



**Σχολή Ηλεκτρολόγων Μηχανικών**

**&**

**Μηχανικών Υπολογιστών**

**Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Άσκηση Πρώτη**

**Μανδηλαράς Νικηφόρος, Α.Μ: 03112012**

**Όγδοο Εξάμηνο**

**Παραδοτέα: 3/ 4 /2016**



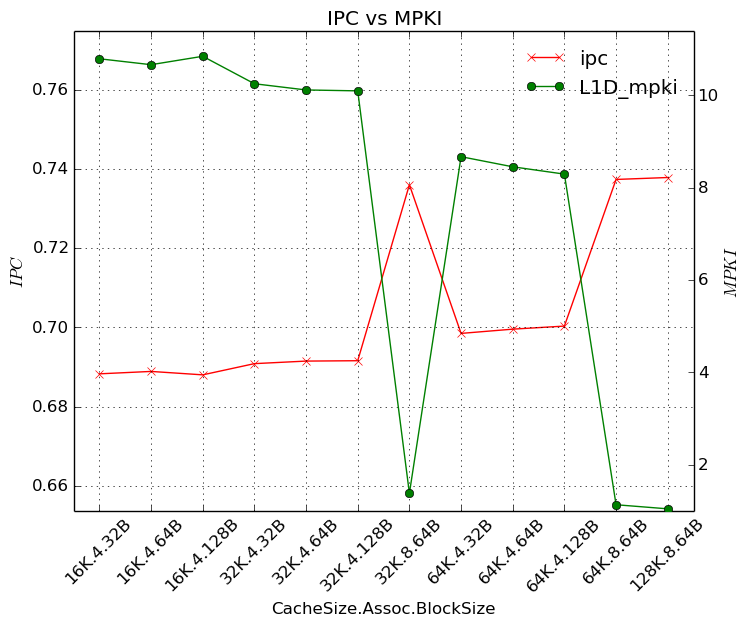
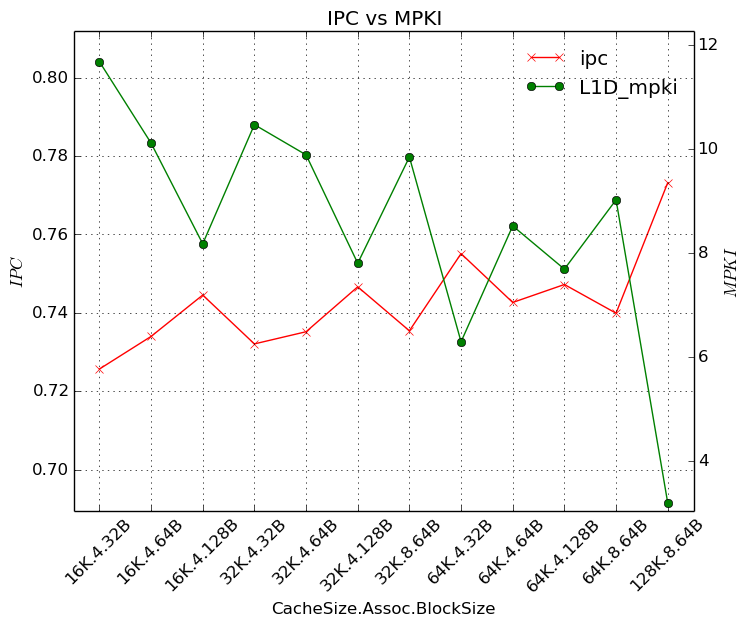
**Περιγραφή και Σκοπός της Άσκησης**

Σ’ αυτή την άσκηση καλούμαστε να εκτελέσουμε μια σειρά μετρήσεων προκειμένου να εκτιμήσουμε την επίδραση των βασικότερων παραμέτρων της ιεραρχίας μνήμης όσο αφορά την απόδοση. Για την υλοποίηση αυτών χρησιμοποιήθηκε το εργαλείο PIN καθώς και μια σειρά μετροπρογραμμάτων.

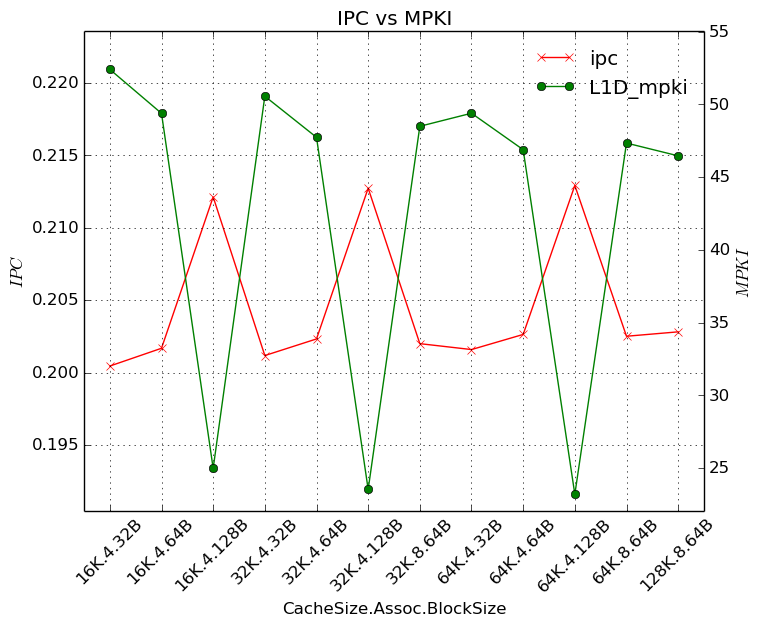
**7.1.1 L1 Cache**

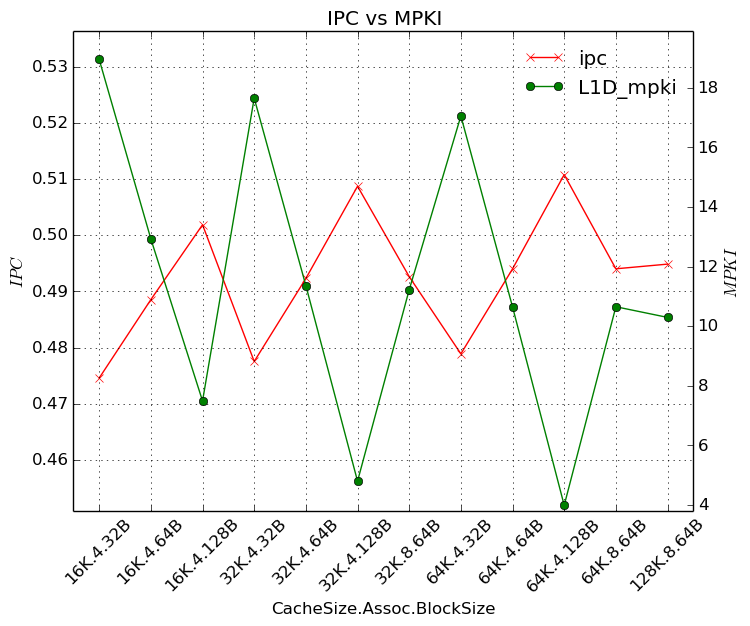
Στο πρώτο κομμάτι της άσκησης μελετήθηκε το πως επηρεάζουν την απόδοση ( Instructions Per Cycle) οι παράμετροι size , associativity , και size block της L1 . Για αυτό το λόγο κρατήθηκαν σταθερές όλες οι άλλες παράμετροι και πήραμε μετρήσεις για διάφορα configurations της L1. Συγκεκριμένα οι παράμετροι της L2 ήταν size:1024KB, associativity:8 size\_block:128B, η πολιτική αντικατάστασης LRU και η ιεραρχία μνήμης WRITE ALLOCATE.

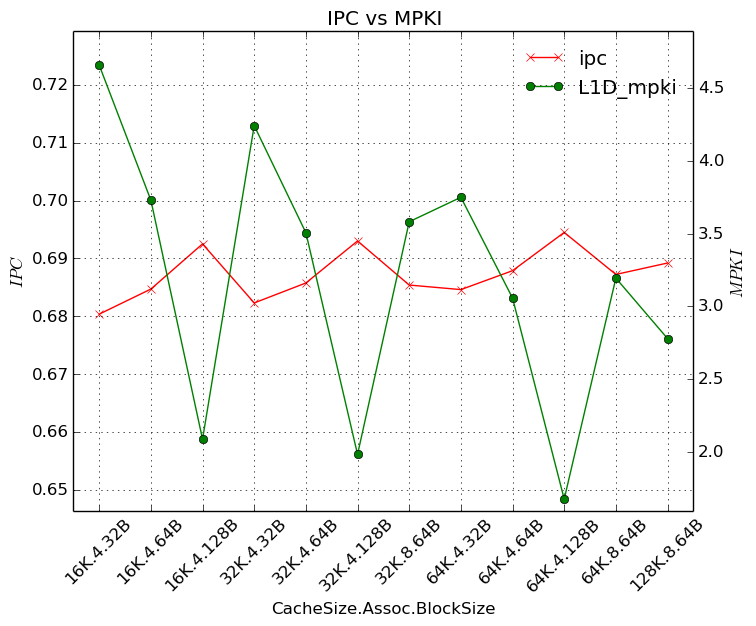
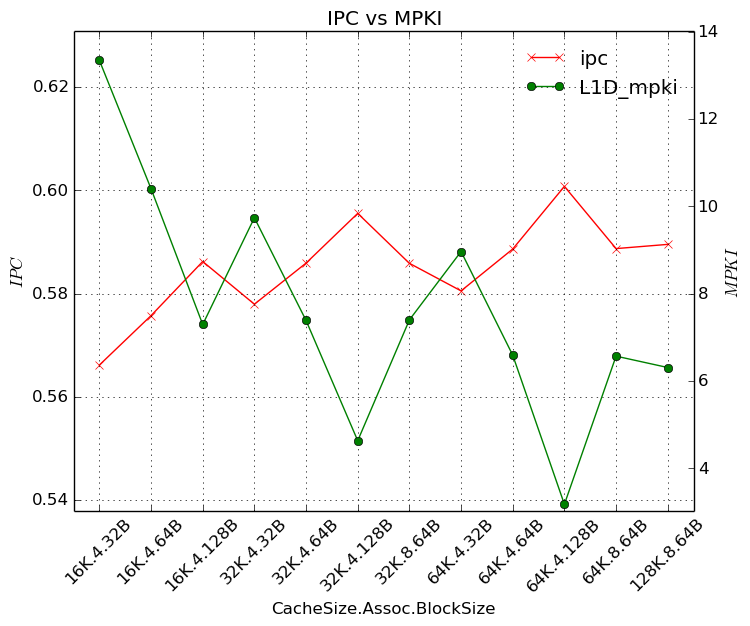
Στις γραφικές παραστάσεις που ακολουθούν φαίνεται για κάθε μετροπρόγραμμα οι τιμές του IPC ανά configuration της L1 καθώς και οι αντίστοιχες τιμές για το miss cache rate. Στις περισσότερες από αυτές μπορούμε να δούμε ότι το IPC παρουσιάζει μια αντίστροφη σχέση με το miss raτe.

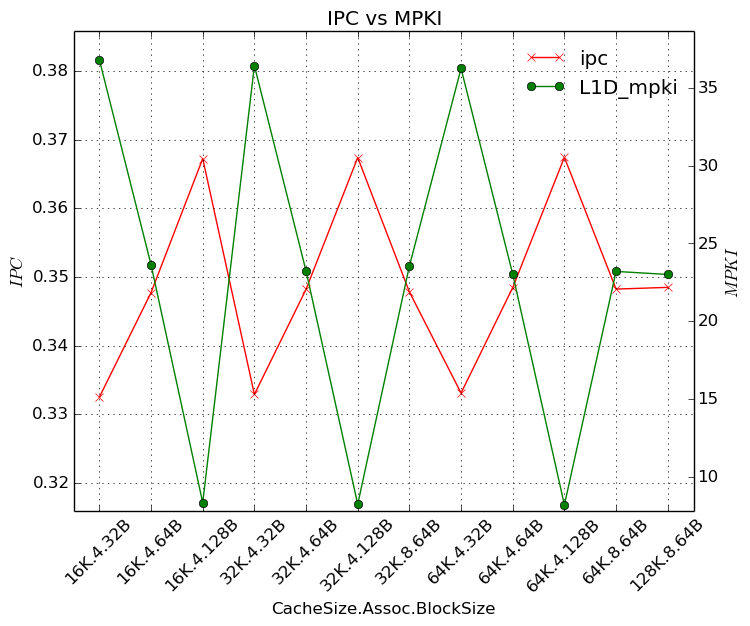
  
Η παραπάνω γραφική αφορά το μετροπρόγραμμα Blackscholes. Για αυτό το μετροπρόγραμμα το καθοριστικό στοιχείο στην απόδοση είναι η παράμετρος associativity καθώς όπου είχαμε 8 set assosiativity το IPC αυξήθηκε κατά πολύ περισσότερο από οποιαδήποτε άλλη αλλαγή.  
Η γραφική για το μετροπρόγραμμα bodytrack δείχνει να επιρεάζεται περισσότερο από το μέγεθος του block και επίσης από το μέγθος της cache. Το assosiativity φαινεται να επιδρά αρνητικά.

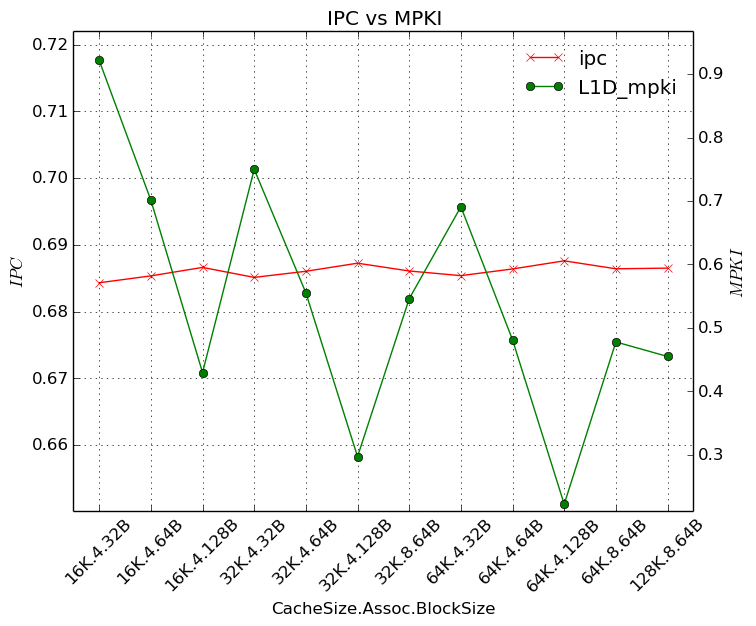
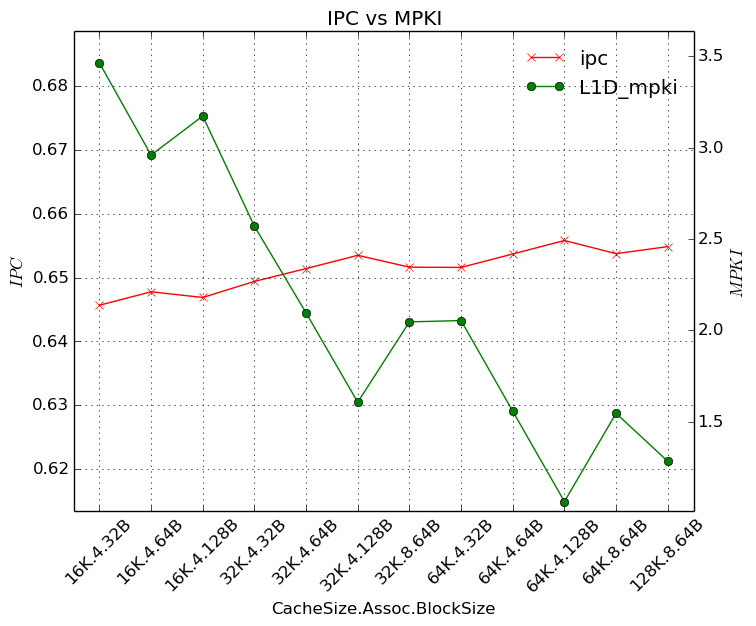
Για το μετροπρόγραμμα canneal που ακολουθεί βλέπουμε ότι οι τιμές του Ipc είναι πολύ χαμηλότερες σε σχέση με τα προηγούμενα. Εδώ τη μεγαλύτερη διαφορά στην απόδηση την κάνει η αύξηση του μεγέθους του block.



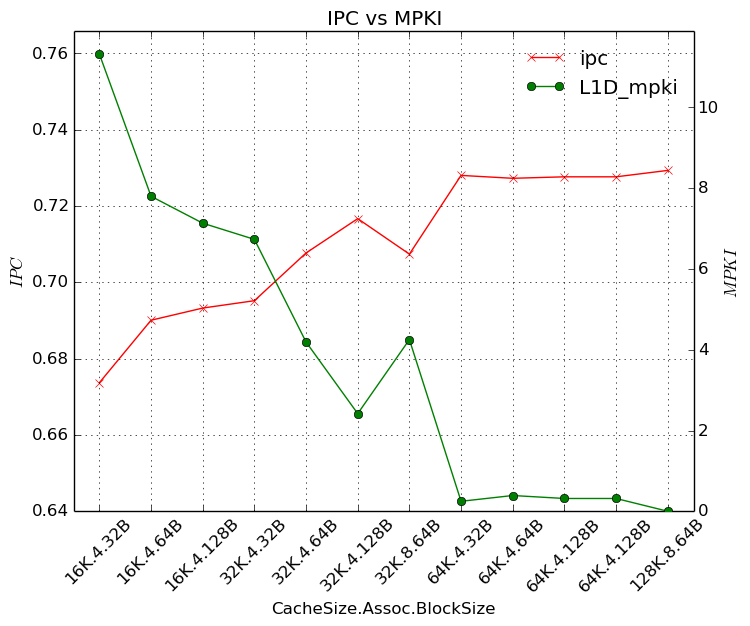


Για το facesim βλέπουμε ότι η άνοδος του IPC εξαρτάται κυρίως από το μέγεθος του μπλόκ (όσο μεγαλύτερο τόσο καλύτερη απόδοση ) και σε ένα μικρότερο βαθμό από το συνολικό μέγεθος.

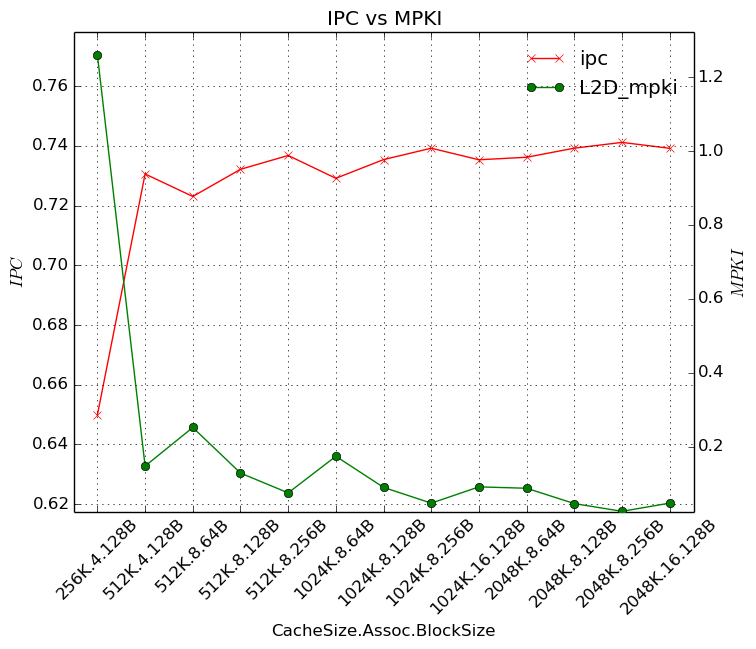
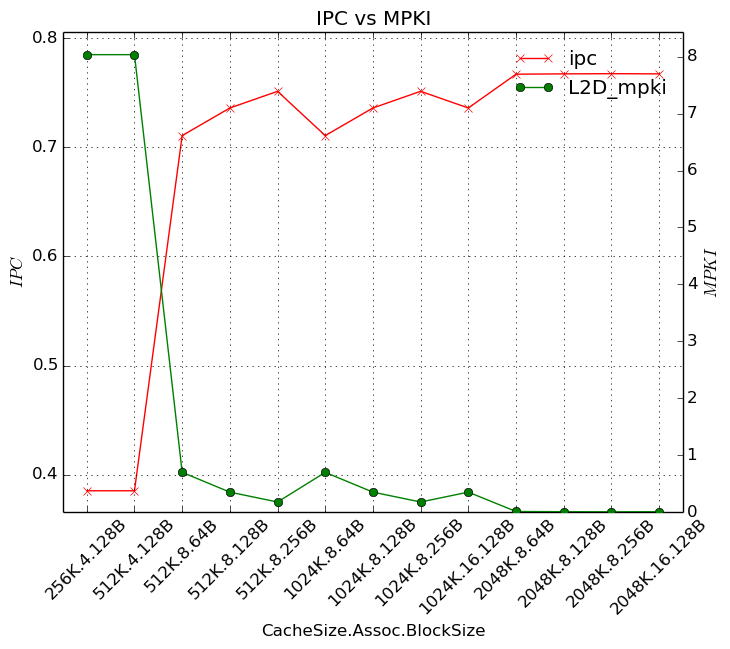
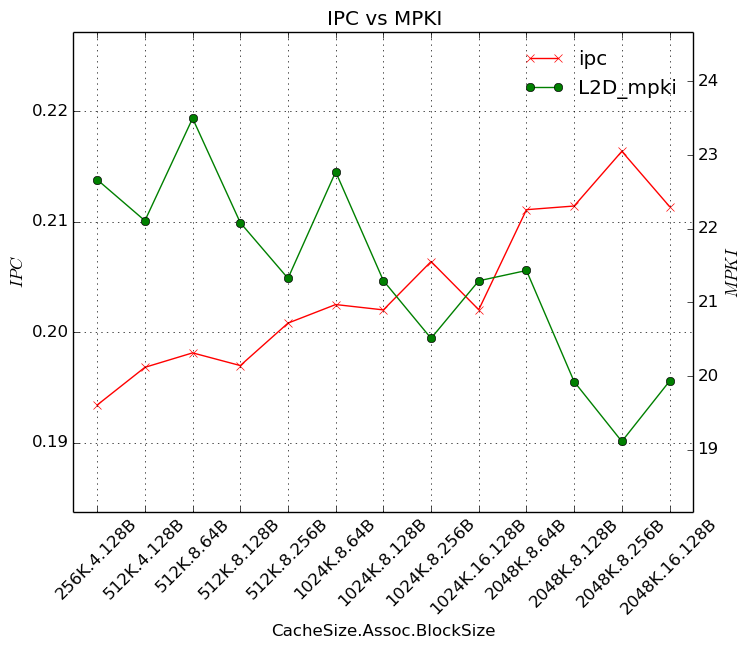
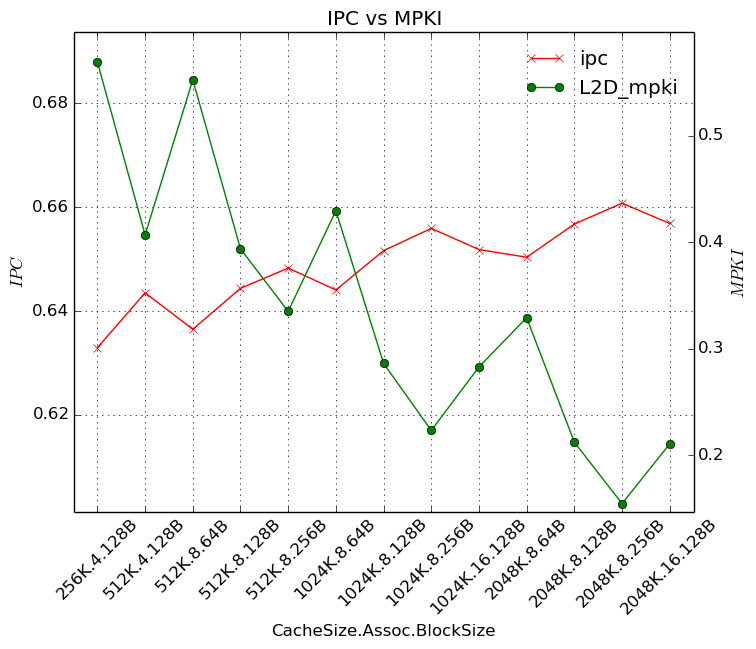


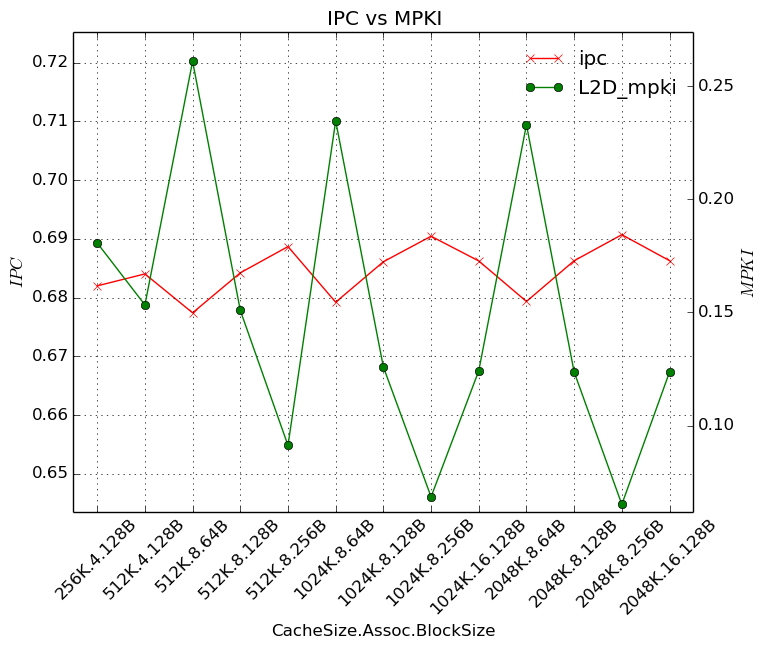
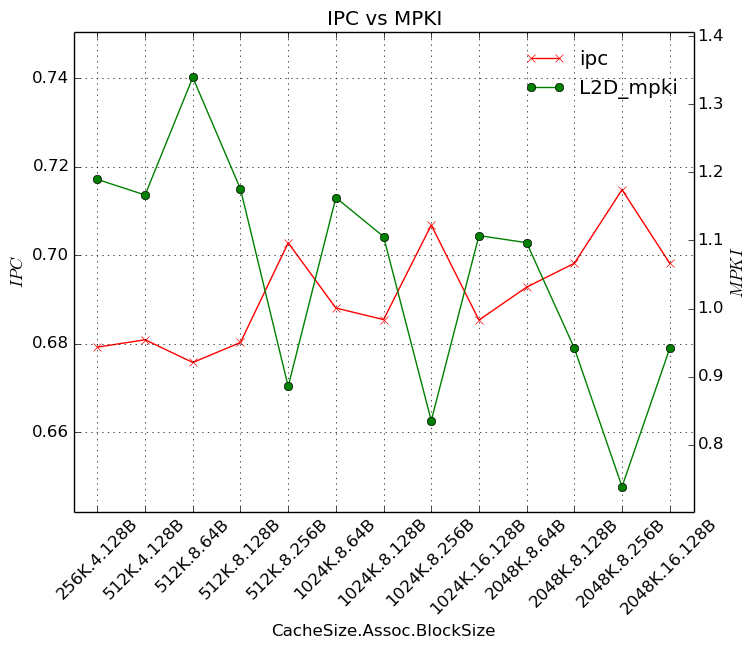
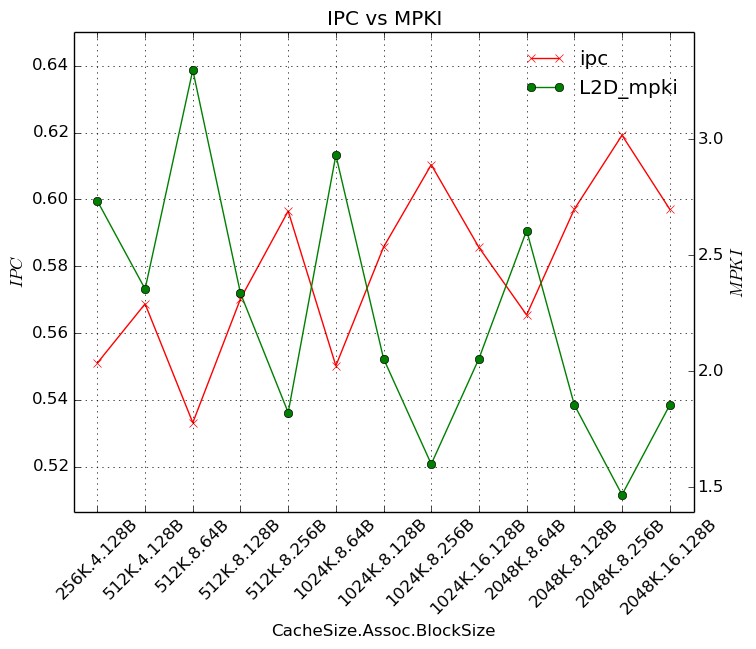
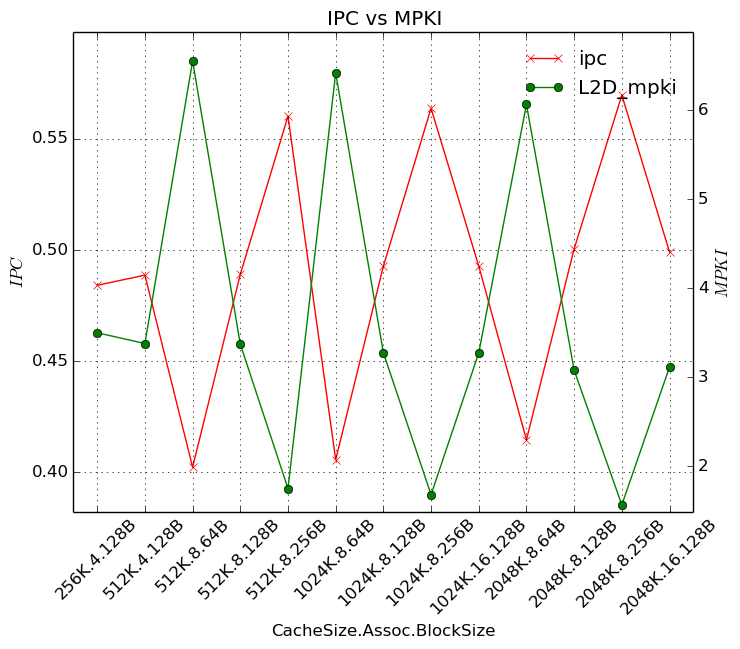
Στις τρεις παραπάνω γραφικές παραστάσεις για τα ferret , fluidanimate, streamcluster βλέπουμε μια παρόμοια κατάσταση με πριν. Ουσιαστικά βλέπουμε την ίδια καμπύλη για τις διάφορες τιμές του μπλόκ μόνο που κάθε φορά που αυξάνουμε το συνολικό μέγεθος της μνήμης η καμπύλη αυτή ξεκινά από μια μεγαλύτερη τιμή του ipc, και επειτα συνεχίζει να ανεβαίνει όσο αυξάνει το μέγεθος του μπλοκ. Σ όλες τις γραφικές παραστάσεις μέχρις στιγμής τα miss που συμβαίνουν συνεχίζουν να ακολουθούν μια αντίστροφη σχέση με το ipc.

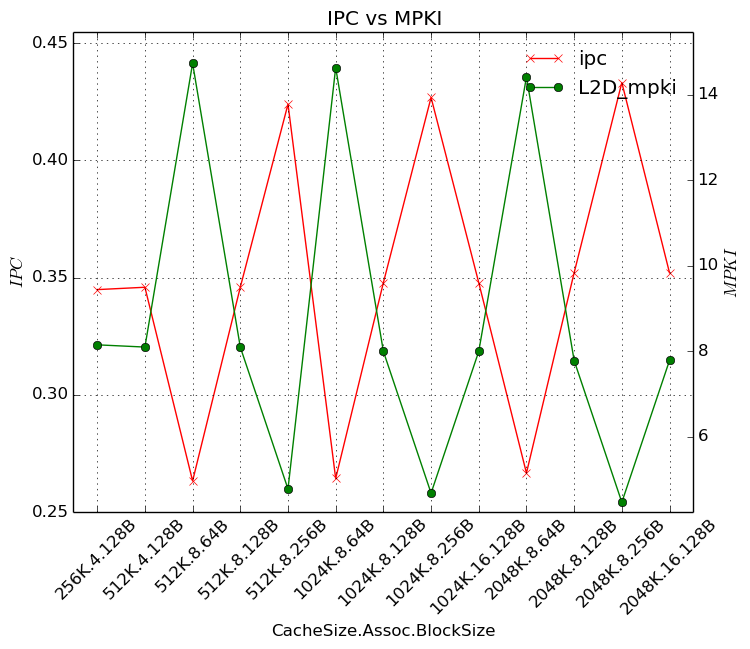
Στις παραπάνω δυο γραφικές παραστάσεις που αντιστοιχούν στα freqmine, rtview έχουμε μια αύξηση του ipc με το μέγεθος της μνήμης για το πρώτο, ενώ στο δεύτερο ουσιαστικά το Ipc παραμένει σταθερό με κάποιες αυξήσεις για μεγάλο μέγεθος μπλοκ.

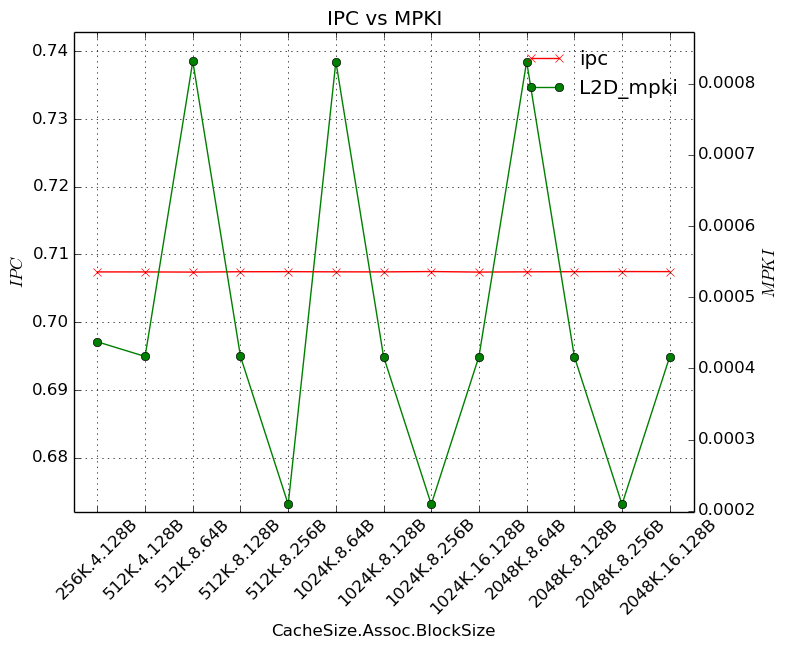


**7.1.2 L2 Cache**

Σ’ αυτό το σκέλος της άσκησης κάναμε την ίδια διαδικασία για την L2 , κρατήσαμε σταθερές όλες τις άλλες παραμέτρους και συγκρίναμε τα αποτελέσματα των μετροπρογραμμάτων για διαφορετικές συνθέσεις της L2. κΣτα παραπάνω 4 μετροπρογράμματα (blackscholes, bodytrack, canneal, freqmine) βλέπουμε πως το κύριο χαρακτηριστικό που επηρεάζει την απόδοση είναι το μέγεθος του μπλοκ, από τις υπόλοιπες παραμέτρους το μέγεθος της μνήμς φαίνεται να επηρεάζει ,στις δύο τελευταίες παραστάσεις ιδιαίτερα, την απόδοση.





Στις παραπάνω παραστάσεις για τα μετροπρογράμματα facesim ferret fluidanimate rtview streamcluster παρατηρούμε εντονότερες διακυμάνσεις . Με μεγάλη διαφορά το μέγεθος του μπλοκ έχει την μεγαλύτερη επίδραση. Στη συνέχεια η αύξηση του συνολικού μεγέθους της μνήμης λειτουργεί ελαφρώς βελτιωτικά (παρατηρούμε πάλι το μοτίβο να επαναλμβάνεται το ίδιο γράφημα λίγο «ψηλότερα», όποτε αυξάνει το μέγεθος της μνήμης). Τέλος το associativity φαίνεται να μην επηρεάζει καθόλου (ελέγχουμε τις περιπτώσεις 1024Κ.8.128Β και 1024.8.128Β ) 

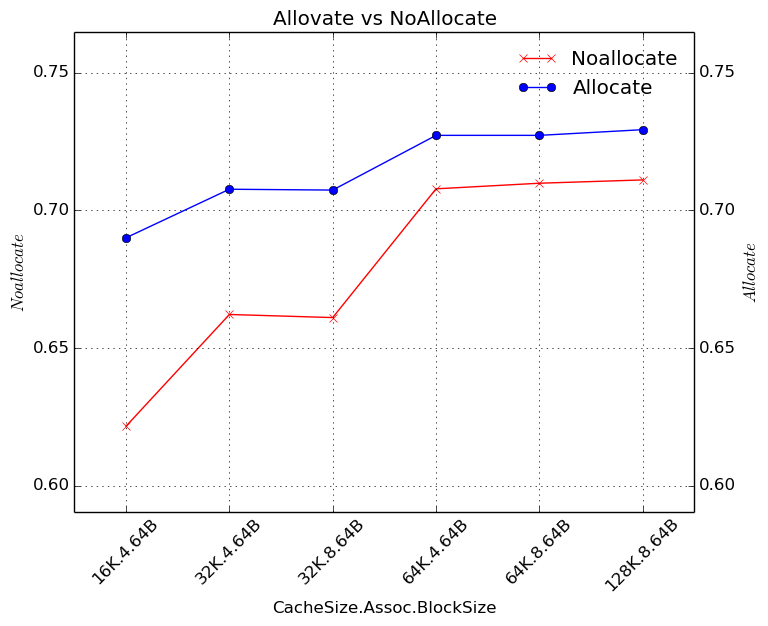
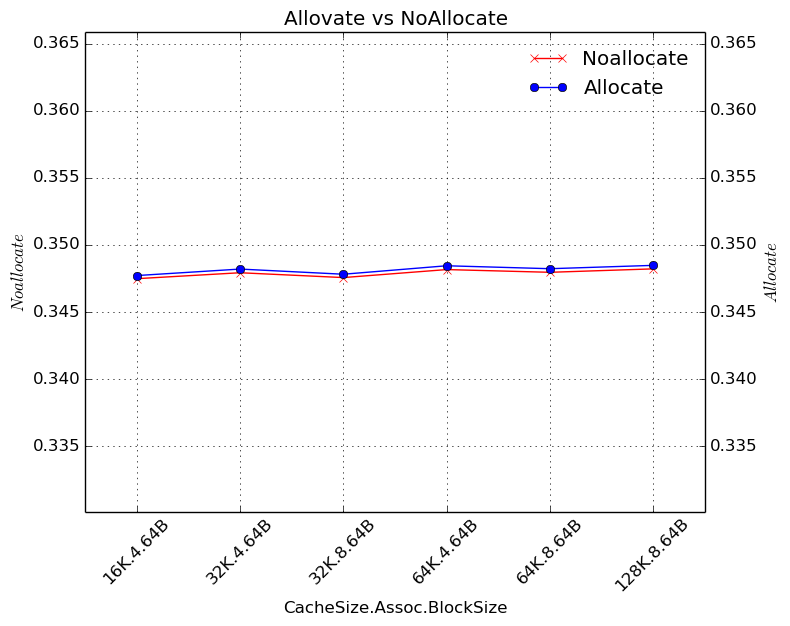
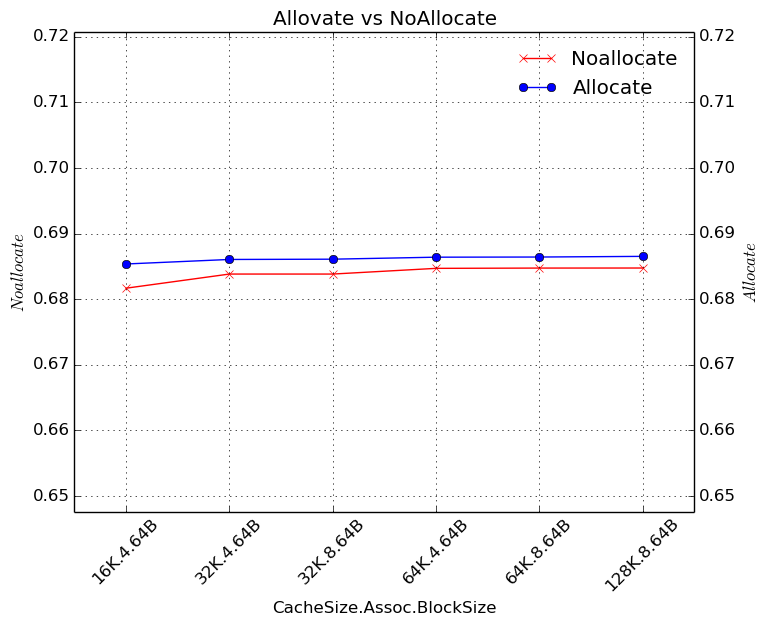
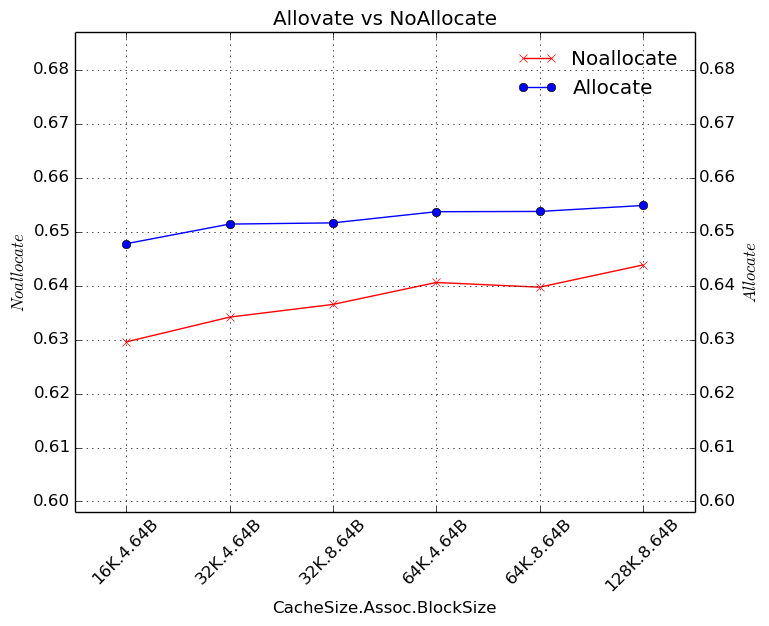
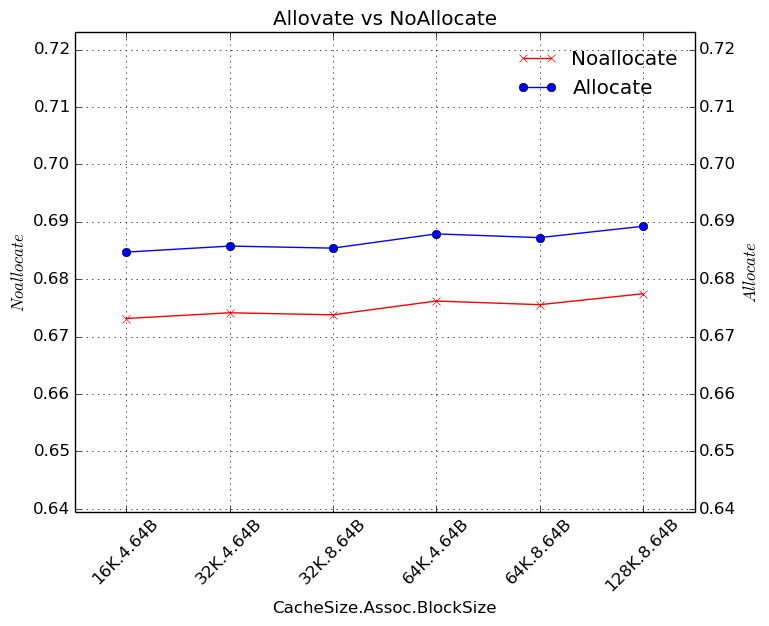
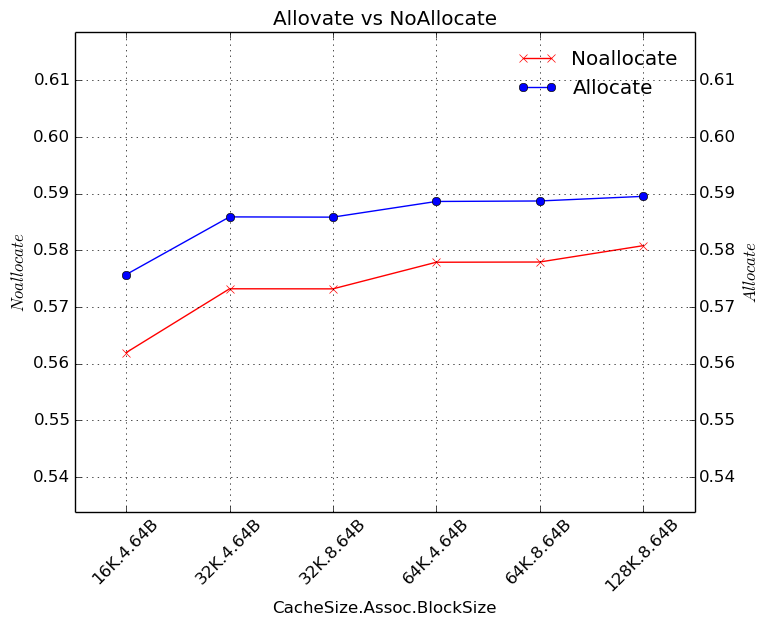
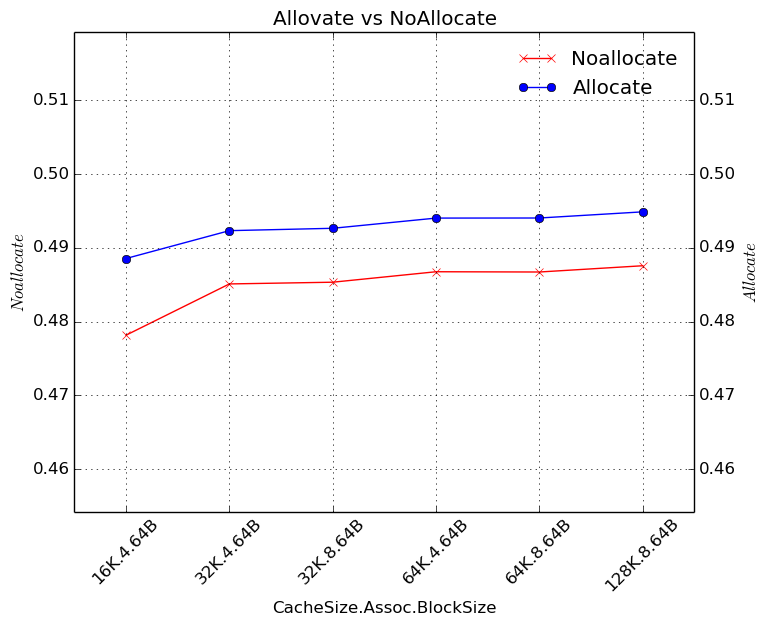
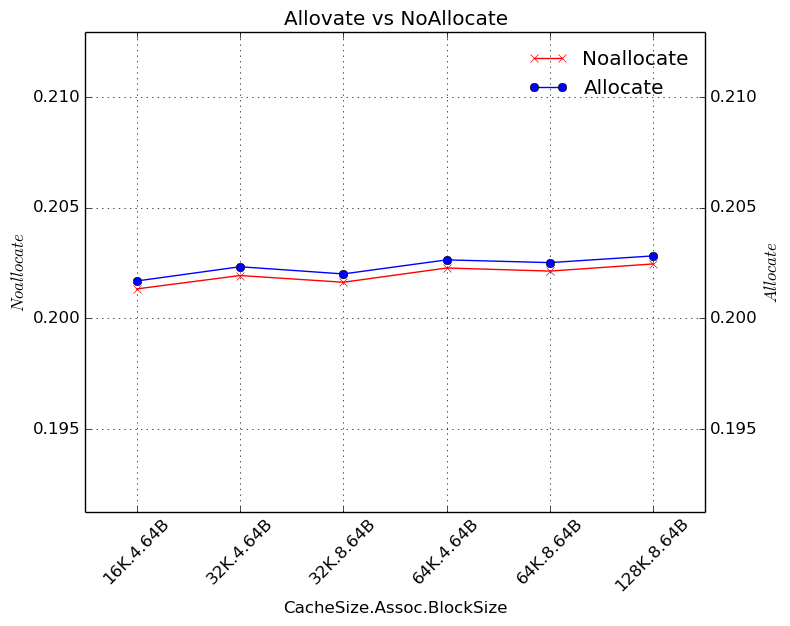
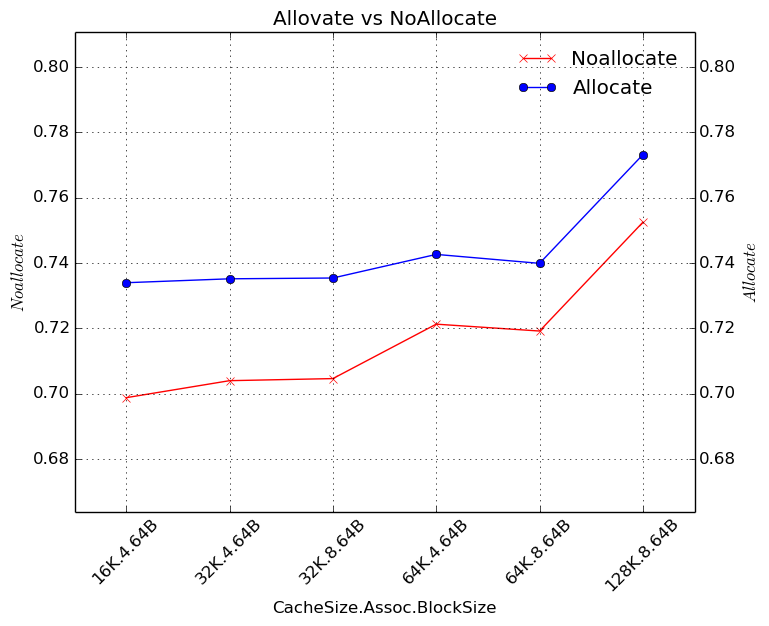
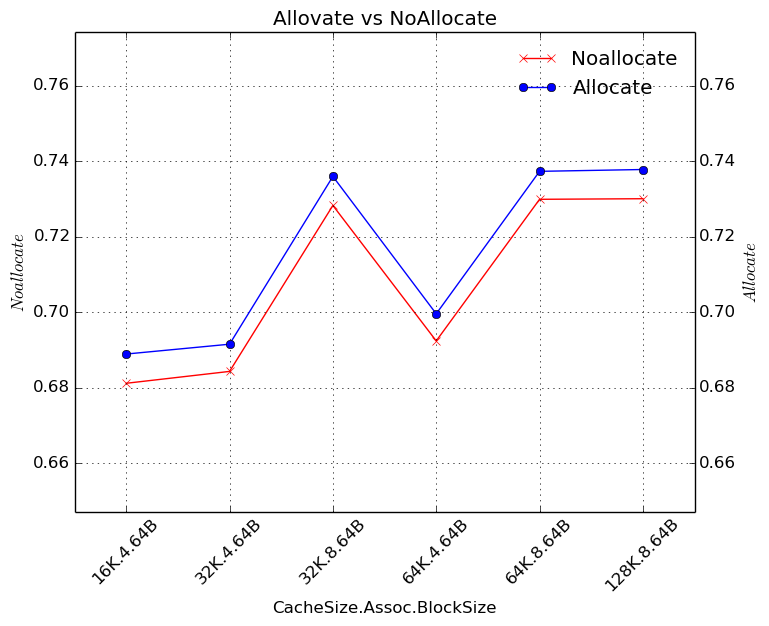
Τέλος στο μετροπρόγραμμα swaptions δε παρατηρούμε διαφοροποιήσεις ούτε στο IPC ούτε στο MPKI(οι αυξομειώσεις που εμφανίζονται στο διάγραμμα είναι φαινομενικές καθώς η διακύμανση στο mpki είναι της τάξεως του 5ου δεκαδικού ψηφίου )

Συμπερασματικά για την δευτέρου επιπέδου κρυφή μνήμη φαίνεται να παίζουν ρόλο πρώτα το μέγεθος του μπλοκ έπειτα το συνολικό μέγεθος και τέλος το associativity.

**7.1.3 Write Allocate**

Σ αυτό το σκέλος τροποποιούμε τα αρχεία cache.h και cslab\_cache.cpp ώστε η τιμή της σταθεράς STORE\_ALLOCATION να γίνει STORE\_NO\_ALLOCATΕ. Στη συνέχεια παίρνουμε μετρήσεις για μια σειρά διαφορετικών συνθέσεων της κρυφής μνήμης πρώτου επιπέδου.

Κατόπιν παρουσιάζονται οι μετρήσεις για την πολιτική no allocation σε σύγκριση με τις αντίστοιχες συνθέσεις κρυφής μνήμης για την πολιτική allocate που χρησιμοποιήθηκε στο πρώτο ερώτημα της εργασίας. Το συμπέρασμα που βγαίνει παρατηρώντας τις γραφικές είναι πως η πολιτική no allocate έχει αρνητική επίδραση στην απόδοση. Ένα ακόμη στοιχείο που βγαίνει από τις γραφικές είναι πως τα χαρακτηριστικά της κρυφής μνήμης επιπέδου 1 επηρεάζουν με τον ίδιο τρόπο την απόδοση και στην πολιτική no allocate. Εξαγάγουμε αυτό το συμπέρασμα από το γεγονός πως οι μορφές των γραφικών για τις δύο πολιτικές σ όλα τα μετροπρογράμματα είναι σχεδόν ταυτόσημες.



**7.1.4 Πολιτική Αντικατάστασης**

Σ αυτό το σημείο τροποποιούμε τον κώδικα του αρχείου cache.h έτσι ώστε να υλοποιήσουμε τις δύο νέες πολιτικές αντικατάστασεις που ζητά η εκφώνηση. Συγκεκριμένα για την πολιτική random τροποποιύμε τον κώδικα στα εξής σημεία. Πρώτο στην περίπτωση που έχουμε κάποιο hit δε μας ενδιαφέρει να κάνουμε κάποια αντικατάσταση έτσι απενεργοποιούμε τις μεταθέσεις που έκανε η πολιτική LRU στην συνάρτηση FIND. Στη συνάρτηση replace τροποιούμε των κώδικα έτσι ώστε στην περίπτωση που σε κάποιο set (η μνήμη μας χωρίζεται σε set ανάλογα με την τιμή του asoosiativity) δεν υπάρχει κενός χώρος να απομακρύνεται κάποιο τυχαία. Το παραπάνω το υλοποιήσαμε με την εισαγωγή των παρακάτω εντολών σε c++:

**thee = std::rand() % \_associativity;**

**\_tags.erase(\_tags.begin() + thee);**

Για την πολιτική LFU (least frequently used) χρειάστηκαν περισσότερες τροποποιήσεις. Καταρχάς δημιουργήσαμε ένα νέο διάνυσμα που περιέχει τις συχνότητες εμφάνισης κάθε στοιχείου της cache. Για τη μέθοδο find όποτε έχουμε κάποιο hit σε στοιχείο αυξάνουμε και την αντίστοιχη καταχώρηση για τη συχνότητα εμφάνισής του.

**int i = it - \_tags.begin();   
 \_frequency[i]++;**

Στη μέθοδο Replace αυτό που κάνουμε σε περίπτωση που το set μνήμης που χρειαζόμαστε είναι γεμάτο, είναι να αναζητούμε το στοιχείο με τη μικρότερη συχνότητα εμφάνισης και να το διαγράφουμε μαζί με τη συχνότητα εμφάνισης του, επίσης για κάθε νέα εισαγωγή στοιχείου αρχικοποιούμε και τη συχνότητα εμφάνισης του στο 0.

**std::vector<int>::iterator it = std::min\_element(\_frequency.begin(),\_frequency.end());**

**int i = it - \_frequency.begin();**

**vector<CACHE\_TAG>::iterator nth = \_tags.begin() + i;**

**ret = \*it;**

**\_tags.erase(nth);**

**\_frequency.erase(it); // delete element and it’s frequency**

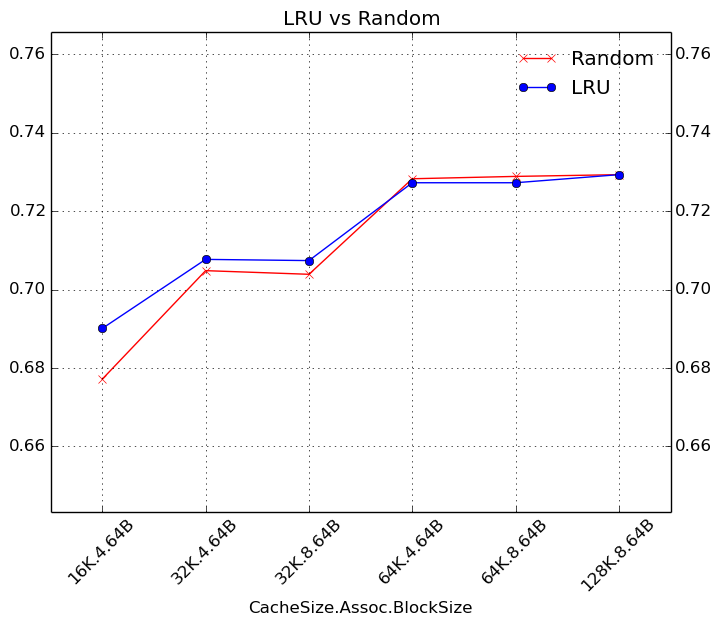
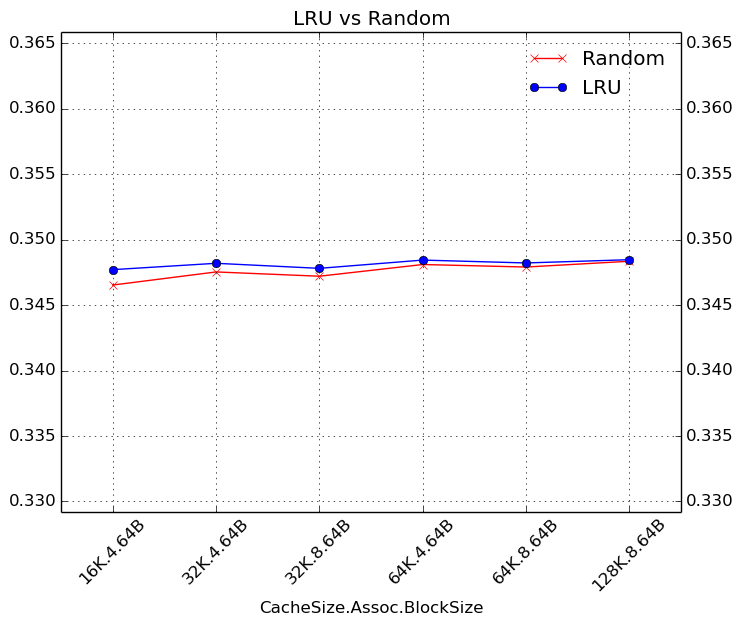
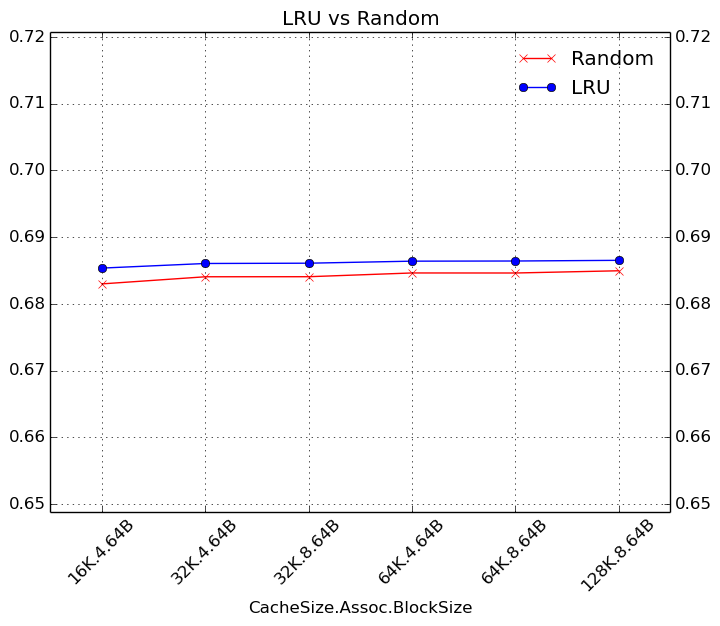
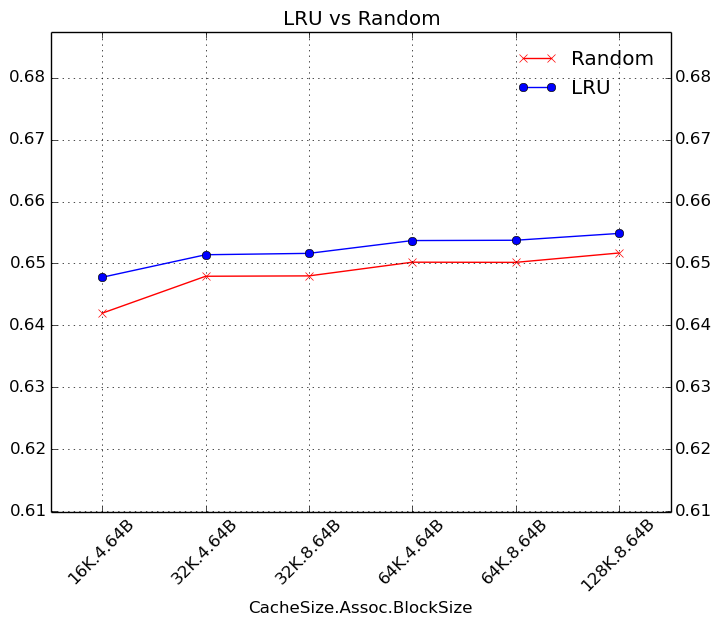
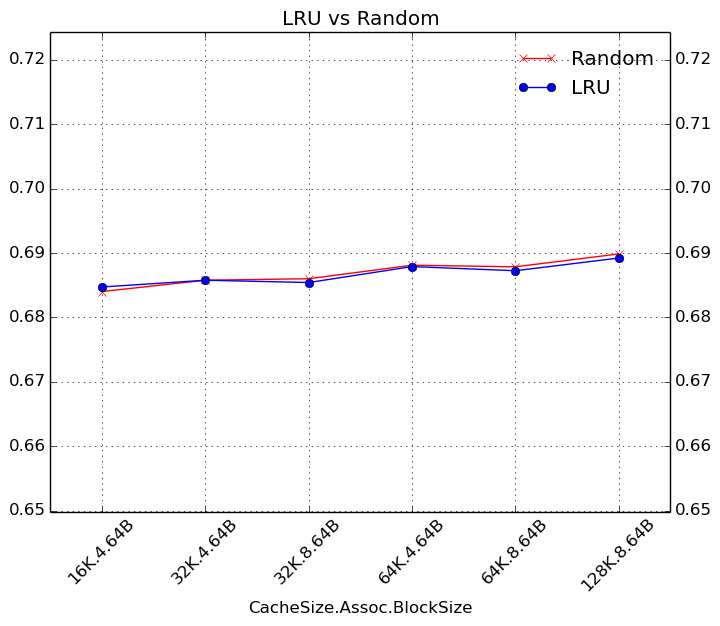
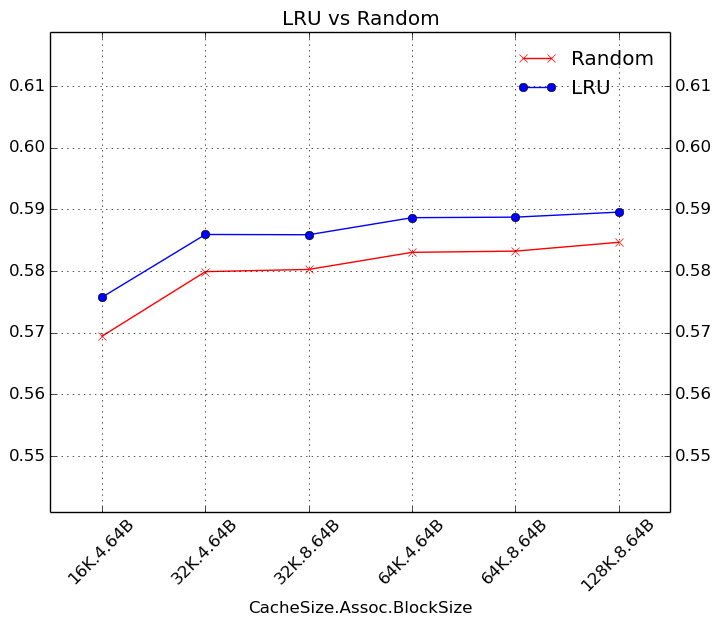
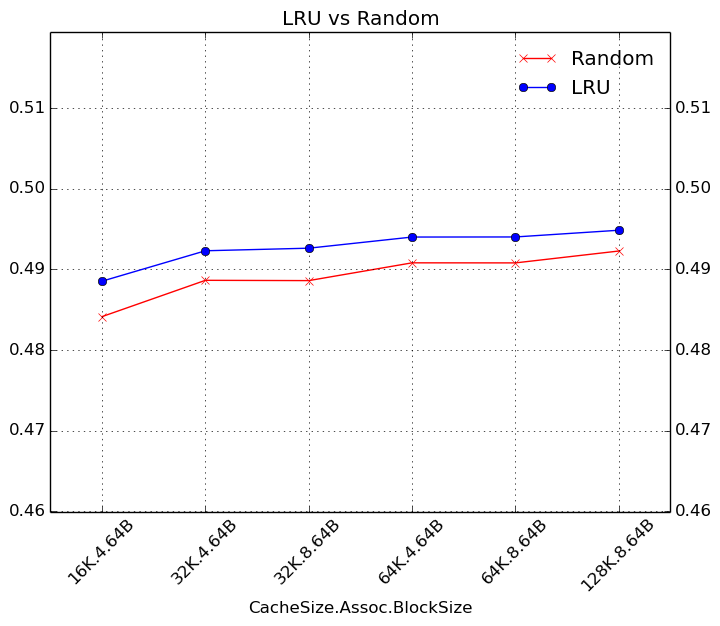
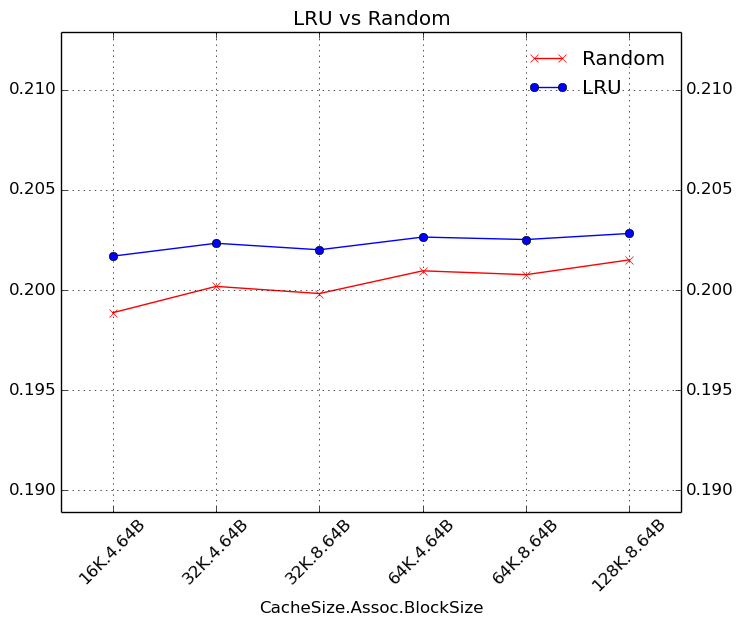
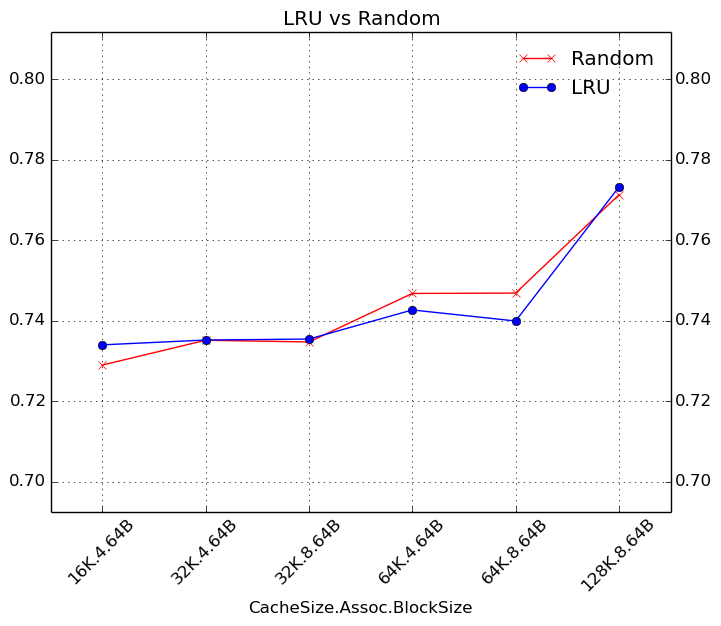
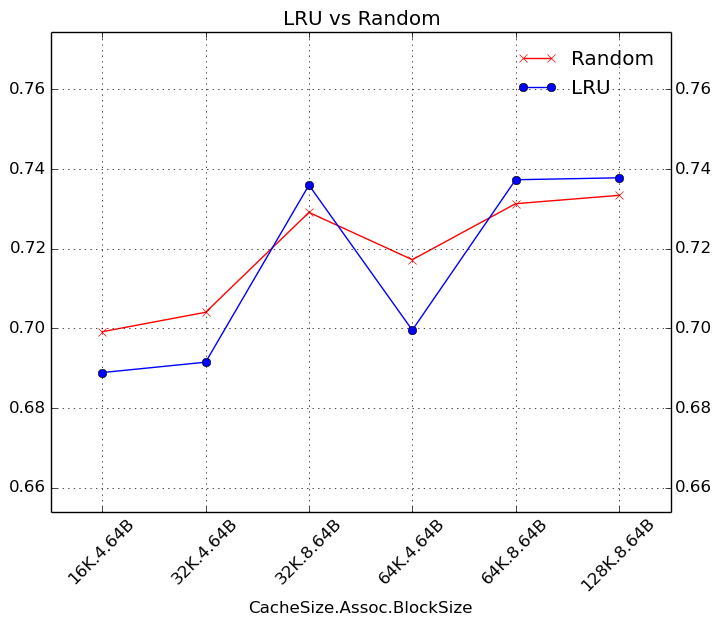
**\_tags.push\_back(tag);**

**\_frequency.push\_back(0); // initilitation to zero**

Τέλος στη συνάρτηση DeleteIfPresent σβήνουμε και την καταχώριση για τη συχνότητα εμφάνισης του στοιχείου.   
  
 **int i = it - \_tags.begin();  
 vector<int>::iterator nth = \_frequency.begin() + i;**

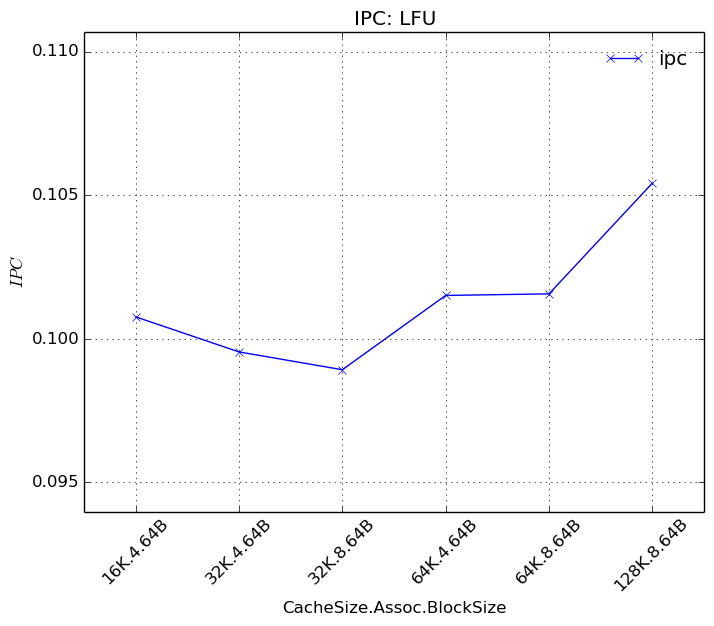
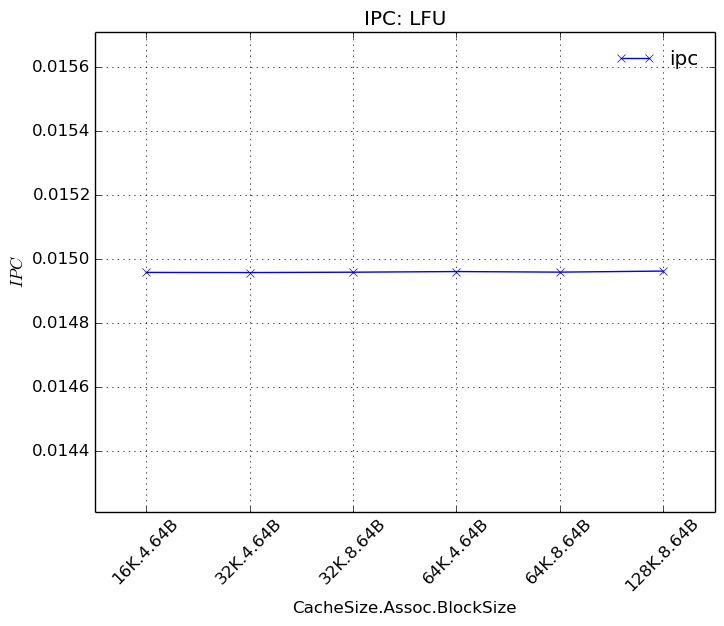
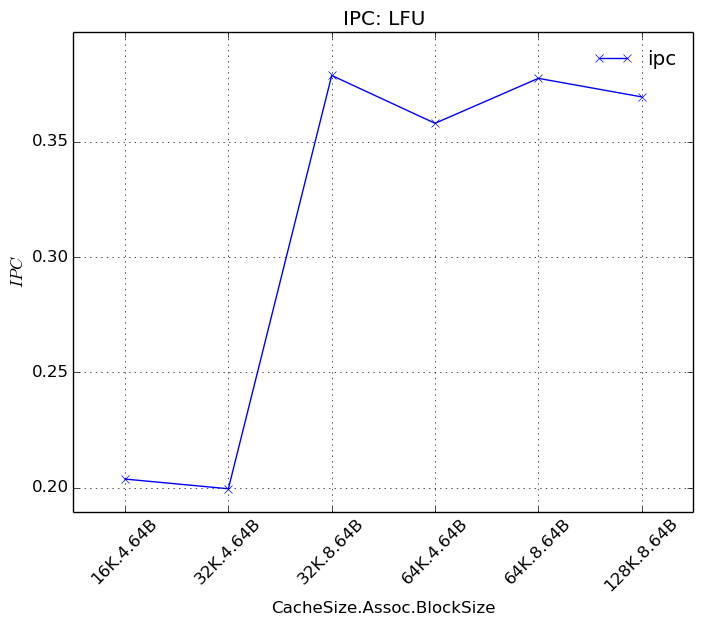
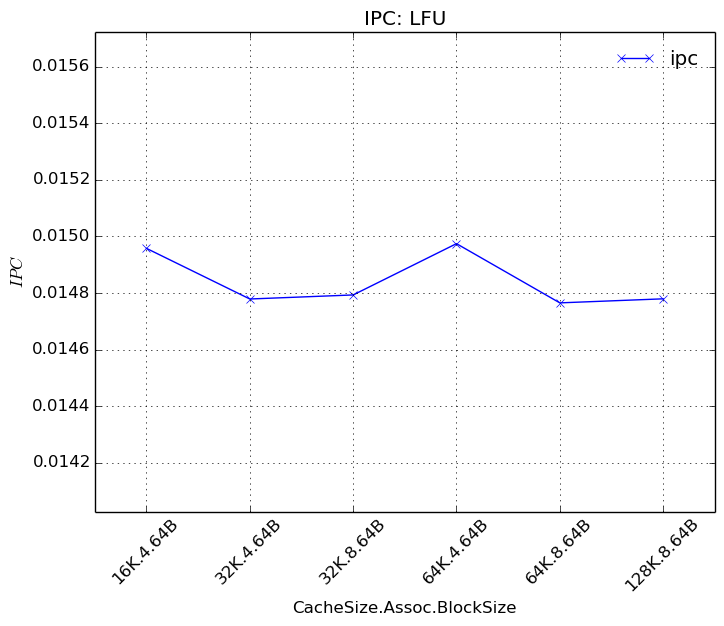
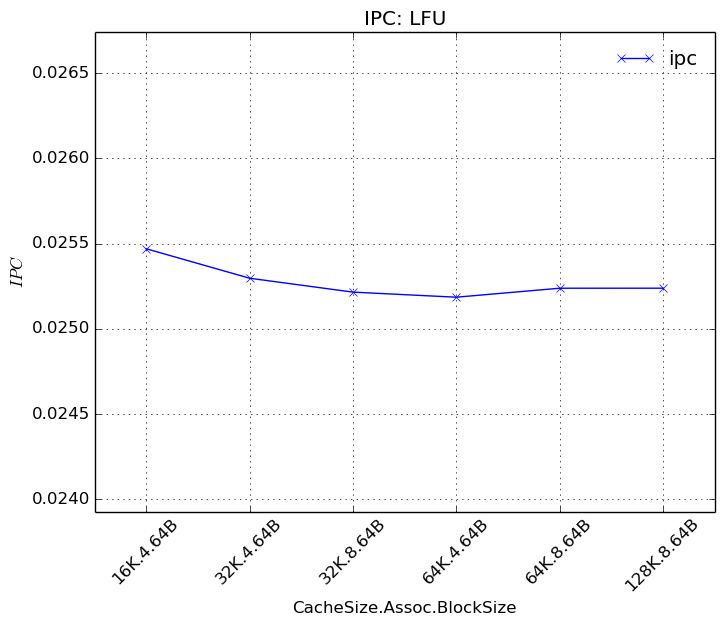
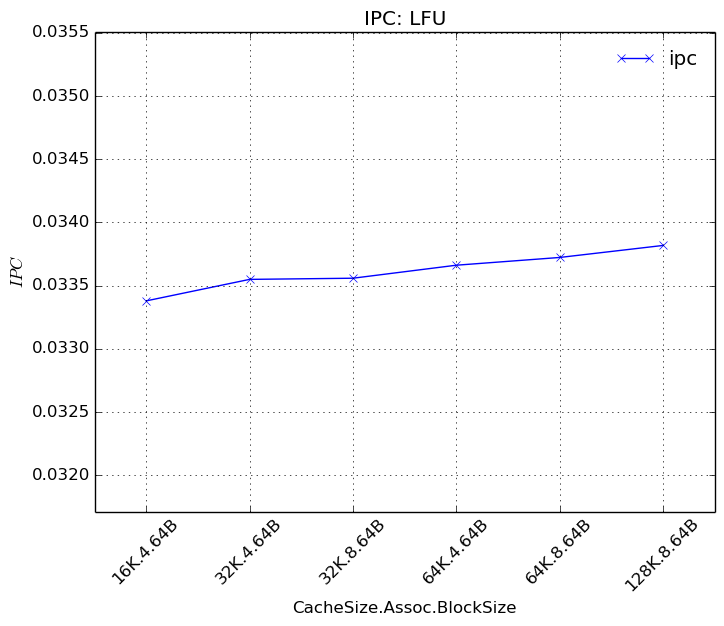
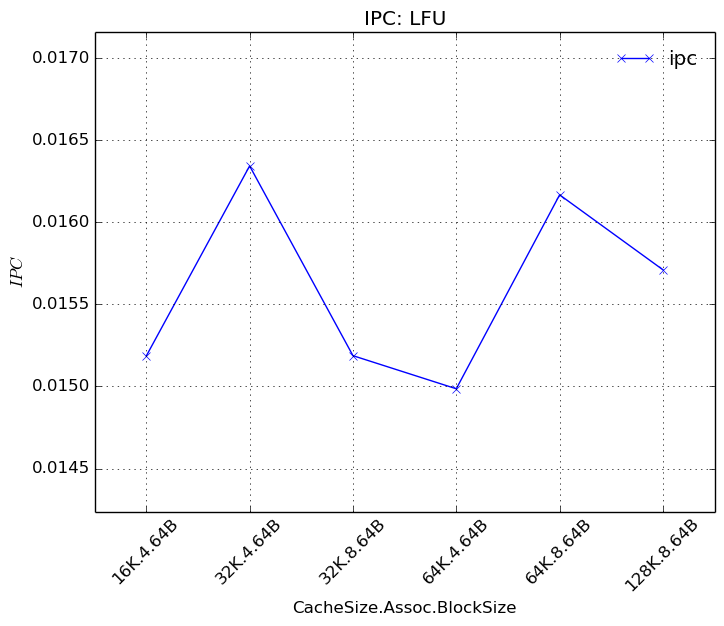
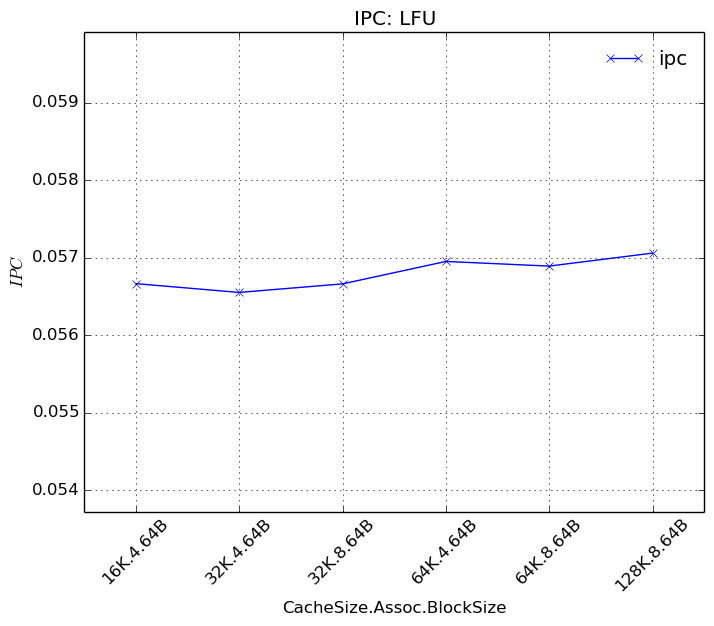
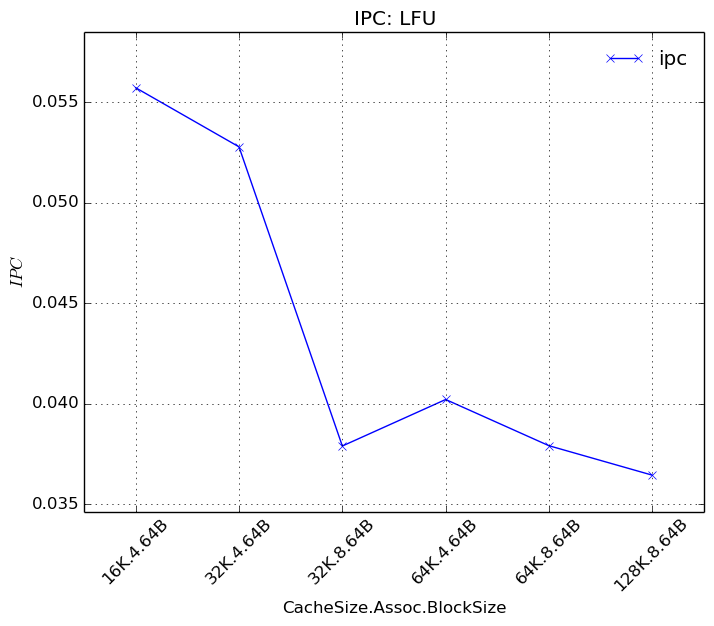
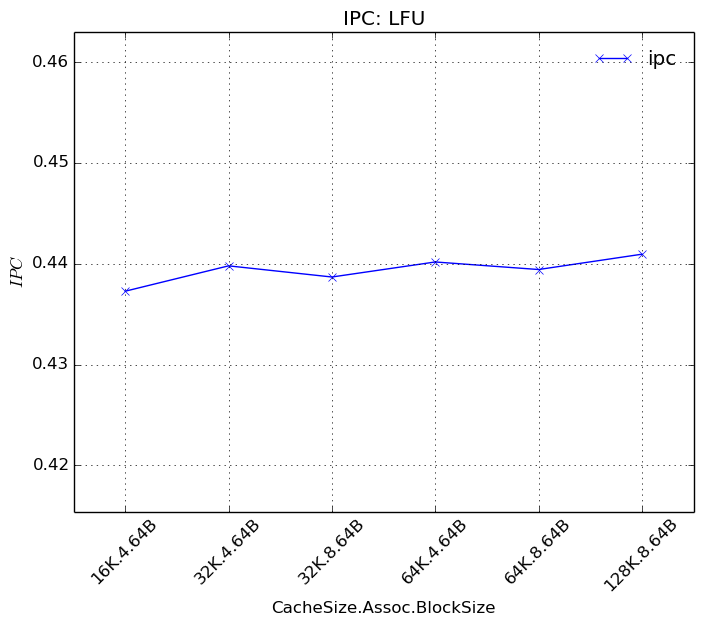
**\_tags.erase(it);  
 \_frequency.erase(nth);**

Στη συνέχεια ακολουθούν οι γραφικές παραστάσεις για τις τρεις πολιτικές αντικατάστασης για τα διάφορα μετροπρογράμματα. Οι γραφικές για τις πολιτικές Random και LRU παρουσιάζονται στο ίδιο πλέγμα . Η πολιτική LFU επιλέχτηκε να παρουσιαστεί ξεχωριστά καθώς οι τιμές τις διέφεραν σημαντικά με των άλλων με αποτέλεσμα να χάνεται η λεπρομέρεια των παραστάσεων.



Στα περισσότερα από τα μετροπρογράμματα βλέπουμε ότι υπάρχει ένα μικρό προβάδισμα της πολιτικής LRU αλλά υπάρχουν και πολλές εναλλαγές μεταξύ τους. Ακόμα παρατηρούμε πως η μορφή των γραφικών μεταβάλεται με τον ίδιο τρόπο για τις διαφορετικές οργανώσεις της κρυφής μνήμης πρώτου επιπέδου

Στη συνέχεια ακολουθούν οι παραστάσεις για την πολιτική LFU οι οποίες παρουσίασαν πολύ χαμηλές τιμές για το IPC. Μόνο για τα μετροπρογράμματα blackscholes και rtview το IPC έχει σχετικά μεγάλες τιμές.



**7.2 Μελέτη μεταβολής μετρικών απόδοσης στο χρόνο**  
  
Στο τελευταίο κομμάτι της άσκησης ζητήθηκε να μελετηθεί η δυναμική συμπεριφορά των εφαρμογών εξετάζοντας τον τρόπο που οι διάφορες μετρικές μεταβάλλονται στο χρόνο. Για να το πετύχουμε αυτό τροποποιήσαμε κατάλληλα το αρχείο cslab\_cache.cpp έτσι ώστε να αποθηκεύουμε το IPC σε αρχείο κάθε 10Μ εντολές. Τοποθετήσαμε τις αλλαγές αυτές στη μέθοδο count\_instruction().

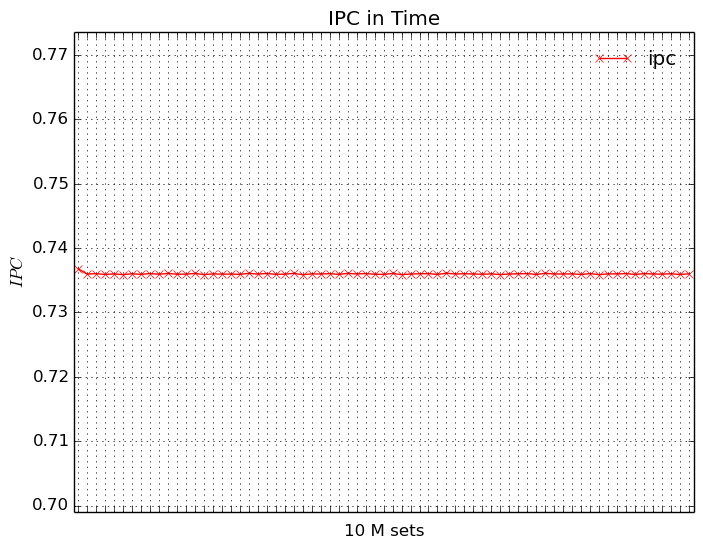
**if (total\_instructions == 10000000) {**

**outFile << "IPC: " << (double)total\_instructions / (double)total\_cycles << "\n";   
 total\_instructions=0;**

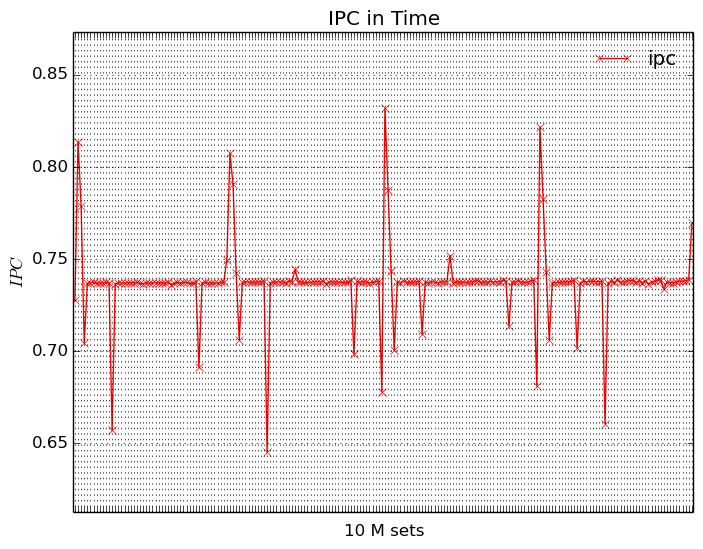
**total\_cycles=0;**

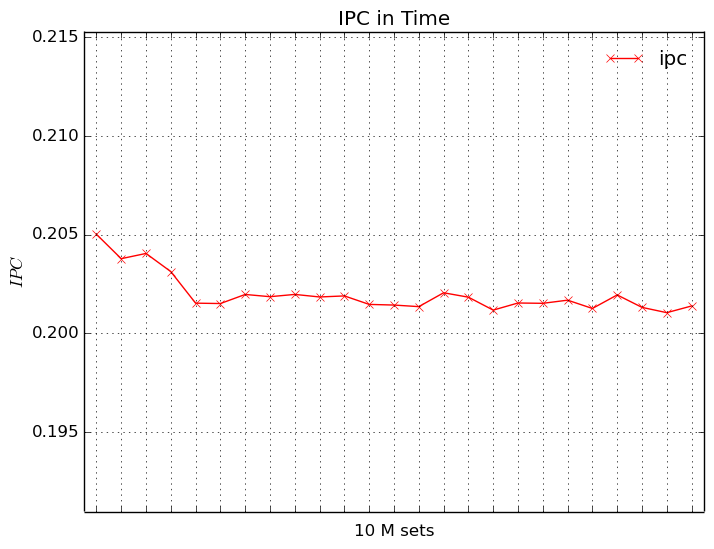
**}**

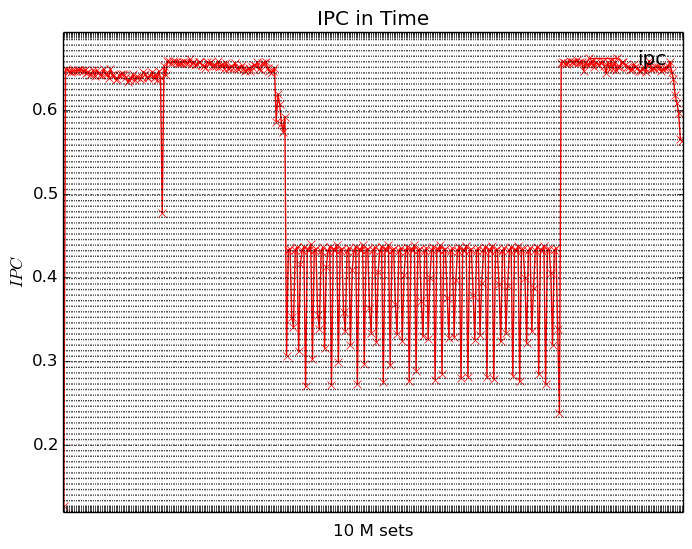
Στις γραφικές που ακολουθούν αναπαριστάται η τιμή του IPC για τα διάφορα set των 10Μ εντολών.

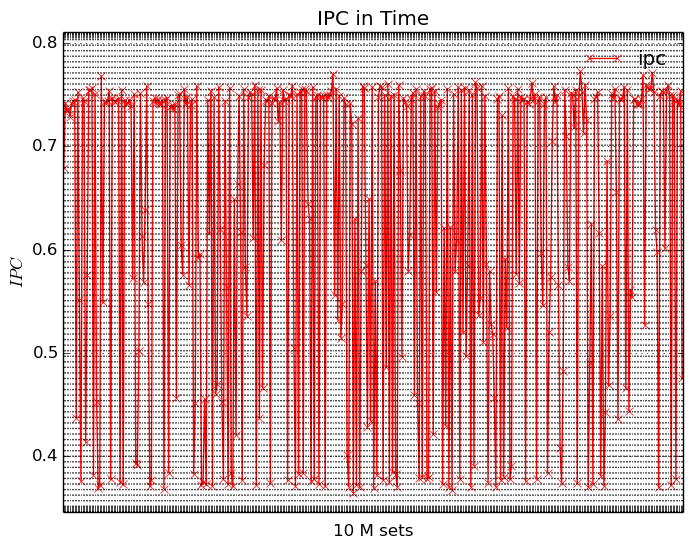


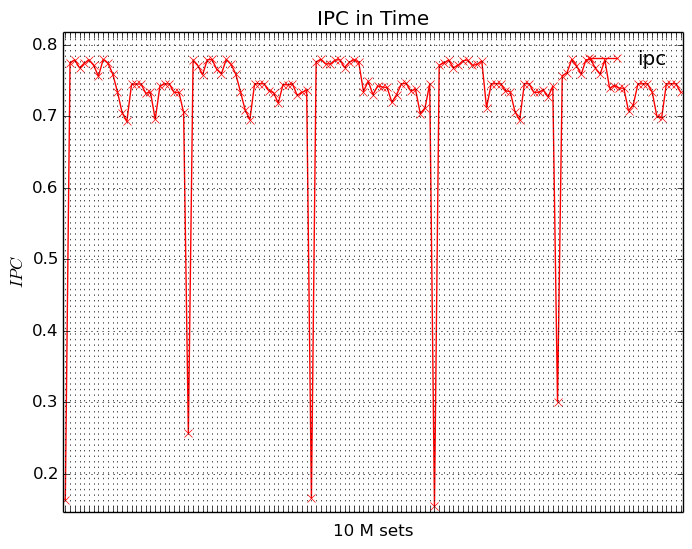
Για το μετροπρόγραμμα blackscholes η γραφική παράσταση του οποίου φαίνεται παραπάνω δε φαίνεται να επηρεάζεται από το χρόνο. Στη γενική περίπτωση όμως όπως φαίνεται και από τα υπόλοιπα μετροπρογράμματα οι μετρήσεις που έγινα ανά 10Μ εντολών δε μπορούν να θεωρηθούν ενδεικτικές. Καταρχάς πρόκειται για ένα μικρό δείγμα εντολών, κυριότερα όμως είναι γεγονός πως οι ανάγκες για μνήμη των προγραμμάτων που χρησιμοποιούμε για δείγματα είναι πολύ πιθανό να μεταβάλλονται σε διαφορετικά σημεία της εκτέλεσης τους κάτι που σίγουρα επιδρά στην τιμή του IPC.  
  
Έτσι λοιπόν μπορούμε να χωρίσουμε τις γραφικές αυτού του σκέλους σε 3ις κατηγορίες . Αυτές που αναφέρονται σε προγράμματα που κάνουν σταθερή περίπου χρήση μνήμης και οι οποίες παρουσιάζουν και τις λιγότερες μεταβολές. Τέτοια είναι και τα swaptions , streamcluster. Αυτές που κατά σημεία κάνουν μεγαλύτερη χρήση μνήμης με αποτέλεσμα συγκεκριμένα συνεχόμενα τμήματα 10μ εντολών να παρουσιάζουν χαμηλότερη τιμή IPC όπως το facesim και τέλος αυτές που δεν κάνουν κάποια κανονικοποιημένη χρήση μνήμης με αποτέλεσμα οι τιμές του IPC να παρουσιάζουν μεγάλες μεταβολές μεταξύ κοντινών set 10m εντολών. Σ αυτή την κατηγορία ανήκουν τα μετροπρογράμματα feret και rtview.

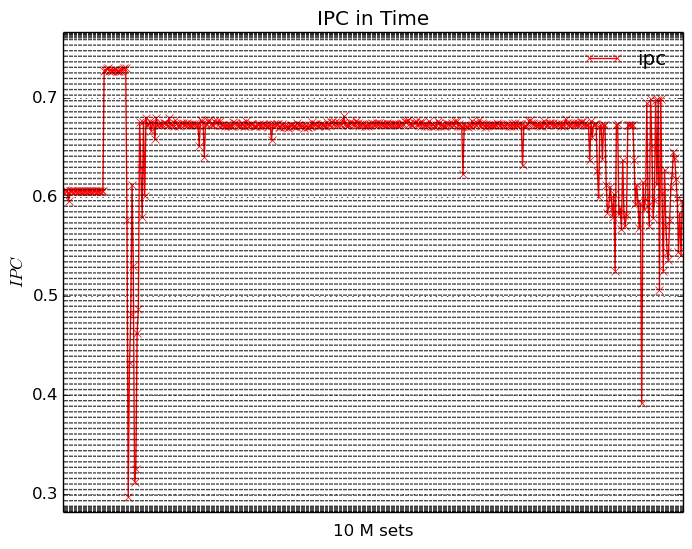
Γραφική bodytrack:

Γραφικήcanneal:

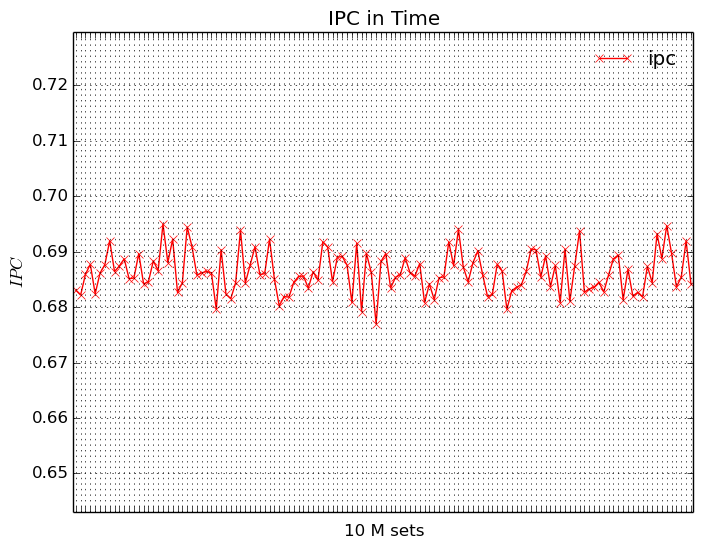
Γραφική Facesim:

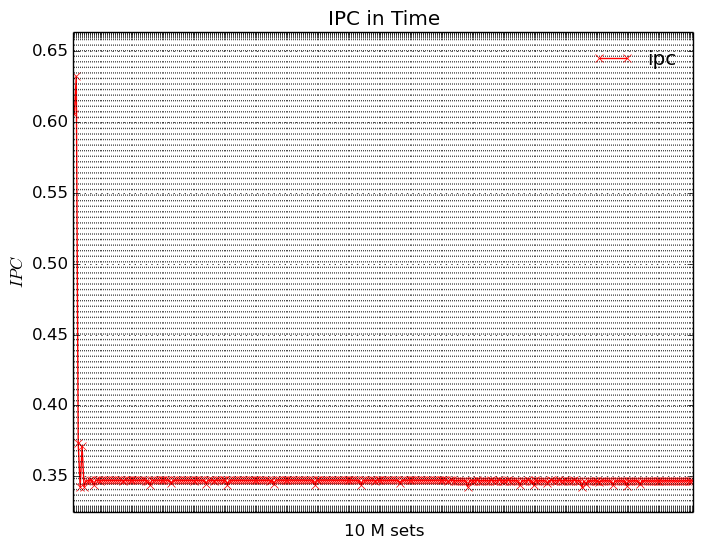
Γραφική ferret:

Γραφική fludanimate:

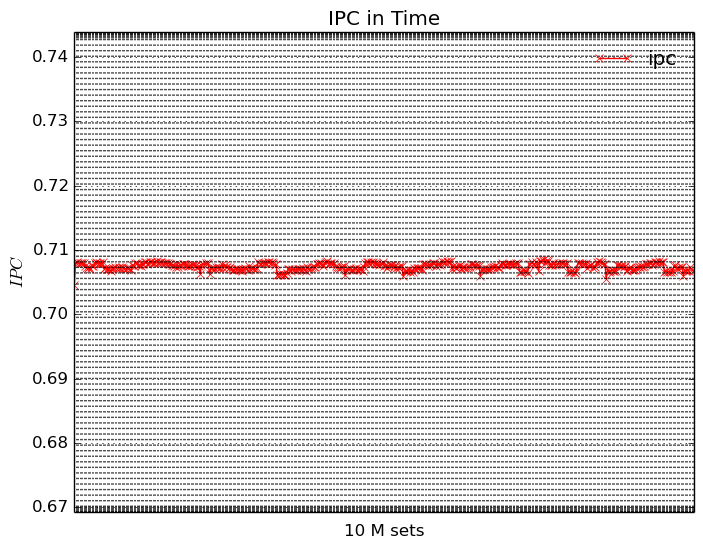
Γραφική freqmine:

Γραφική rtview



Γραφική streamcluster:

Γραφική swaptions:



Για το σχεδιασμό όλων των παραπάνω γραφικών έγιναν τροποποιήσεις στο αρχικό script που δόθηκε, ενδεικτικά παραθέτουμε τα script που χρησιμοποιήσαμε για το σχεδιασμό των παραστάσεων του 7.1.3 και του 7.2

**7.1.3:**

|  |
| --- |
| #!/usr/bin/env python  import sys  import numpy as np  ## We need matplotlib:  ## $ apt-get install python-matplotlib  import matplotlib  matplotlib.use('Agg')  import matplotlib.pyplot as plt  x\_Axis = []  ipc\_Axis = []  mpki\_Axis = []  for outFile in sys.argv[1:]:  fp = open(outFile)  line = fp.readline()  while line:  tokens = line.split()  if(outFile.startswith("no")):  if (line.startswith("IPC:")):  ipc = float(tokens[1])  elif (line.startswith(" L1-Data Cache")):  sizeLine = fp.readline()  l1\_size = sizeLine.split()[1]  bsizeLine = fp.readline()  l1\_bsize = bsizeLine.split()[2]  assocLine = fp.readline()  l1\_assoc = assocLine.split()[1]  else:  if (line.startswith("IPC:")):  mpki = float(tokens[1])  line = fp.readline()  fp.close()  if(outFile.startswith("no")):  l1ConfigStr = '{}K.{}.{}B'.format(l1\_size,l1\_assoc,l1\_bsize)  print l1ConfigStr  x\_Axis.append(l1ConfigStr)  ipc\_Axis.append(ipc)  else:  mpki\_Axis.append(mpki)  print x\_Axis  print ipc\_Axis  print mpki\_Axis  fig, ax1 = plt.subplots()  ax1.grid(True)  ax1.set\_xlabel("CacheSize.Assoc.BlockSize")  xAx = np.arange(len(x\_Axis))  ax1.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))  ax1.set\_xticklabels(x\_Axis, rotation=45)  ax1.set\_xlim(-0.5, len(x\_Axis) - 0.5)  ax1.set\_ylim(min(min(ipc\_Axis),min(mpki\_Axis)) - 0.05 \* min(ipc\_Axis), max(max(mpki\_Axis),max(ipc\_Axis)) + 0.05 \* max(ipc\_Axis))  ax1.set\_ylabel("$Noallocate$")  line1 = ax1.plot(ipc\_Axis, label="Noallocate", color="red",marker='x')  ax2 = ax1.twinx()  ax2.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))  ax2.set\_xticklabels(x\_Axis, rotation=45)  ax2.set\_xlim(-0.5, len(x\_Axis) - 0.5)  ax2.set\_ylim(min(min(ipc\_Axis),min(mpki\_Axis)) - 0.05 \* min(ipc\_Axis), max(max(mpki\_Axis),max(ipc\_Axis)) + 0.05 \* max(ipc\_Axis))  ax2.set\_ylabel("$Allocate$")  line2 = ax2.plot(mpki\_Axis, label="Allocate", color="blue",marker='o')  lns = line1 + line2  labs = [l.get\_label() for l in lns]  plt.title("Allovate vs NoAllocate")  lgd = plt.legend(lns, labs)  lgd.draw\_frame(False)  name=outFile  plt.savefig("L1\_noall\_"+name+".png",bbox\_inches="tight") |

**7.2**

|  |
| --- |
| #!/usr/bin/env python  import sys  import numpy as np  ## We need matplotlib:  ## $ apt-get install python-matplotlib  import matplotlib  matplotlib.use('Agg')  import matplotlib.pyplot as plt  x\_Axis = []  ipc\_Axis = []  i=0  for outFile in sys.argv[1:]:  fp = open(outFile)  line = fp.readline()  while line:  tokens = line.split()  if (line.startswith("IPC:")):  ipc = float(tokens[1])  ipc\_Axis.append(ipc)  i+=1  l1ConfigStr = ' '  x\_Axis.append(l1ConfigStr)    fp.readline()  fp.readline()  fp.readline()  fp.readline()  fp.readline()  line = fp.readline()  fp.close()  ##print l1ConfigStr  print x\_Axis  print ipc\_Axis  fig, ax1 = plt.subplots()  ax1.grid(True)  ax1.set\_xlabel("10 M sets")  xAx = np.arange(len(x\_Axis))  ax1.xaxis.set\_ticks(np.arange(0, len(x\_Axis), 1))  ax1.set\_xticklabels(x\_Axis, rotation=45)  ax1.set\_xlim(-0.5, len(x\_Axis) - 0.5)  ax1.set\_ylim(min(ipc\_Axis) - 0.05 \* min(ipc\_Axis), max(ipc\_Axis) + 0.05 \* max(ipc\_Axis))  ax1.set\_ylabel("$IPC$")  line1 = ax1.plot(ipc\_Axis, label="ipc", color="red",marker='x')  lns = line1  labs = [l.get\_label() for l in lns]  plt.title("IPC in Time")  lgd = plt.legend(lns, labs)  lgd.draw\_frame(False)  name=outFile  plt.savefig("L1\_noall\_"+name+".png",bbox\_inches="tight") |