configuration is achieved by understanding the tradeoff between Minimum CPI and Minimum Cost.