ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕΛΕΤΗ ΤΩΝ NAS PARALLEL BENCHMARKS ΣΤΟ TORQUE ΚΑΙ ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ CO-SCHEDULING

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Ν. Πέππας

Επιβλέπων: Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π

ΑΘΗΝΑ , ΟΚΤΩΒΡΙΟΣ 2020

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕΛΕΤΗ ΤΩΝ NAS PARALLEL BENCHMARKS ΣΤΟ TORQUE ΚΑΙ ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ CO-SCHEDULING

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παναγιώτης Ν. Πέππας

Επιβλέπων: Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελη εξεταστική επιτροπή την 7η Οκτωβρίου 2020

(*Υπογραφή*)	(*Υπογραφή*)	(*Υπογραφή*)
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Επ.Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

ΑΘΗΝΑ , ΟΚΤΩΒΡΙΟΣ 2020

(*Υπογραφή*)

……

Παναγιώτης Ν. Πέππας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Τα HPC (High Performance Computing), δηλαδή οι υπερυπολογιστές και τα υπολογιστικά συστήματα υψηλής επίδοσης (HPC clusters) είναι ευρέως διαδεδομένα τα τελευταία χρόνια, καθώς προσφέρουν σημαντικές δυνατότητες για προβλήματα μεγάλης κλίμακας. Κάθε υπολογιστικό σύστημα υψηλής επίδοσης, διαθέτει έναν χρονοδρομολογητή, δηλαδή ένα σύστημα διαχείρισης πόρων.

Το TORQUE (Terascale Open-source Resource and QUEue Manager), είναι ένας τέτοιος διαχειριστής κατανεμημένων πόρων, που παρέχει έλεγχο σε πακέτα εργασιών και σε κατανεμημένους κόμβους υπολογισμού.

Τα NPB (Nas Parallel Benchmarks) είναι ένα μικρό σύνολο προγραμμάτων που έχουν σχεδιαστεί για να βοηθήσουν στην αξιολόγηση της απόδοσης των παράλληλων υπερυπολογιστών.

Σκοπός της παρούσας διπλωματικής, είναι η εφαρμογή και η μελέτη των NPB στο cluster της σχολής και βάσει των αποτελεσμάτων της παραπάνω μελέτης η ανάπτυξη ενός σύντομου αλγορίθμου χρονοδρομολόγησης που εκμεταλλεύεται τις πιθανές συνεργασίες των διαφορετικών NPB. Τέλος, εκτελείται σενάριο υποβολής μιας ουράς εργασιών στον default αλγόριθμο του Torque και στο δικό μου αλγόριθμο, και σύγκριση των αποτελεσμάτων.

Λέξεις κλειδιά

Torque, mpirun, χρονοδρομολόγηση, πολιτικές χρονοδρομολήγησης, Nas Parallel Benchmarks, qsub, συστήματα μεγάλης κλίμακας

Abstract

HPC (High Performance Computing), that is supercomputers and high-performance computing systems (HPC clusters) are widespread in recent years, as they offer significant potential for large-scale problems. Each high-performance computing system has a scheduler, that is, a resource management system.

TORQUE (Terascale Open-source Resource and QUEue Manager) is one such distributed resource manager that provides control over batch jobs and distributed compute nodes.

The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers.

The purpose of this thesis is to apply and analyse NPBs in the ntua cluster and develop a simple scheduling algorithm based on the above study results that exploits the potential synergies of different NPB. Finally, a scenario of submitting a queue of works to the default algorithm of Torque and to my own algorithm is performed, and a comparative study of the results is conducted.

Keywords

Torque, mpirun, scheduling, co-scheduling policies
Nas Parallel Benchmarks, qsub, HPC

Ευχαριστίες

Ευχαριστώ τον καθηγητή μου κ. Γεώργιο Γκούμα, για την εποπτεία κατά την εκπόνηση της εργασίας μου και τις γνώσεις που μου προσέφερε με τη διδασκαλία του στο μάθημα των παράλληλων συστημάτων.

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον Υποψήφιο Διδάκτορα Νικόλαο Τριανταφύλλη, για την σημαντική του βοήθεια καθόλη τη διάρκεια της προσπάθειάς μου και για την συνεχή του καθοδήγηση και επίβλεψη.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου και τον αδελφό μου, που ήταν μαζί μου όλα αυτά τα χρόνια και με στήριζαν συνεχώς σε πολλούς τομείς.

Περιεχόμενα

Κατάλογος Σχημάτων

Κώδικες

Γραφικές Παραστάσεις

1.Εισαγωγή

1.1 Αντικείμενο της Διπλωματικής

Τα τελευταία χρόνια, η ανάπτυξη της τεχνολογίας είναι ραγδαία. Αυτό απεικονίζεται άμεσα στην αλματώδη ανάπτυξη των υπερυπολογιστών και των υπολογιστικών συστημάτων υψηλής επίδοσης. Τη σημερινή εποχή, οι κορυφαίοι υπερυπολογιστές έχουν καταφέρει να πετυχαίνουν 500 Tflops [1] ενώ πριν το 1960 μπορούσαν να κάνουν μόλις μερικές χιλιάδες πράξεις σε ένα δευτερόλεπτο. Αυτό συμβαίνει καθώς κάθε δεκαετία η ταχύτητα των υπερυπολογιστών είναι 200 φορές μεγαλύτερη από την προηγούμενη.

Η ανάγκη για αυτή την ραγδαία ανάπτυξη των υπερυπολογιστών, έρχεται σε άμεση συσχέτιση με την τεράστια συνεισφορά τους σε πολλούς τομείς της ζωής όπως: υπολογιστική δυναμική ρευστού, δομές, πρόγνωση καιρού και κλιματική αλλαγή, ιατρικές επιστήμες, χρηματοοικονομικές πράξεις, εικονική πραγματικότητα και πολλοί άλλοι.

Η δύναμη αυτών των υπολογιστών όμως έχει και ως συνέπεια την μεγάλη κατανάλωση ενέργειας για τη λειτουργία τους. Για το λόγο αυτό, η χρήση τους και η δυνατότητα πρόσβασης ενός ερευνητή σε αυτά περνάει πολλά στάδια, πριν καταφέρει ο ενδιαφερόμενος να τρέξει τις εφαρμογές του. Επιτακτική είναι η ανάγκη της σωστής και ώριμης χρήσης τους. Σημαντικός παράγοντας για να επιτευχθεί αυτό είναι ο σωστός διαμοιρασμός του χρόνου στους ενδιαφερόμενους και η ορθή και αποτελεσματική χρήση του από τους ίδιους.

Σκοπός αυτής της διπλωματικής, είναι η μελέτη των Nas Parallel Benchmarks και η προσπάθεια βελτίωσης του αυτόματου τρόπου υποβολής εργασιών του Torque. Αυτό το πετυχαίνουμε τρέχοντας τα προβλήματα αυτά μόνα τους και έπειτα σε ζευγάρια (πολιτική stripe) ώστε να διαπιστωθεί αν κάποια προβλήματα αξίζει από χρονικής άποψης να συνδυαστούν με κάποια άλλα. Έπειτα αναπτύσσεται ένας αλγόριθμος που διαβάζοντας από μια ουρά εργασιών μια λίστα με τέτοια προβλήματα, δεν τα υποβάλει να τρέξουν με τον αυτόματο τρόπο αλλά με έναν καλύτερο συνδυασμό μεταξύ τους βάσει της παραπάνω μελέτης.

1.2 Δομή της Εργασίας

Η εργασία αυτή έχει οργανωθεί σε επτά κεφάλαια και σε ένα παράρτημα. Το πρώτο είναι μια σύντομη εισαγωγή για το σύνολο της εργασίας.

Στο δεύτερο, γίνεται αναφορά στα HPC, καθώς η εργασία εντάσσεται στον τομέα των παράλληλων συστημάτων μεγάλης κλίμακας.

Στο τρίτο, αναφέρεται ο χρονοδρομολογητής Torque, που είναι ο διαχειριστής πόρων που χρησιμοποιεί το σύστημα της σχολής.

Στο τέταρτο, αναλύονται τα Nas Parallel Benchmarks, τα οποία εξετάζονται στη μεγαλύτερη έκταση της παρούσας εργασίας.

Στο πέμπτο, παρουσιάζεται η μελέτη των NPB τόσο σε compact όσο και σε stripe μορφή.

Στο έκτο, γίνεται μια ανάλυση αλγορίθμου για διαφορετική σειρά υποβολής εργασιών από την αυτόματη του χρονοδρομολογητή Torque καθώς επίσης παρουσιάζονται και τα αποτελέσματα του.

Στο έβδομο, αναφέρονται τα συμπεράσματα τις εργασίας και πιθανές κατευθύνσεις για περαιτέρω μελέτη του θέματος.

Στο παράρτημα σημειώνεται το σύστημα της σχολής πάνω στο οποίο έγινε η διπλωματική.

2.HPC

Πριν κάποια χρόνια, τα HPC clusters τα χρησιμοποιούσαν μόνο λίγοι επιστήμονες και μηχανικοί που είχαν ανάγκη να κάνουν πράξεις με αριθμούς όσο πιο γρήγορα γινόταν. Στις μέρες μας όμως, οποιοσδήποτε ερευνητής με ένα σοβαρό έργο που έχει ανάγκη να χρησιμοποιήσει μεγάλη υπολογιστική δύναμη για να εκτελέσει τους υπολογισμούς του, έχει τη δυνατότητα να αποκτήσει πρόσβαση σε κάποιον υπερυπολογιστή. Τέτοιοι υπολογισμοί είναι εφικτοί μόνο με τη χρήση υπολογιστών με πολλούς επεξεργαστές σε συνδυασμό με μεθόδους παράλληλης επεξεργασίας. Οι μέθοδοι αυτές των υπερυπολογιστών, στοχεύουν στην ταυτόχρονη εκμετάλλευση πολλών επεξεργαστών για την αντιμετώπιση μιας μεγάλης υπολογιστικής διεργασίας (task) χωρίζοντάς την σε μικρότερες ανεξάρτητες υποδιεργασίες (subtasks). Πλέον οι ίδιοι οι υπερυπολογιστές, κατασκευάζονται από εμπορικούς επεξεργαστές όπως αυτούς που ο καθένας μπορεί να αποκτήσει στο σπίτι του, καθιστώντας τους έτσι περισσότερο προσιτούς στους ερευνητές.

2.1 Οι μορφές του HPC

Στις μέρες μας υπάρχουν 4 μορφές ενός υπερυπολογιστή [2] :

- **The commodity HPC cluster:** Τα τελευταία χρόνια, τα HPC cluster έχουν εισχωρήσει δυναμικά στην περιοχή των υπερυπολογιστών. Κατασκευάζονται από τυπικούς διακομιστές και χρησιμοποιούν γρήγορες διασυνδέσεις, με αποτέλεσμα να μπορούν να παρέχουν κορυφαίες, οικονομικά αποδοτικές επιδόσεις. Ένα τυπικό cluster μπορεί να χρησιμοποιεί χιλιάδες ή και δεκάδες χιλιάδες servers που συνεργάζονται όλοι για ένα συγκεκριμένο πρόβλημα. Αυτή είναι και η αρχή του “διαίρει και βασίλευε” που χρησιμοποιούν οι υπερυπολογιστές. Λόγω της καλής του απόδοσης και του χαμηλού του κόστους, το commotidy cluster, καθίσταται η πιο διαδεδομένη μορφή HPC.
- **Dedicated supercomputer:** Αρχικά οι dedicated supercomputers ήταν ο μόνος τρόπος για να έχεις υψηλές επιδόσεις και πολλούς υπολογιστικούς κύκλους σε προβλήματα. Οι κλασικοί υπερυπολογιστές πρωταγωνιστούν και σήμερα και συνηθίζουν να χρησιμοποιούν για την κατασκευή τους όχι απαραίτητα εμπορικούς επεξεργαστές και εξαρτήματα. Αναλόγως τις υψηλές

απαιτήσεις κάποιου, αν και είναι ακριβή η χρήση τους, οι κλασικοί υπερυπολογιστές μπορεί να είναι η καλύτερη λύση.

- **HPC cloud computing:** Αυτή η μέδοθος χρησιμοποιείται τα τελευταία χρόνια όπου το internet έχει καταφέρει να έχει υψηλές ταχύτητες, αφού μέσω αυτού ο χρήστης αποκτά πρόσβαση σε απομακρυσμένα HPC. Ο χρήστης έχει τον έλεγχο ενός τέτοιου είδους hpc από απόσταση. Ένα τέτοιο hpc cloud παρέχει δυναμικούς και κλιμακώσιμους πόρους, αλλά και εικονοποίηση, στον τελικό χρήστη. Παρόλο που είναι οικονομικά συμφέρουσα αυτή η λύση, τοποθετούνται κάποια επίπεδα μεταξύ του τελικού χρήστη και του hardware που μπορεί να ελαττώνουν ελαφρώς την επίδοση του συστήματος.
- **Grid computing:** Το grid είναι παρόμοιο με το cloud computing αλλά απαιτεί περισσότερο έλεγχο από τον τελικό χρήστη. Χρησιμοποιείται κυρίως σε ακαδημαϊκές εργασίες όπου χρησιμοποιούνται τοπικά hpc clusters.

2.2 Η ταξινόμηση του Flynn

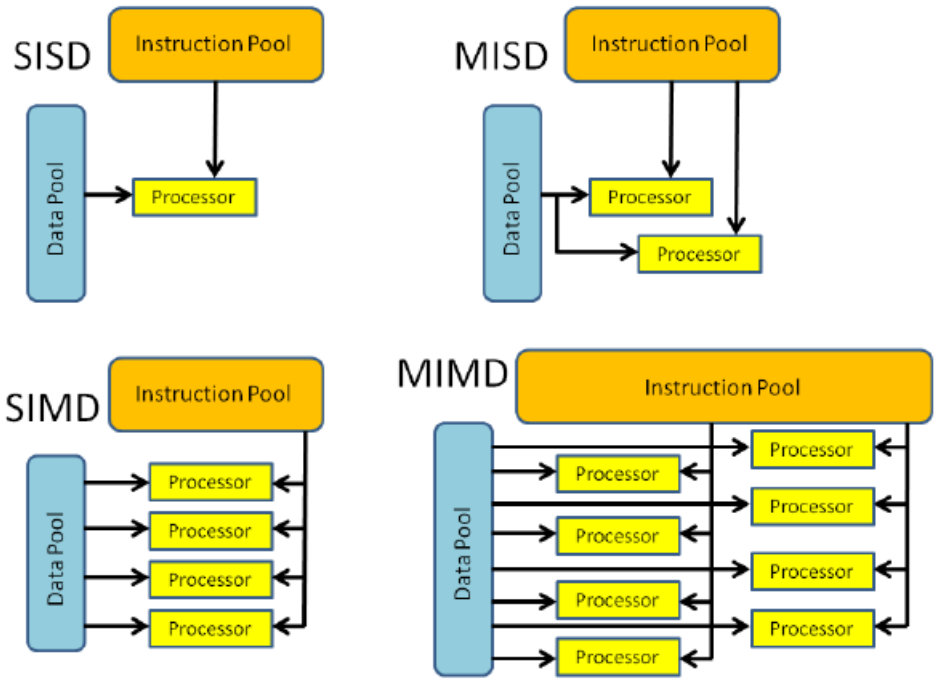
Η ταξινομία του Flynn [3] είναι μια ταξινόμηση των αρχιτεκτονικών υπολογιστών. Το σύστημα ταξινόμησης έχει επικρατήσει και έχει χρησιμοποιηθεί ως εργαλείο στο σχεδιασμό σύγχρονων επεξεργαστών και των λειτουργιών τους. Από την άνοδο των κεντρικών μονάδων επεξεργασίας πολλαπλών επεξεργαστών (CPU), ένα περιβάλλον πολλαπλού προγραμματισμού έχει εξελιχθεί ως επέκταση του συστήματος ταξινόμησης.

Οι τέσσερις ταξινομήσεις που ορίζονται από τον Flynn βασίζονται στον αριθμό των ταυτόχρονων ροών εντολών (insrtuctions) και των ροών δεδομένων (data) που διατίθενται στην αρχιτεκτονική.

- **Single instruction stream, single data stream (SISD):** Ένας ακολουθιακός υπολογιστής που δεν εκμεταλλεύεται κανένα παραλληλισμό ούτε στις οδηγίες ούτε στις ροές δεδομένων. Η μονάδα ενιαίου ελέγχου (CU) ανακτά μία ροή εντολών (IS) από τη μνήμη. Στη συνέχεια, η CU παράγει κατάλληλα σήματα ελέγχου για να κατευθύνει ένα μόνο στοιχείο επεξεργασίας (PE) ώστε να λειτουργεί σε μία ροή δεδομένων (DS), δηλαδή μία λειτουργία τη φορά. Χρησιμοποιούνταν παλιά με τους μονοπύρηνους προσωπικούς επεξεργατές.
- **Single instruction stream, multiple data streams (SIMD):** Μια μεμονωμένη οδηγία λειτουργεί σε πολλές διαφορετικές ροές δεδομένων. Οι οδηγίες μπορούν να εκτελεστούν διαδοχικά, όπως μέσω σωληνώσεων ή παράλληλα από πολλαπλές λειτουργικές μονάδες. Στην κατηγορία αυτή εντάσσεται και το

SIMT, δηλαδή το single instruction multiple threads, που είναι συνδυασμός του SIMD με multithreading.

- **Multiple instruction streams, single data stream (MISD):** Πολλές οδηγίες λειτουργούν σε μία ροή δεδομένων. Πρόκειται για μια ασυνήθιστη αρχιτεκτονική που χρησιμοποιείται γενικά για ανοχή σφαλμάτων. Τα ετερογενή συστήματα λειτουργούν στην ίδια ροή δεδομένων και πρέπει να συμφωνήσουν για το αποτέλεσμα.
- **Multiple instruction streams, multiple data streams (MIMD):** Πολλοί αυτόνομοι επεξεργαστές εκτελούν ταυτόχρονα διαφορετικές οδηγίες σε διαφορετικά δεδομένα. Οι αρχιτεκτονικές MIMD [4] περιλαμβάνουν επεξεργαστές πολλαπλών πυρήνων superscalar και κατανεμημένα συστήματα, χρησιμοποιώντας είτε έναν κοινόχρηστο χώρο μνήμης είτε έναν κατανεμημένο χώρο μνήμης. Αυτή η αρχιτεκτονική είναι η κυρίαρχη στις μέρες μας.



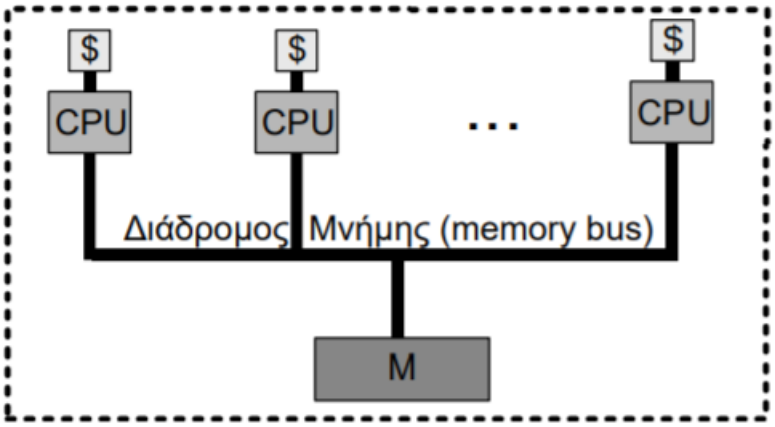
Σχήμα 1: Ταξινόμηση κατά Flynn

2.3 Παράλληλες Αρχιτεκτονικές – Οργάνωση μνήμης

Τα MIMD , χωρίζονται σε τρεις βασικές κατηγορίες, όπως και όλες οι παράλληλες αρχιτεκτονικές, αναλόγως με τον τρόπο που είναι οργανωμένη η μνήμη : αρχιτεκτονική κοινής μνήμης, αρχιτεκτονική κατανεμημένης μνήμης και υβριδική αρχιτεκτονική.

2.3.1 Κοινής μνήμης – Shared Memory

Στην επιστήμη των υπολογιστών, η κοινή μνήμη είναι η μνήμη που μπορεί να προσπελαστεί ταυτόχρονα από πολλά προγράμματα με σκοπό την επικοινωνία μεταξύ τους ή την αποφυγή περιττών αντιγράφων. Η κοινή μνήμη είναι ένας αποτελεσματικός τρόπος μετάδοσης δεδομένων μεταξύ προγραμμάτων. Ανάλογα με το περιβάλλον, τα προγράμματα ενδέχεται να εκτελούνται σε έναν μόνο επεξεργαστή ή σε πολλούς ξεχωριστούς επεξεργαστές. Συνήθως η διασύνδεση γίνεται μέσω διαδρόμου μνήμης (memory bus), αλλά μπορεί να χρησιμοποιούνται και πιο εξελιγμένα δίκτυα διασύνδεσης. Σημειώνεται πως ενώ οι επεξεργαστές έχουν κοινή μνήμη, κάθε επεξεργαστής διαθέτει τοπική ιεραρχία κρυφών μνημών. Το βασικό πλεονέκτημα της αρχιτεκτονικής αυτής είναι ότι διευκολύνει τον παράλληλο προγραμματισμό (προγραμματισμός σε OPENMP) αλλά μπορεί να δημιουργήσει προβλήματα λόγω race conditions. Είναι δύσκολα κλιμακώσιμη αρχιτεκτονική, τυπικά μέχρι λίγες δεκάδες κόμβους.



Σχήμα 2: Αρχιτεκτονική κοινής μνήμης

Στο υλικό του υπολογιστή, η κοινόχρηστη μνήμη αναφέρεται σε ένα σχετικά μεγάλο μπλοκ μνήμης τυχαίας προσπέλασης (RAM) που μπορεί να προσεγγιστεί από πολλές διαφορετικές κεντρικές μονάδες επεξεργασίας (CPU) σε ένα σύστημα υπολογιστών πολλαπλών επεξεργαστών.

Τα συστήματα κοινής μνήμης μπορεί να χρησιμοποιούν τους εξής τρόπους προσπέλασης της μνήμης:

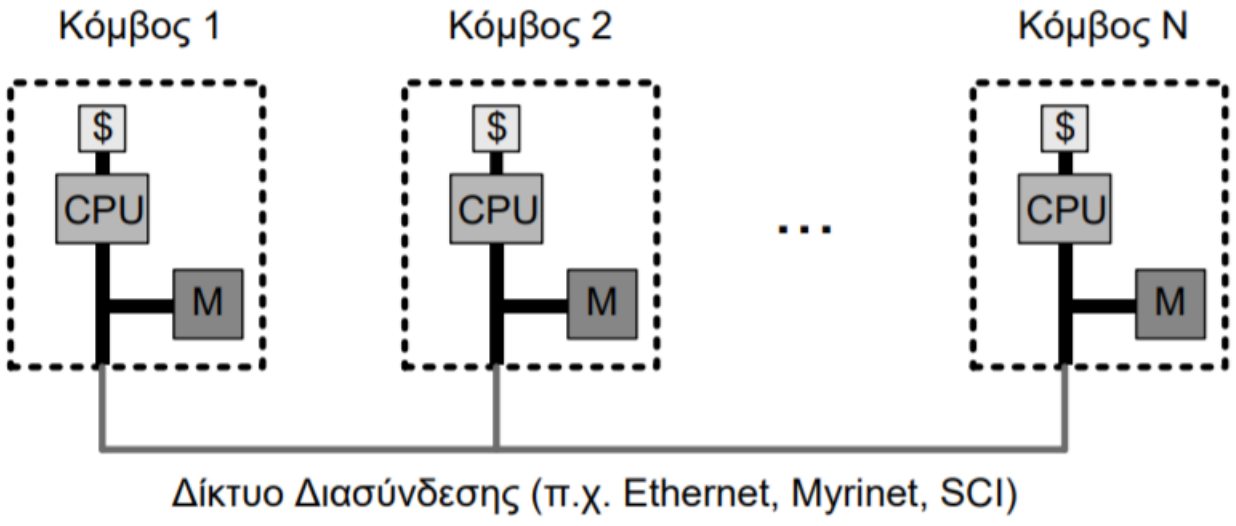
- **UMA (uniform memory access):** Όλοι οι επεξεργαστές έχουν πρόσβαση ομοιόμορφα στην φυσική μνήμη.

- **NUMA (non-uniform memory access):** Ο χρόνος πρόσβασης στη μνήμη εξαρτάται από τη θέση μνήμης σε σχέση με έναν επεξεργαστή.
- **COMA (cache-only memory architecture):** Οι τοπικές μνήμες για τους επεξεργαστές σε κάθε κόμβο χρησιμοποιούνται ως προσωρινή μνήμη αντί ως πραγματική κύρια μνήμη.

2.3.2 Κατανεμημένης μνήμης – Distributed Memory

Στην επιστήμη των υπολογιστών, η κατανεμημένη μνήμη αναφέρεται σε ένα σύστημα υπολογιστών πολλαπλών επεξεργαστών στο οποίο κάθε επεξεργαστής έχει τη δική του ιδιωτική μνήμη. Οι υπολογιστικές εργασίες μπορούν να λειτουργούν μόνο σε τοπικά δεδομένα και εάν απαιτούνται απομακρυσμένα δεδομένα, η υπολογιστική εργασία πρέπει να επικοινωνεί με έναν ή περισσότερους απομακρυσμένους επεξεργαστές.

Σε ένα κατανεμημένο σύστημα μνήμης υπάρχει συνήθως ένας επεξεργαστής, μια μνήμη και κάποια μορφή διασύνδεσης που επιτρέπει σε προγράμματα σε κάθε επεξεργαστή να αλληλεπιδρούν μεταξύ τους. Η διασύνδεση μπορεί να οργανωθεί με συνδέσμους από σημείο σε σημείο ή ξεχωριστό υλικό μπορεί να παρέχει ένα δίκτυο εναλλαγής. Η τοπολογία του δικτύου είναι ένας βασικός παράγοντας για τον καθορισμό του τρόπου κλιμάκωσης της μηχανής πολλαπλών επεξεργαστών.



Σχήμα 3: Αρχιτεκτονική Κατανεμημένης Μνήμης

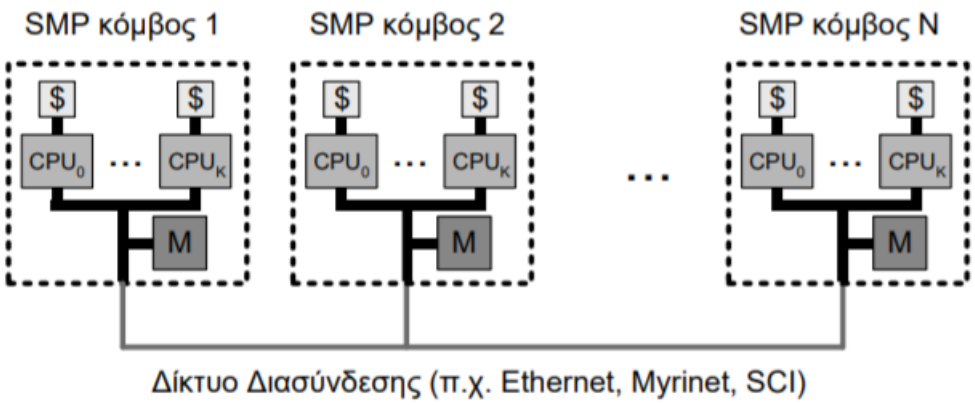
Το βασικό ζήτημα στον προγραμματισμό κατανεμημένων συστημάτων μνήμης είναι ο τρόπος διανομής των δεδομένων μέσω των μνημών (προγραμματισμός σε

MPI). Ανάλογα με το πρόβλημα που επιλύεται, τα δεδομένα μπορούν να διανεμηθούν στατικά ή να μετακινηθούν μέσω των κόμβων. Τα δεδομένα μπορούν να μετακινηθούν κατά απαίτηση ή τα δεδομένα μπορούν να προωθηθούν στους νέους κόμβους εκ των προτέρων. Τα δεδομένα μπορούν να διατηρηθούν στατικά σε κόμβους εάν οι περισσότεροι υπολογισμοί συμβαίνουν τοπικά και μόνο αλλαγές στις άκρες πρέπει να αναφέρονται σε άλλους κόμβους.

Έχει το μειονέκτημα πως δυσκολεύει τον προγραμματισμό γιατί ο προγραμματιστής απαιτείται να σχεδιάσει και να υλοποιήσει την πρόσβαση σε διακριτές μνήμες (κατακερματισμένος προγραμματισμός). Ωστόσο, σε αντίθεση με την αρχιτεκτονική κοινής μνήμης, κλιμακώνει σε χιλιάδες υπολογιστικούς κόμβους.

2.3.3 Υβριδική αρχιτεκτονική – Hybrid Architecture

Η υβριδική αρχιτεκτονική συνδυάζει τις δύο παραπάνω αρχιτεκτονικές.



Σχήμα 4: Υβριδική αρχιτεκτονική

Στην αρχιτεκτονική αυτή, κόμβοι με αρχιτεκτονική κοινής μνήμης διασυνδέονται με ένα δίκτυο διασύνδεσης σε αρχιτεκτονική κατανεμημένης μνήμης. Για το λόγο αυτό αποτελεί την τυπική αρχιτεκτονική των σύγχρονων συστοιχιών-υπερυπολογιστών, των data centers και των υποδομών cloud, αφού συνδυάζει τα πλεονεκτήματα των δύο προηγούμενων αρχιτεκτονικών.

2.4 Νομος του Amdahl

Οι υπερυπολογιστές χρησιμοποιούνται για να επιλυθούν μεγάλα προβλήματα σε εύλογο χρονικό διάστημα. Αυτό συμβαίνει γιατί χρησιμοποιούν παράλληλο προγραμματισμό. Ένα σειριακό πρόγραμμα εκτελείται σε έναν πυρήνα ενός υπολογιστή και συνεπώς η ταχύτητά του εξαρτάται αποκλειστικά από την δύναμη του επεξεργαστή αυτού. Σκοπός του παράλληλου προγραμματισμού είναι να μειωθεί ο χρόνος που απαιτείται για να επιλυθεί ένα πρόβλημα σε σχέση με τον σειριακό του χρόνο. Αυτό θα επιτευχθεί επειδή το πρόγραμμα θα επιλυθεί ταυτόχρονα σε παραπάνω από έναν πυρήνα ή ακόμα και σε παραπάνω από έναν επεξεργαστές.

Για να εξετασθεί η αποτελεσματικότητα του παράλληλου προγραμματισμού έχουν οριστεί κάποιες μεταβλητές ως εξής:

Έστω ένα πρόγραμμα, το οποίο έχει έναν αλγόριθμο επίλυσης.

Ο χρόνος του καλύτερου σειριακού αλγορίθμου συμβολίζεται με T_s.

Ο χρόνος του παράλληλου αλγορίθμου ορίζεται με T_p.

Στόχος είναι το παράλληλο πρόγραμμα να είναι πολλές φορές πιο γρήγορο από το σειριακό, δηλαδή να υπάρξει μεγάλη επιτάχυνση.

Επιτάχυνση (speedup) $S = T_s/T_p$.

Το μέγεθος αυτό δείχνει πόσες φορές είναι πιο γρήγορο το παράλληλο πρόγραμμα από το σειριακό.

Αυτό που επιδιώκεται σε κάθε παράλληλο πρόγραμμα είναι η ιδανική επιτάχυνση που είναι η γραμμική. Όσο πολλαπλασιάζεται το πλήθος των επεξεργαστών τόσο να υποπολλαπλασιάζεται ο χρόνος του προβλήματος και συνεπώς να έχουμε μια γραμμική επιτάχυνση. Αν έχουμε p επεξεργαστές δηλαδή, θέλουμε η επιτάχυνση S να είναι περίπου ίση με το p.

Στην πράξη, η γραμμική επιτάχυνση επιτυγχάνεται σπάνια. Στη γενική περίπτωση ισχύει $S \leq p$. Αυτό συμβαίνει για διάφορους λόγους, κάποιους από τους οποίους ίσως να μπορεί να βελτιώσει ο προγραμματιστής ενώ άλλους όχι. Υπάρχουν και περιπτώσεις που συναντάται superlinear speedup, όταν δηλαδή $S > p$, κάτι το οποίο χρήζει ερμηνείας όποτε συμβαίνει.

Ορίζεται επίσης το μέγεθος της αποδοτικότητας (efficiency) $E = S / p$, το οποίο δείχνει πόσο επιτυχημένη είναι η παραλληλοποίηση και για το οποίο αν δεν έχουμε superlinear speedup ισχύει $E \leq 1$.

Το επιδιωκόμενο λοιπόν στον παράλληλο προγραμματισμό είναι να έχουμε καλή κλιμακωσιμότητα. Αυτό σημαίνει πως ένα πρόγραμμα θα

βελτιώνει την επίδοσή του με την προσθήκη επιπλέον πόρων, δηλαδή επεξεργαστών. Το ιδανικό θα ήταν το πρόγραμμα να μπορεί να βελτιώνεται επ' άπειρον όσο αυξάνεται ο αριθμός των επεξεργαστών.

Όλα τα προσδοκώμενα ιδανικά όρια όμως στην πράξη δεν επιτυγχάνονται, καθώς συναντούν ένα σημαντικό άνω φράγμα που ονομάζεται νόμος του Amdahl [5] .

Έστω T_s ο χρόνος του καλύτερου σειριακού αλγορίθμου.
Έστω f το κλάσμα του χρόνου ενός σειριακού προγράμματος που δεν παραλληλοποιείται και συνεπώς 1-f αυτό που παραλληλοποιείται. Ο νόμος του Amdahl λέει:

$T_p = f*T_s + (1 - f) * Ts / p$ και συνεπώς

$$S = T_s / T_p = \frac{1}{f + \frac{1-f}{p}}$$

Αυτό σημαίνει πως ένα πρόβλημα μπορούμε να το παραλληλοποιήσουμε όσο επιδέχεται παραλληλισμό και άρα πρέπει να αναζητεί ο προγραμματιστής παραλληλισμό παντού.

Συνεπώς, ένα πρόγραμμα μπορεί όταν παραλληλοποιηθεί να μην καταφέρει να επιτύχει επιτάχυνση S = 1 καθώς περιορίζεται από το νόμο του Amdahl. Εκτός αυτού, βασικά εμπόδια που προκύπτουν σε ένα παράλληλο πρόγραμμα και δεν υπάρχουν σε ένα σειριακό είναι η ανάγκη για πρόσβαση στη μνήμη των διαφορετικών πυρήνων ή επεξεργαστών (memory intensive) και η ανάγκη για επικοινωνία μεταξύ τους (communication traffic).

2.5 Τομείς εφαρμογής των HPC

Στη σημερινή εποχή, τα HPC εξυπηρετούν ερευνητές που ασχολούνται με προβλήματα σε διάφορους τομείς και οργανισμούς και επιχειρήσεις που το έργο τους εξαπλώνεται σε πολλές πτυχές της ζωής. Αυτό τα καθιστά πολύ σημαντικά, καθώς με τις μεγάλες δυνατότητες τους δίνουν λύσεις σε σημαντικά προβλήματα της ανθρωπότητας. Σημειώνονται κάποιες από τις εφαρμογές τους.

- **Βιοεπιστήμες και ανθρώπινο γονιδίωμα:** Ανακάλυψη φαρμάκων, ανίχνευση και πρόληψη ασθενειών

- **Μηχανική με τη βοήθεια υπολογιστών (CAE):** Σχεδιασμός αυτοκινήτων και δοκιμές, μεταφορά, δομικός και μηχανικός σχεδιασμός
- **Χημική μηχανική:** Διαδικασία και μοριακός σχεδιασμός
- **Δημιουργία ψηφιακού περιεχομένου (DCC) και διανομή:** Γραφικά που υποστηρίζονται από υπολογιστή σε φιλμ και μέσα
- **Οικονομικα-χρηματοοικονομικά:** Ανάλυση κινδύνου Wall Street, διαχείριση χαρτοφυλακίου, αυτοματοποιημένες συναλλαγές
- **Ηλεκτρονικός σχεδιασμός και αυτοματισμός (EDA):** Ηλεκτρονικός σχεδιασμός και επαλήθευση εξαρτημάτων
- **Γεωεπιστήμες και γεωμηχανική:** Εξερεύνηση πετρελαίου και φυσικού αερίου και μοντελοποίηση δεξαμενών
- **Μηχανικός σχεδιασμός και σύνταξη:** 2D και 3D σχεδιασμός και επαλήθευση, μηχανική μοντελοποίηση
- **Άμυνα και ενέργεια:** Πυρηνική διαχείριση, βασική και εφαρμοσμένη έρευνα
- **Κυβερνητικά εργαστήρια:** Βασική και εφαρμοσμένη έρευνα
- **Πανεπιστήμια:** Βασική και εφαρμοσμένη έρευνα
- **Πρόγνωση καιρού:** Μοντελοποίηση καιρού και κλίματος

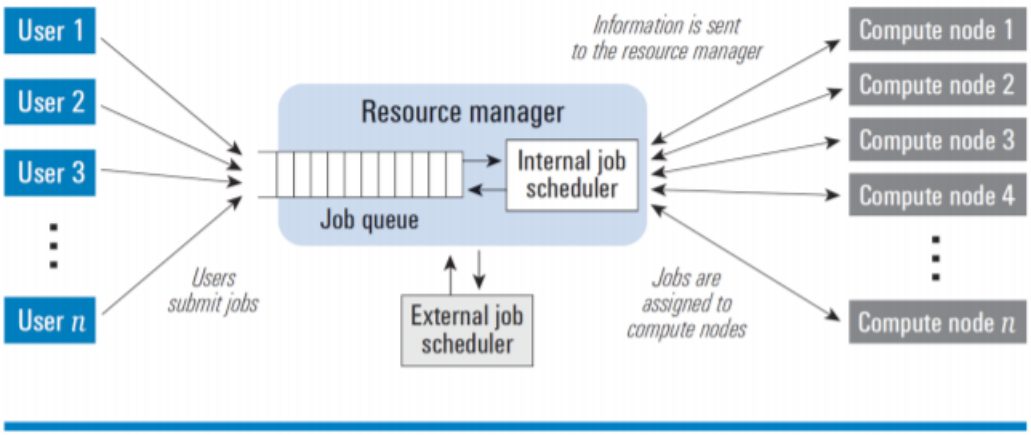
3. Χρονοδρομολογητές

3.1 Resource Manager

Τα μεγάλα υπολογιστικά συστήματα απαιτούν σημαντικά χρηματικά ποσά για την εγκατάστασή τους και τη λειτουργία τους. Η σωστή διαχείριση των πόρων τους αποτελεί βασική επιδίωξη των ιδιοκτητών τους όπως επίσης και η βελτιστοποίηση της χρήσης τους.

Την λειτουργία αυτή, αναλαμβάνουν να επιτελούν τα συστήματα διαχείρισης πόρων (resource management systems) [6]. Τα συστήματα αυτά, αποτελούν αναπόσπαστο κομμάτι του λογισμικού των υπερυπολογιστών. Πραγματοποιούν τρεις βασικές λειτουργίες: κατανομή πόρων, χρονοδρομολόγηση εργασιών και παρακολούθηση της εκτέλεσής τους.

Στην παρακάτω εικόνα φαίνεται η γενική εικόνα ενός συστήματος διαχείρισης πόρων. Σημειώνεται πως περιλαμβάνουν ένα job queue και έναν εσωτερικό job scheduler καθώς επίσης έχουν επαφή με τους χρήστες και με τους πόρους του συστήματος αλλά και με έναν εξωτερικό job scheduler. Το σύστημα διαχείρισης πόρων του cluster της σχολής είναι το Torque, όπου αναλύεται στη συνέχεια όπως επίσης και το βασικό μέρος των συστημάτων διαχείρισης πόρων που είναι οι χρονοδρομολογητές.



Σχήμα 5: Αναπαράσταση ενός συστήματος διαχείρισης πόρων

Ένα σύστημα διαχείρισης πόρων αναγνωρίζει τις εξής διαφορετικές κατηγορίες πόρων: υπολογιστικοί κόμβοι, πυρήνες επεξεργασίας, διασυνδέσεις, μέσα αποθήκευσης, επιταχυντές. Περισσότερα για τα συστήματα διαχείρισης πόρων υπάρχουν στο [6].

3.2 Γενικά για τους χρονοδρομολογητές

Οι χρονοδρομολογητές εφαρμόζονται συχνά, ώστε να διατηρούν όλους τους πόρους απασχολημένους (όπως στην εξισορρόπηση φορτίου), να επιτρέπουν σε πολλούς χρήστες να μοιράζονται αποτελεσματικά τους πόρους του συστήματος ή να επιτυγχάνουν μια στοχευμένη ποιότητα υπηρεσιών.

Ένας χρονοδρομολογητής μπορεί να στοχεύει σε έναν ή περισσότερους στόχους, όπως για παράδειγμα: μεγιστοποίηση της απόδοσης (το συνολικό ποσό της εργασίας που ολοκληρώνεται ανά μονάδα χρόνου), ελαχιστοποίηση του χρόνου αναμονής (χρόνος από την προετοιμασία της εργασίας μέχρι το πρώτο σημείο που ξεκινά την εκτέλεση), ελαχιστοποίηση του λανθάνοντος χρόνου ή του χρόνου απόκρισης (χρόνος από την προετοιμασία της εργασίας μέχρι την ολοκλήρωσή της σε περίπτωση μαζικής δραστηριότητας, ή έως ότου το σύστημα ανταποκριθεί και παραδώσει την πρώτη έξοδο στον χρήστη σε περίπτωση διαδραστικής δραστηριότητας), μεγιστοποίηση της δικαιοσύνης (ίσος χρόνος CPU σε κάθε διαδικασία ή γενικότερα κατάλληλοι χρόνοι σύμφωνα με την προτεραιότητα και τον φόρτο εργασίας κάθε διαδικασίας). Στην πράξη, αυτοί οι στόχοι συχνά συγκρούονται (π.χ. απόδοση έναντι καθυστέρησης), επομένως ένας χρονοδρομολογητής θα εφαρμόσει έναν κατάλληλο συμβιβασμό. Η προτίμηση μετράται από οποιαδήποτε από τις ανησυχίες που αναφέρονται παραπάνω, ανάλογα με τις ανάγκες και τους στόχους του χρήστη.

Οι αλγόριθμοι για χρονοδρομολόγηση μπορούν να χωριστούν σε δύο κατηγορίες: time-sharing , space-sharing.

- **Time-sharing:** Διαιρούν τον χρόνο σε έναν επεξεργαστή σε αρκετά διακριτά διαστήματα και θέσεις. Αυτές οι θέσεις εκχωρούνται στη συνέχεια σε μοναδικές εργασίες
- **Space-sharing:** Παρέχουν τους απαιτούμενους πόρους σε μια μόνο εργασία μέχρι αυτή να ολοκληρώσει την εκτέλεσή της. Οι περισσότεροι cluster schedulers χρησιμοποιούν space-sharing λειτουργία.

Οι κύριοι στόχοι των αλγορίθμων χρονοδρομολόγησης είναι η ελαχιστοποίηση της έλλειψης πόρων και η διασφάλιση της δικαιοσύνης μεταξύ των μερών που χρησιμοποιούν τους πόρους. Η χρονοδρομολόγηση έρχεται αντιμέτωπη με το πρόβλημα να αποφασιστεί σε ποια από τις εκκρεμείς αιτήσεις θα διατεθούν πόροι. Υπάρχουν πολλοί διαφορετικοί αλγόριθμοι χρονοδρομολόγησης [7].

- **First come, first served:** Ο αυτόματος και απλούστερος αλγόριθμος που έχουν οι περισσότεροι χρονοδρομολογητές που είναι γνωστότερος ως first in first out. Στον αλγόριθμο αυτό, ο χρονοδρομολογητής απλώς βάζει τις διεργασίες με τη σειρά που έρχονται. Η τακτική αυτή έχει το πλεονέκτημα της απλότητας και της ταχύτητας ανάθεσης εργασιών, καθώς κρατείται απλά μια ουρά με τη σειρά που ήρθαν οι εργασίες. Ωστόσο, η απόδοση σε πολλές περιπτώσεις είναι χαμηλή, αφού εργασίες που χρειάζονται πολύ λίγο χρόνο μπορεί να περιμένουν στην ουρά.
- **Priority scheduling:** Η πρώτη προθεσμία πρώτα – earliest deadline first (EDF) ή ο ελάχιστος χρόνος ολοκλήρωσης, είναι ένας δυναμικός αλγόριθμος χρονοδρομολόγησης, που χρησιμοποιείται σε λειτουργικά συστήματα ενώ ήδη τρέχουν, για να τοποθετήσει διαδικασίες σε ουρά προτεραιότητας. Κάθε φορά που εμφανίζεται ένα συμβάν χρονοδρομολόγησης (μια εργασία τελειώνει, μια νέα εργασία εμφανίζεται), η ουρά θα αναζητηθεί για τη διεργασία που πλησιάζει στην προθεσμία της, η οποία θα είναι η επόμενη που θα προγραμματιστεί για εκτέλεση.
- **Shortest remaining time first:** Με αυτήν την στρατηγική, ο χρονοδρομολογητής οργανώνει διαδικασίες με τον ελάχιστο εκτιμώμενο χρόνο επεξεργασίας που απομένει να είναι ο επόμενος στην ουρά. Αυτό απαιτεί προηγμένες γνώσεις ή εκτιμήσεις σχετικά με το χρόνο που απαιτείται για την ολοκλήρωση μιας διαδικασίας.
- **Round robin scheduling:** Ο χρονοδρομολογητής εκχωρεί μια σταθερή μονάδα χρόνου στις διεργασίες και κινείται κυκλικά σε αυτές για αυτό το χρονικό διάστημα σε κάθε διεργασία. Εάν η διαδικασία ολοκληρωθεί εντός αυτού του χρονικού διαστήματος τερματίζεται, αλλιώς επαναπρογραμματίζεται αφού πρώτα πάρουν χρόνο όλες οι υπόλοιπες διαδικασίες.
- **Fixed priority pre-emptive scheduling:** Το λειτουργικό σύστημα εκχωρεί μια σταθερή κατάταξη προτεραιότητας σε κάθε διαδικασία και ο χρονοδρομολογητής τακτοποιεί τις διεργασίες στην έτοιμη ουρά σύμφωνα με

την προτεραιότητά τους. Οι διαδικασίες χαμηλότερης προτεραιότητας διακόπτονται από τις εισερχόμενες διαδικασίες υληλότερης προτεραιότητας.

3.3 Χρονοδρομολογητής εργασιών – Job Scheduler

Ο χρονοδρομολογητής εργασιών [8] είναι μια εφαρμογή υπολογιστή για τον έλεγχο της εκτέλεσης των εργασιών χωρίς παρακολούθηση στο παρασκήνιο. Αυτό ονομάζεται συνήθως batch scheduling, καθώς η εκτέλεση μη διαδραστικών jobs συχνά ονομάζεται batch processing, αν και το job και batch διακρίνονται. Η δομή από εργασίες για εκτέλεση είναι γνωστή ως ουρά εργασίας.

Οι σύγχρονοι χρονοδρομολογητές παρέχουν συνήθως ένα γραφικό περιβάλλον εργασίας χρήστη και ένα μόνο σημείο ελέγχου για τον ορισμό και την παρακολούθηση των εκτελέσεων στο παρασκήνιο σε ένα κατανεμημένο δίκτυο υπολογιστών. Όλο και περισσότερο, οι χρονοδρομολογητές εργασίας απαιτούνται για την ενοποίηση της ολοκλήρωσης επιχειρηματικών δραστηριοτήτων σε πραγματικό χρόνο με την παραδοσιακή επεξεργασία πληροφορικής στο υπόβαθρο σε διαφορετικές πλατφόρμες λειτουργικού συστήματος και περιβάλλοντα επιχειρηματικών εφαρμογών.

Η έννοια του job scheduler δεν πρέπει να συγχέεται με την χρονοδρομολογηση διεργασιών (process scheduling), που είναι η εκχώρηση διεργασιών που εκτελούνται αυτήν τη στιγμή σε CPU από το λειτουργικό σύστημα.

Κατά τη διάρκεια των scheduling events, ένας αλγόριθμος πρέπει να κάνει assign nodes σε jobs. Οι πιο γνωστοί χρονοδρομολογητές είναι οι εξής:

◦ Simple Linux Utility for Resource Management (SLURM)

◦ Maui Cluster Scheduler (Maui)

◦ Moab High-Performance Computing Suite (Moab)

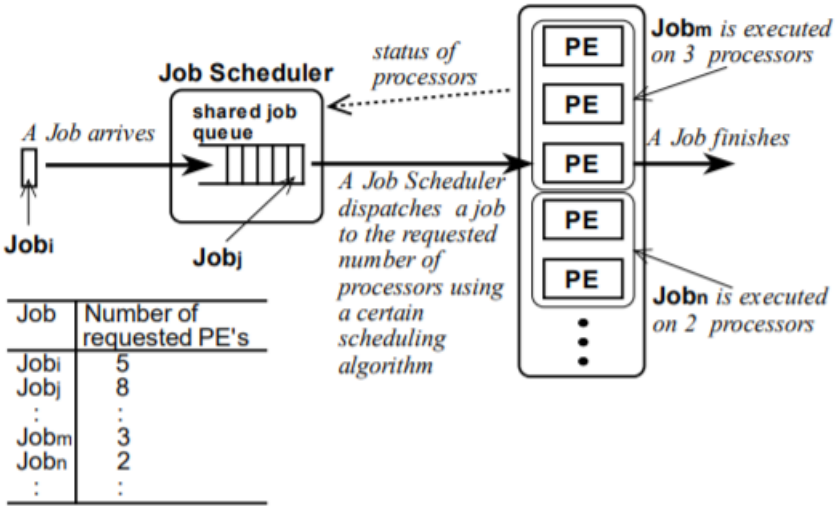
◦ TORQUE

◦ Portable Batch System (PBS) [OpenPBS, TORQUE, PBS Pro]

◦ Globus toolkit

◦ Open MPI

Στο σχήμα 6 [9] φαίνεται ο τρόπος λειτουργίας ενός Job Scheduler.



Σχήμα 6: Τρόπος λειτουργίας Job Scheduler

3.4 Co scheduling

Οι περισσότερες παράλληλες επιστημονικές εφαρμογές έχουν συχνές φάσεις επικοινωνίας. Οποιαδήποτε ανισορροπία στους χρόνους υπολογισμού οδηγεί σε χρόνους αναμονής, με αποτέλεσμα να μειώνεται σε μεγάλο βαθμό η κλιμακωσιμότητα. Ο ευκολότερος τρόπος για να εξασφαλιστεί ισορροπημένη απόδοση είναι η εκχώρηση ειδικών πόρων στο επίπεδο του κόμβου.

Για να αποφευχθεί το παραπάνω πρόβλημα, εισάγεται η έννοια και η τακτική του co-scheduling που μπορεί να χρησιμοποιηθεί σε πολλές περιπτώσεις.

Το co-scheduling, είναι η αρχή για ταυτόχρονα συστήματα να χρονοδρομολογούν εργασίες να τρέχουν σε διαφορετικούς επεξεργαστές ή πυρήνες την ίδια ώρα, παράλληλα. Με άλλα λόγια, το co-scheduling έχει ως αρχή να μην διαθέτει εξ’ ολοκλήρου τους πόρους ενός συστήματος σε μια εφαρμογή, αν αυτή δεν χρειάζεται να τους χρησιμοποιήσει όλους, αλλά να προσπαθεί να τους μοιράσει σε παραπάνω εφαρμογές, αν είναι εφικτό αυτές να συνδυαστούν και συνολικά να αυξηθεί η απόδοση του συστήματος.

Όταν μια συγκεκριμένη εφαρμογή, χρειάζεται να τρέξει πολλές φορές η ίδια συνεχόμενα στους πόρους του συστήματος, αυτό ίσως να μειώνει την συνολική

επίδοση και απόδοση του συστήματος. Αυτό μπορεί να συμβαίνει γιατί η εφαρμογή αυτή θα χρησιμοποιεί συγκεκριμένους πόρους του συστήματος με αποτέλεσμα αυτοί οι πόροι να υπερφορτώνονται, ενώ οι υπόλοιποι να μένουν ανεκμετάλλευτοι.

Οι κόμβοι ενός πραγματικού συστήματος γίνονται πιο παράλληλοι και γρήγοροι, φιλοξενώντας δεκάδες πυρήνες και πρόσθετα εξαρτήματα επιταχυντή. Αυτό δημιουργεί την ανάγκη για τις εφαρμογές να χρησιμοποιούν όλα τα συστατικά των ολοένα και πιο ετερογενών κόμβων αποτελεσματικά.

Το co-scheduling προσπαθεί λοιπόν να επιτύχει καλά ισορροπημένους κόμβους (nodes), ώστε να βελτιωθεί η συνολική απόδοση του συστήματος και να επιτευχθεί υψηλό throughput και χαμηλή κατανάλωσης ενέργειας. Ταυτόχρονα, στόχος είναι η μείωση του συνολικού χρόνου, δηλαδή των συνολικών “ωρών πυρήνων” (core hours) που το σύστημα θα δουλεύει, καθώς στους υπερυπολογιστές, ένας ερευνητής ή οποιοσδήποτε αποκτά πρόσβαση για να τα χρησιμοποιήσει, χρεώνεται με τις συνολικές ώρες που έτρεχαν οι πυρήνες τους οποίους είχε δεσμεύσει.

Το co-scheduling λοιπόν, προσπαθεί να συνδυάσει κυρίως εφαρμογές με ετερογενείς απαιτήσεις σε πόρους. Ένα κλασικό παράδειγμα είναι πως αν μια εφαρμογή είναι memory intensive, δηλαδή χρησιμοποιεί πολύ την μνήμη του συστήματος, κάνοντας προσβάσεις σε αυτή για ανάγνωση και εγγραφή, τότε είναι συμφέρον να συνδυαστεί με μια εφαρμογή που είναι compute intensive, δηλαδή απαιτεί επεξεργαστική ισχύ. Αυτό συμβαίνει καθώς η μια εφαρμογή δεν συγκρούεται με την άλλη αφού χρησιμοποιούν διαφορετικούς πόρους του συστήματος. Από την άλλη η απόπειρα συνδυασμού δύο memory intensive εφαρμογών μπορεί να οδηγήσει σε ολέθρια αποτελέσματα, όπως και δύο compute intensive. Αυτό όμως είναι σχετικό, καθώς το σύστημα μπορεί να έχει πολύ μεγάλη επεξεργαστική ισχύ, και οι δύο compute intensive εφαρμογές να μην πλησιάζουν τα όρια του με αποτέλεσμα να συνεργαστούν και αυτές αποτελεσματικά.

Από τα παραπάνω, προκύπτει το συμπέρασμα πως η αποτελεσματικότητα ενός co-scheduling αλγορίθμου εξαρτάται από την γνώση των χαρακτηριστικών της εφαρμογής. Αυτό μπορεί να γίνει με δύο τρόπους. Ο πρώτος είναι το άτομο που ενδιαφέρεται να τρέξει τις εφαρμογές σε κάποιο HPC cluster να γνωρίζει ο ίδιος στο σημείο που είναι εφικτό τα χαρακτηριστικα και τις απαιτήσεις των εφαρμογών του. Όταν πάει να τις υποβάλει λοιπόν, μπορεί να ενημερώσει τους αρμόδιους ώστε να χρησιμοποιηθεί κάποιος αλγόριθμος co-scheduling στο σύστημα αν θεωρείται πως αυτό θα βελτιώσει τον συνολικό χρόνο που θα τρέχουν οι εφαρμογές. Ο δεύτερος τρόπος είναι να υποβάλει το ενδιαφερόμενο άτομο τις εφαρμογές του κανονικά στο

HPC cluster, το οποίο και συνήθως συμβαίνει, και το σύστημα να καταφέρει να μάθει ή να προβλέψει τα χαρακτηριστικά των εφαρμογών και τη συμπεριφορά τους από νωρίς, και έπειτα αν κρίνει πως αξίζει να εφαρμοστεί αλγόριθμος co-scheduling να τον εφαρμόσει. Αυτό για να έχει εφαρμογή, προϋποθέτει να χρειάζεται να τρέξουν πολλές φορές οι εργασίες, ώστε στα πρώτα τρεξίματα να γίνει η εξαγωγή συμπερασμάτων για αυτές και στα επόμενα να αλλάξει η πολιτική χρονοδρομολόγησης που χρησιμοποιείται.

3.4.1 Co-scheduling, NP problem

Η διαδικασία που περιγράφηκε παραπάνω έχει πολλές προϋποθέσεις για να μπορέσει να εφαρμοστεί στον πραγματικό κόσμο αποτελεσματικά. Το co-scheduling προϋποθέτει γνώση των χαρακτηριστικών των εφαρμογών από πριν. Ωστόσο, ακόμα και αν αυτό ήταν πάντα εφικτό, μια εφαρμογή δεν χαρακτηρίζεται πάντα από ένα στοιχείο. Στην πραγματικότητα, οι περισσότερες εφαρμογές απαιτούν λίγο από όλα τα στοιχεία του συστήματος (επεξεργαστική ισχύ, προσβάσεις στη μνήμη, επικοινωνία). Συνεπώς λίγες είναι οι εφαρμογές που είναι ξεκάθαρο πως μπορούν να συνεργαστούν με άλλες και να υπάρξει βελτίωση του συνολικού χρόνου και καλύτερο utilization του συστήματος.

Ο αρχικός co-scheduling προγραμματισμός δύο εφαρμογών μπορεί στην πορεία της εκτέλεσης να χρειαστεί να αναπροσαρμοστεί. Αυτό μπορεί να συμβεί γιατί μία από τις δύο εφαρμογές θα τελειώσει πριν από την άλλη και στη θέση της θα βρεθεί μια νέα που δεν θα έχει τα χαρακτηριστικά της προηγούμενης, για τα οποία είχε αποφασιστεί αρχικά ο συνδυασμός των δύο αρχικών εφαρμογών. Αν ο co-scheduling αλγόριθμος συνδυάζει και παραπάνω από δύο εφαρμογές, αυτό οδηγεί σε μία κλάση προβλημάτων που δεν μπορούν να επιλυθούν σε πολυωνυμικό χρόνο. Η διαδικασία του να αντιστοιχίζω μια ουρά από εργασίες σε πόρους μπορεί να συγκριθεί με το travelling salesman problem, που είναι ένα NP-complete πρόβλημα [10].

3.4.2 Τεχνικές Co-scheduling

Στα πραγματικά supercomputing systems, οι χρήστες χρεώνονται για το συνολικό χρόνο που χρησιμοποιούσαν τους CPUs για τις εφαρμογές τους. Συγκεκριμένα το κόστος προκύπτει ως εξής [11]:

$$COST = k * N * P * T$$

όπου,

k : σταθερά κόστους

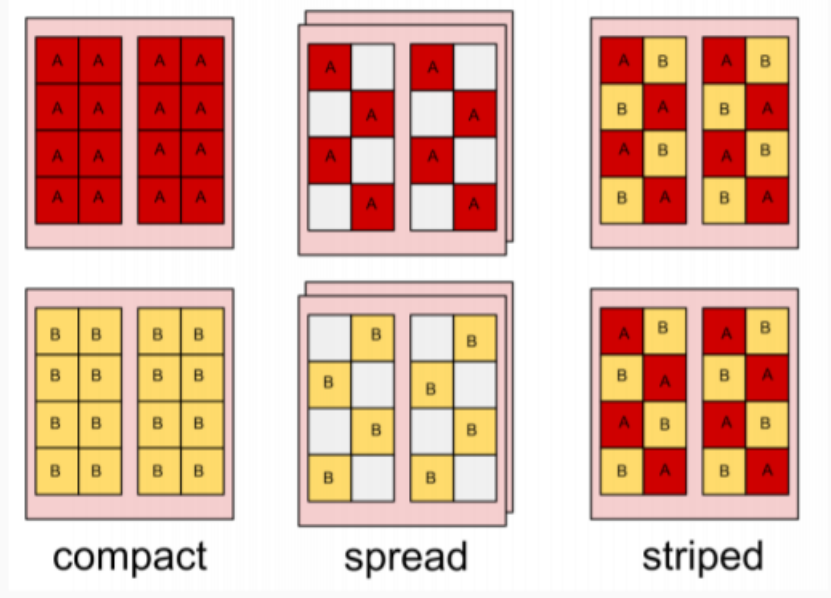
N : αριθμός nodes που χρησιμοποιήθηκαν για την δουλειά

P : αριθμός πυρήνων σε κάθε node (συνήθως τα nodes ενός συστήματος έχουν ίδιο αριθμό πυρήνων)

T : συνολικός χρόνος που έτρεχε η δουλειά.

Συνεπώς, ο χρήστης επιθυμεί να χρησιμοποιήσει όσο λιγότερους πόρους γίνεται για όσο λιγότερο χρόνο μπορεί.

Υπάρχουν τρεις βασικές πολιτικές για να αντιστοιχιστούν οι δουλειές στους πόρους ενός συστήματος: compact, spread και stripe όπως φαίνεται στο σχήμα 7 [11].



Σχήμα 7: Πολιτικές χρονοδρομολόγησης: compact, spread, striped

- **Compact:** Στην πολιτική αυτή, η δουλειά καταλαμβάνει όσους πυρήνες χρειάζεται για να εκτελεστεί συνεχόμενα από ένα node-επεξεργαστή.
- **Spread:** Στην πολιτική αυτή, η εφαρμογή γεμίζει τα μισά cores του μηχανήματος, τρέχει δηλαδή “διάσπαρτα” και χρησιμοποιεί και άλλο μηχάνημα για να συμπληρώσει τους απαιτούμενους πυρήνες που χρειάζεται.
- **Stripe:** Στην πολιτική αυτή, δύο ή περισσότερες εφαρμογές συνδυάζονται μεταξύ τους. Στην πραγματικότητα, οι εφαρμογές τρέχουν σε spread μορφή. Έτσι η κάθε μία καταλαμβάνει κάποιους πυρήνες και οι υπόλοιπες καταλαμβάνουν τους υπόλοιπους με αποτέλεσμα να μην μένουν κενοί και αχρησιμοποίητοι πυρήνες.

Η πολιτική compact έχει το πλεονέκτημα πως χρησιμοποιεί όλους τους πηρήνες ενός node, με αποτέλεσμα να επιτυγχάνεται μια καλή χρήση του συστήματος.

Η πολιτική spread χρησιμοποιεί τους μισούς πυρήνες ενός μηχανήματος (ή και λιγότερους αν το επιθυμεί ο χρήστης) και εξαπλώνεται σε παραπάνω μηχανήματα για να συμπληρώσει τους πυρήνες που χρειάζεται για να εκτελεστεί σωστά. Έχει το πλεονέκτημα έναντι της compact πως εφόσον οι ίδιοι πυρήνες κατανέμονται σε διαφορετικά μηχανήματα που το καθένα έχει τους δικούς του πόρους, αν ένα πρόβλημα χρειάζεται αυτούς τους πόρους έχει πρόσβαση σε πολλά μηχανήματα και έτσι αποφεύγονται προβλήματα και καθυστερήσεις. Εφαρμογές που είναι memory bound ή compute bound σε μια πολιτική compact, δεν έχουν πρόβλημα σε μια πολιτική spread. Έτσι η εφαρμογή εκτελείται ταχύτερα στην πολιτική spread από την πολιτική compact. Εφαρμογές που απαιτούν πολύ communication μπορεί να καθυστερούν παραπάνω σε spread πολιτική. Ωστόσο η πολιτική αυτή, ενώ είναι ταχύτερη από την compact, χρησιμοποιεί πολλούς πόρους του συστήματος και δεν κάνει καλό utilization αφού αφήνει πόρους ανεκμετάλλευτους. Για τον λόγο αυτό σε πραγματικά συστήματα δεν την βλέπουμε να χρησιμοποιείται, αφού χρεώνονται οι συνολικές ώρες που χρησιμοποιούνται πόροι του συστήματος και συνεπώς επιδιώκεται όσο μεγαλύτερο utilization είναι εφικτό.

Η πολιτική stripe αποτελεί συνδυασμό των δύο προηγούμενων. Χρησιμοποιεί όλους τους πόρους του συστήματος, συνεπώς έχει καλό utilization και ταυτόχρονα διασπά την κάθε εφαρμογή σε διάφορα μηχανήματα. Είναι πολύ καλή πολιτική για εφαρμογές που χρειάζονται περισσότερους πόρους από αυτούς που τους προσφέρει ένας μόνο κόμβος και μπορούν να συνεργαστούν μεταξύ τους, δηλαδή δεν είναι ίδιου είδους. Οι εφαρμογές τρέχουν ταυτόχρονα σε κάποιους από τους πυρήνες των

μηχανημάτων η κάθε μία με αποτέλεσμα να βελτιώνεται η συνολική επίδοση του συστήματος. Αν για παράδειγμα για μια εφαρμογή Α οι διεργασίες της χρειάζονται 6 MB από την L3 cache και έχει 8 διεργασίες συνολικά, τότε χρειάζεται συνολικά 48 MB από την L3 cache. Ωστόσο οι περισσότερες σύγχρονες L3 cache έχουν 32 MB, με αποτέλεσμα η εφαρμογή Α να κάνει πολλά cache misses αν τρέχει σε compact πολιτική και να πέφτει η συνολική επίδοση. Αν όμως η εφαρμογή Α γίνει stripe με μια εφαρμογή Β που η κάθε διεργασία της χρειάζεται 1 MB στην L3 cache τότε συνολικά θα χρησιμοποιούν 6*4 + 1*4 δηλαδή 28 MB από την L3 cache και το πρόβλημα του bottleneck για την L3 cache που είχε η εφαρμογή Α έχει λυθεί. Περισσότερα για την πολιτική stripe αναφέρονται στο [11].

3.5 Ο χρονοδρομολογητής TORQUE

Ο Terascale Open-source Resource and QUEue Manager (TORQUE) [12],[13] είναι ένας διανεμημένος διαχειριστής πόρων που διαθέτει χρονοδρομολογητή και παρέχει έλεγχο σε εργασίες δέσμης (batch jobs) και κατανεμημένους κόμβους υπολογισμού. Ο TORQUE μπορεί να ενσωματωθεί με τον μη εμπορικό Maui Cluster Scheduler ή τον εμπορικό Moab Workload Manager για τη βελτίωση της συνολικής χρήσης, της χρονοδρομολόγησης και της διαχείρισης σε ένα σύμπλεγμα.

Ο Torque ξεκίνησε να είναι ένα λογισμικό ανοιχτού κώδικα. Ωστόσο η κοινότητα TORQUE επέκτεινε το αρχικό PBS για να βελτιώσει την επεκτασιμότητα, την ανοχή σφαλμάτων και τη λειτουργικότητα. Έτσι από το 2018 δεν είναι πλέον ανοιχτού κώδικα (open source).

Βασικά χαρακτηριστικά του είναι:

- Ανοχή σε σφάλματα, μέσω ελέγχου και χειρισμού συνθηκών αστοχίας-αποτυχίας και υποστήριξη σεναρίου ελέγχου της υγείας του κόμβου
- Διεπαφή χρονοδρομολογητή. Ο TORQUE διαθέτει μια εκτεταμένη διεπαφή ερωτημάτων στον προγραμματιστή που του επιτρέπει να έχει τον έλεγχο της συμπεριφοράς και των χαρακτηριστικών της εργασίας του, καθώς επίσης να συλλέγει σημαντικές πληροφορίες και στατιστικά για τις ολοκληρωμένες εργασίες.

Βασικές εντολές του είναι οι εξής:

- **qsub** Η εντολή αυτή χρησιμοποιείται για να υποβληθούν δουλειές στον Torque . Συγκεκριμένα, για να δημιουργηθεί μια εργασία πρέπει να υποβληθεί ένα εκτελέσιμο αρχείο (script) στον διακομιστή. Η εντολή αυτή παίρνει πολλές παραμέτρους. Αν υποβληθεί με την απλή της μορφή: qsub script.sh, τότε υποβάλλεται το script.sh στον αυτόματο διακομιστή. Με την επιλογή -q υπάρχει η δυνατότητα να δηλωθεί ρητά ο διακομιστής που θα επιλεγεί, δηλαδή η ουρά στην οποία θα υποβληθεί το αρχείο, αν υπάρχουν παραπάνω από μια διαθέσιμες. Με την επιλογή -l δίνεται σαν επιλογή η λίστα με τους πόρους που θα δεσμευτούν. Μπορούν να δηλωθούν ρητά δηλαδή τα nodes που θα τρέξει το script, το πόσα cores από το καθένα, ακόμα και τα χαρακτηριστικά που πρέπει να έχουν τα nodes για να τρέξει σε αυτά η εφαρμογή. Περισσότερες επιλογές για την qsub υπάρχουν διαθέσιμες στο [14].
- **qstat** Η εντολή αυτή εμφανίζει την κατάσταση των τρέχουσων ουρών που υπάρχουν. Διαθέτει επιλογές για να πάρεις πληροφορίες για συγκεκριμένες ουρές, εργασίες ή και χρήστες.
- **qdel** Η εντολή αυτή διαγράφει μια τρέχουσα εργασία με συγκεκριμένο ID και απελευθερώνει τους πόρους που χρησιμοποιούσε.
- **mpirun** Η εντολή αυτή είναι η βασική για να τρέξεις ένα πρόγραμμα MPI, δηλαδή πρόγραμμα που χρησιμοποιεί πολλά cores και αυτά θα κατανεμηθούν είτε σε έναν είτε σε περισσότερους κόμβους. Βασικές της παράμετροι είναι οι εξής:

◆ **-N** Πόσα αντίγραφα του προγράμματος να δημιουργήσει, τα οποία θα αντιστοιχιστούν στους διαθέσιμους πόρους που έχουν δηλωθέι στο αρχείο

◆ **--map-by** Η παράμετρος αυτή σου δίνει τη δυνατότητα να διαλέξεις το πως τα N προγράμματα θα αντιστοιχιστούν (θα γίνουν bind) στους πόρους του συστήματος. Οι κυριότερες επιλογές είναι οι εξής:
by core , όπου τα προγράμματα ανατίθενται με τη σειρά στα cores του πρώτου κόμβου, μετά του δεύτερου και συνεχίζεται έτσι έως ότου αντιστοιχιστούν N αντίγραφα.
by node , όπου τα προγράμματα ανατίθενται με τη σειρά στα nodes, με κυκλικό τρόπο, έως ότου να αντιστοιχιστούν N αντίγραφα.
by socket, όπου τα προγράμματα ανατίθενται κυκλικά στα socket του κάθε κόμβου.

◆ **--rankfile** Όπου δίνεται αρχείο Rankfile, το οποίο περιέχει τον τρόπο με τον οποίο θα γίνουν bind τα προγράμματα στους πόρους του συστήματος.

Όταν γράφουμε ένα αρχείο (script) για τον Torque, η βασική λέξη που χρησιμοποιείται για να δωθούν βοηθητικές πληροφορίες στον Torque είναι η #PBS.

Αυτή ακολουθείται από παραμέτρους όπως:

- **-N <name>** , όπου δηλώνεται το όνομα που θα έχει η εργασία και θα εμφανίζεται στο queue.
- **-o <path>** , όπου δηλώνεται το μονοπάτι και το αρχείο στο οποίο θα γραφούν τα αποτελέσματα του προγράμματος.
- **-e <paht>** , όπου δηλώνεται το μονοπάτι και το αρχείο στο οποίο θα γραφούν τυχόν λάθη και προβλήματα που δημιουργήθηκαν, ή ακόμα και τα binds που έχουν γίνει αν αυτό τεθεί σαν επιλογή.

Περισσότερες επιλογές για την εντολή mpirun υπάρχουν στο [15].

```
#!/bin/bash

## Give the Job a descriptive name
#PBS -N JobName

## Output and error files
#PBS -o /home/users/.../logs/result.out
#PBS -e /home/users/.../logs/result.err

## Limit memory, runtime etc.
#PBS -l walltime=10:00:00

## How many nodes:processors_per_node should we get?
#PBS -l nodes=16:ppn=8

export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/home/users/ppeppas/mpi1/bin:$PATH
export LD_LIBRARY_PATH=/various/common_tools/gcc-4.5.2/lib64/

RANKFILE=/home/users/.../rankfile

mpirun -np 64 --rankfile $RANKFILE -v --report-bindings --timestamp-output --mca btl self,tcp /home/users/.../executable
```

Κώδικας 1: Παράδειγμα αρχείου υποβολής στον Torque

Ο TORQUE διαθέτει μια πληθώρα ακόμα εντολών με επιλογές για πιο συγκεκριμένες λειτουργείες. Οι δουλειές που υποβάλλονται σε αυτόν από τους χρήστες, μπάινουν σε μια ουρά αναμονής και με έναν αυτόματα αλγόριθμο που έχει ανατίθενται στους πόρους του συστήματος, αν αυτοί είναι επαρκείς για να καλύψουν τις απαιτήσεις της εκάστοτε δουλειάς. Αν δεν είναι τότε τοποθετούνται στην ουρά αναμονής έως ότου ελευθερωθούν οι διαθέσιμοι πόροι και ο TORQUE τους διαθέσει σε αυτές.

4. Nas Parallel Benchmarks (NPB)

4.1 Γενικά για τα Benchmarks

Οι παραδοσιακοί δείκτες αναφοράς (benchmarks) που υπήρχαν πριν από το NPB [16][17], όπως οι βρόχοι Livermore, το σημείο αναφοράς LINPACK και το πρόγραμμα συγκριτικής αξιολόγησης πυρήνα NAS, ήταν συνήθως εξειδικευμένοι για υπολογιστές φορέα. Γενικά υπέφεραν από ανεπάρκειες, συμπεριλαμβανομένων περιορισμών συντονισμού που παρεμποδίζουν τον παραλληλισμό και ανεπαρκών μεγεθών προβλημάτων, που τα καθιστούσαν ακατάλληλα για εξαιρετικά παράλληλα συστήματα. Εξίσου ακατάλληλα ήταν τα benchmark εφαρμογών πλήρους κλίμακας λόγω του υψηλού κόστους μεταφοράς και της μη διαθεσιμότητας αυτόματων εργαλείων παραλληλισμού λογισμικού. Ως αποτέλεσμα, το NPB αναπτύχθηκε το 1991 και κυκλοφόρησε το 1992 για να αντιμετωπίσει την επακόλουθη έλλειψη από benchmark που εφαρμόζονται σε εξαιρετικά παράλληλες μηχανές.

Τα NAS Parallel Benchmark (NPB) είναι ένα μικρό σύνολο προγραμμάτων που έχουν σχεδιαστεί για να βοηθήσουν στην αξιολόγηση της απόδοσης των παράλληλων υπερυπολογιστών. Αναπτύσσονται και συντηρούνται από τη NASA Advanced Supercomputing (NAS) Division (πρώην NASA Numerical Aerodynamic Simulation Program) με έδρα το NASA Ames Research Center.

Οι δείκτες αναφοράς (benchmarks) προέρχονται από εφαρμογές υπολογιστικής δυναμικής ρευστού (CFD) και αποτελούνται από πέντε πυρήνες και τρεις ψευδο-εφαρμογές στην αρχική προδιαγραφή (NPB 1). Η σουίτα των benchmarks επεκτάθηκε ώστε να περιλαμβάνει νέα benchmark για μη δομημένα προσαρμοστικά πλέγματα (adaptive meshes), παράλληλα I / O, εφαρμογές πολλαπλών ζωνών (multi-zone) και υπολογιστικά πλέγματα. Τα μεγέθη προβλημάτων στο NPB είναι προκαθορισμένα και υποδεικνύονται ως διαφορετικές κατηγορίες.

4.2 Λόγοι δημιουργίας των NPB

Σκοπός των benchmarks είναι η αξιολόγηση της επίδοσης υπερυπολογιστών. Ωστόσο αυτό για να γίνει για κοινούς και ταιριαστούς τύπους επιστημονικών υπολογισμών και προβλημάτων για όλους τους υπερυπολογιστές είναι πολύ δύσκολο. Κάποιοι άλλοι βασικοί λόγοι για την ανάπτυξη ενός ουσιαστικού benchmark για εξαιρετικά παράλληλους υπερυπολογιστές είναι οι ακόλουθοι:

- Τα προηγμένα παράλληλα συστήματα απαιτούν συχνά νέες αλγοριθμικές και λογισμικές προσεγγίσεις και αυτές οι νέες μέθοδοι είναι συχνά αρκετά διαφορετικές από τις συμβατικές μεθόδους που εφαρμόζονται στον πηγαίο κώδικα για μια ακολουθιακή ή διανυσματική μηχανή.
- Τα κριτήρια αξιολόγησης (benchmarks) πρέπει να είναι γενικά και να μην ευνοούν κάποια παράλληλη αρχιτεκτονική. Αυτή η απαίτηση αποκλείει τη χρήση οποιουδήποτε κώδικα για συγκεκριμένη αρχιτεκτονική, όπως κώδικα μετάδοσης μηνυμάτων.
- Η ορθότητα των αποτελεσμάτων και των σχημάτων απόδοσης πρέπει να επαληθεύεται εύκολα. Αυτή η απαίτηση συνεπάγεται ότι τόσο τα σύνολα δεδομένων εισόδου όσο και εξόδου πρέπει να διατηρούνται πολύ μικρά. Αυτό συνεπάγεται επίσης ότι η φύση του υπολογισμού και τα αναμενόμενα αποτελέσματα πρέπει να προσδιορίζονται με μεγάλη λεπτομέρεια.
- Το μέγεθος της μνήμης και οι απαιτήσεις χρόνου εκτέλεσης πρέπει να προσαρμόζονται εύκολα για να φιλοξενούν νέα συστήματα με αυξημένη ισχύ.
- Τα benchmark πρέπει να είναι πραγματικά διανεμητέα.

Για το λόγο αυτό φτιάχτηκαν οι πρώτες εκδόσεις των NPB και σήμερα διαθέτουμε και τρέχουμε στην παρούσα διπλωματική την έκδοση 3.4.

Η NASA προτρέπει τους ενδιαφερόμενους να αποστέλλουν τα αποτελέσματα από τα benchmarks πίσω σε αυτή μαζί με τα χαρακτηριστικά του συστήματος που τα έτρεξαν ώστε να τα βελτιώνουν και να ελέγχουν αν έχουν την επιθυμητή συμπεριφορά.

4.3 Κλάσεις των NPB

Τα Nas Parallel Benchmarks χωρίζονται σε κλάσεις, αναλόγως με το μέγεθος των προβλημάτων, τον αριθμό παραμέτρων τους και τον αριθμό των επαναλήψεων που κάνουν.

Benchmark code	Problem size	Memory (Mw)	Time (sec)	Rate (Mflop/s)
Embarrassingly parallel (EP)	2^{28}	1	151	147
Multigrid (MG)	256^3	57	54	154
Conjugate gradient (CG)	14000	10	22	70
3-D FFT PDE (FT)	$256^2 \times 128$	59	39	192
Integer sort (IS)	2^{23}	26	21	37.2
LU solver (LU)	64^3	30	344	189
Pentadiagonal solver (SP)	64^3	6	806	175
Block tridiagonal solver (BT)	64^3	24	923	192

Σχήμα 8: Npb ClassA Χαρακτηριστικά

Benchmark code	Problem size	Memory (Mw)	Time (sec)	Rate (Mflop/s)
Embarrassingly parallel (EP)	2^{30}	18	512	197
Multigrid (MG)	256^3	59	114	165
Conjugate gradient (CG)	75000	97	998	55
3-D FFT PDE (FT)	$512 \times 256 \times 256$	162	366	195
Integer sort (IS)	2^{25}	114	126	25
LU solver (LU)	102^3	122	1973	162
Pentadiagonal solver (SP)	102^3	22	2160	207
Block tridiagonal solver (BT)	102^3	96	3554	203

Σχήμα 9: Npb ClassB Χαρακτηριστικά

Αρχικά υπάρχουν δύο μικρές κλάσεις, η S που έχει μικρά προβλήματα για γρήγορα test, και η W που χρησιμοποιούνταν παλιά σε σταθμό εργασίας αλλά πλέον είναι μικρά τα προβλήματα. Έπειτα τα benchmarks έχουν τις κλάσεις A,B,C,D,E,F. Οι A,B,C θεωρούνται μικρές. Όσο αυξάνουμε από την μία στην άλλη, τα μεγέθη των

προβλημάτων μεγαλώνουν τέσσερις φορές από την προηγούμενη. Οι D,E,F περιέχουν μεγάλα προβλήματα. Από την μία στην άλλη το μέγεθος των προβλημάτων αυξάνει 16 φορές, καθιστώντας τις κλάσεις αυτές δύσκολο να χρησιμοποιηθούν σε μεσαίων δυνατοτήτων υπερυπολογιστές.

4.4 Τα NPB προβλήματα

Μετά από αξιολόγηση ενός αριθμού εφαρμογών μεγάλης κλίμακας CFD και υπολογιστικών αεροεπιστημών στους υπερυπολογιστές NAS της NASA Ames, επιλέχθηκε ένας αριθμός πυρήνων (kernels) για τα benchmarks. Αυτά συμπληρώθηκαν από ορισμένους άλλους πυρήνες που προορίζονται να δοκιμάσουν συγκεκριμένα χαρακτηριστικά παράλληλων μηχανών. Οι βασικοί πυρήνες είναι πέντε.

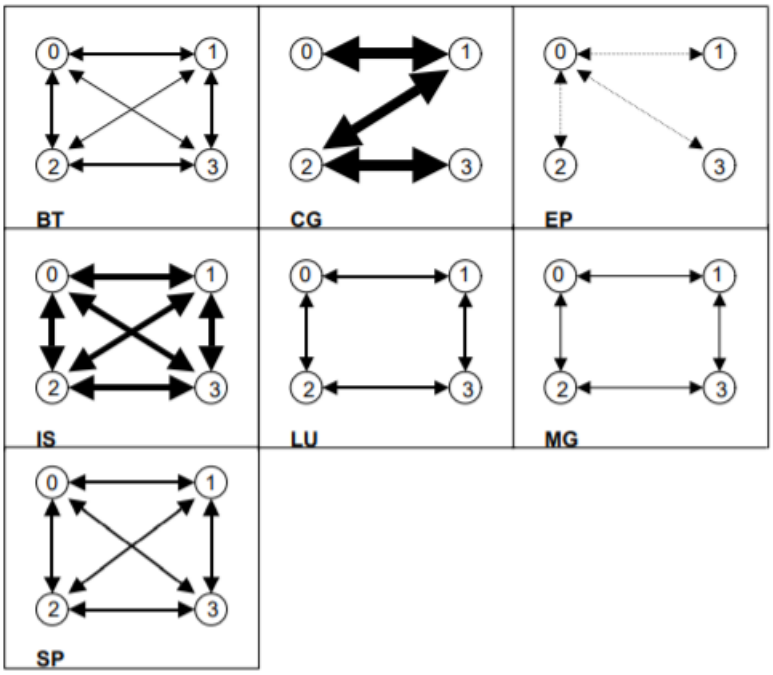
- **EP:** Ένας embarrassingly parallel πυρήνας. Παρέχει μια εκτίμηση των ανώτερων επιτεύξιμων ορίων για την απόδοση floating πράξεων, δηλαδή, την απόδοση χωρίς σημαντική επικοινωνία μεταξύ επεξεργαστών.
- **MG:** Ένας απλοποιημένος multigrid kernel. Απαιτεί πολύ δομημένη επικοινωνία μεγάλων αποστάσεων και δοκιμάζει επικοινωνία δεδομένων μικρής και μεγάλης απόστασης.
- **CG:** Χρησιμοποιείται μια μέθοδος conjugate gradient για τον υπολογισμό μιας προσέγγισης με τη μικρότερη ιδιοτιμή μιας μεγάλης, αραιής, συμμετρικής θετικής μήτρας. Αυτός ο πυρήνας είναι χαρακτηριστικός των υπολογισμών πλέγματος χωρίς δομή, καθώς ελέγχει ακανόνιστη επικοινωνία μεγάλων αποστάσεων, χρησιμοποιώντας πολλαπλασιασμό μη δομημένων matrix vector.
- **FT:** Μια τρισδιάστατη λύση μερικής διαφορικής εξίσωσης χρησιμοποιώντας FFTs. Αυτός ο πυρήνας εκτελεί την ουσία πολλών φασματικών κωδικών. Είναι μια καλή δοκιμασία της απόδοσης επικοινωνίας σε απόσταση.
- **IS:** Μια μεγάλη ταξινόμηση ακεραίων. Αυτός ο πυρήνας εκτελεί μια λειτουργία ταξινόμησης που είναι σημαντική στους κώδικες μεθόδου σωματιδίων. Ελέγχει τόσο την ακέραια ταχύτητα υπολογισμού όσο και την απόδοση επικοινωνίας.

Αυτοί είναι οι πέντε βασικοί πυρήνες των Nas Parallel Benchmarks. Οι BT,SP και LU είναι τρεις ακόμα βασικές ψεύδο-εφαρμογές.

Συγκεκριμένα, στην διπλωματική αυτή εξετάζονται και μελετώνται τα τρεξίματα και των 8 προβλημάτων: BT, CG, EP, FT, IS, LU, MG, SP. Το κάθε ένα έχει διαφορετικά χαρακτηριστικά και απαιτήσεις ως προς το σύστημα. Οι διαφορετικές απαιτήσεις αφορούν την υπολογιστική ισχύ, την συχνότητα αναφοράς στη μνήμη και την ανάγκη επικοινωνίας μέσω ανταλλαγής μηνυμάτων. Στο σχήμα 10 βλέπουμε τα μοτίβα επικοινωνίας των NPB που προέρχονται από μετρήσεις της κυκλοφορίας. Το πλάτος της γραμμής είναι ενδεικτικό του εύρους ζώνης επικοινωνίας [18].

Τα τρία προβλήματα BT, SP και LU λύνουν ένα συνθετικό σύστημα μη γραμμικών PDEs χρησιμοποιώντας τρεις διαφορετικούς αλγορίθμους.

- **BT:** Block tri-diagonal solver
- **SP:** Scalar penta-diagonal slover
- **LU:** Lower-Upper Gauss-Seidel solver



Σχήμα 10: Μοτίβα επικοινωνίας των NPB

Και τα τρία προβλήματα αυτά έχουν multi zone εκδοχές που έχουν σχεδιαστεί για να εκμεταλλεύονται πολλαπλά επίπεδα παραλληλισμού σε εφαρμογές και για να

ελέγχουν την αποτελεσματικότητα των πολυεπίπεδων και υβριδικών παραδειγμάτων και εργαλείων παραλληλισμού.

5. Μελέτη NPBs

Στο κεφάλαιο αυτό παρουσιάζεται η μελέτη που έγινε στα προβλήματα των Nas Parallel Benchmarks. Η μελέτη χωρίστηκε αρχικά στο τρέξιμο των προβλημάτων σειριακά, δηλαδή σε ένα μηχάνημα με έναν πυρήνα και στη συνέχεια παράλληλα με τη χρήση openmpi, σε πολλά μηχανήματα με πολλούς πυρήνες.

5.1 OpenMP NPBs

Τα openMP ή αλλιως OMP προβλήματα των NPB, είναι τα ίδια προβλήματα που περιγράφηκαν παραπάνω, φτιαγμένα για να τρέχουν με openMP, δηλαδή με μοντέλο κοινού χώρου διευθύνσεων. Συνεπώς, τρέχουν σε ένα μηχάνημα με δυνατότητα χρήσης όλων των πυρήνων του. Στη μελέτη αυτή, τα προβλήματα χρησιμοποίησαν έναν πυρήνα το καθένα, ώστε να παραχθεί ο χρόνος της σειριακής εκτέλεσης. Το αρχείο με το οποίο έτρεχε το καθένα είναι το ακόλουθο:

```
#!/bin/bash

## Give the Job a descriptive name
#PBS -N bt.A

## Output and error files
#PBS -o /home/users/ppeppas/NPB3.4/NPB3.4-OMP/logs/run.bt.A.out
#PBS -e /home/users/ppeppas/NPB3.4/NPB3.4-OMP/logs/run.bt.A.err

## Limit memory, runtime etc.
#PBS -l walltime=10:00:00

## How many nodes:processors_per_node should we get?
#PBS -l nodes=1:ppn=1

export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/home/users/ppeppas/mpi1/bin:$PATH
export LD_LIBRARY_PATH=/various/common_tools/gcc-4.5.2/lib64/

export OMP_NUM_THREADS=1
/home/users/ppeppas/NPB3.4/NPB3.4-OMP/bin/bt.A.x
```

Κώδικας 2: OMP mpirun example

Με τη χρήση της εντολής *export OMP_NUM_THREADS=1* δηλώνεται το 1 thread που θα χρησιμοιήσουν τα προβλήματα.

5.2 MPI NPBs

Με τη χρήση του μοντέλου ανταλλαγής μηνυμάτων, δηλαδή το MPI πρωτόκολλο, εκτελέστηκαν τα προβλήματα BT, CG, EP, FT, IS, LU, MG, SP σε τέσσερις κλάσεις μεγεθών, για διάφορους συνδυασμούς πυρήνων και μηχανημάτων.

```
#!/bin/bash

## Give the Job a descriptive name
#PBS -N bt.A.8x8

## Output and error files
#PBS -o /home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/run.bt.A.8x8.out
#PBS -e /home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/run.bt.A.8x8.err

## Limit memory, runtime etc.
#PBS -l walltime=10:00:00

## How many nodes:processors_per_node should we get?
#PBS -l nodes=8:ppn=8

export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/home/users/ppeppas/mpi1/bin:$PATH
export LD_LIBRARY_PATH=/various/common_tools/gcc-4.5.2/lib64/

RANKFILE=/home/users/ppeppas/NPB3.4/NPB3.4-MPI/rankfiles/rankfile8x8

mpirun -np 64 --rankfile $RANKFILE -v --report-bindings --timestamp-output --mca btl self,tcp /home/users/ppeppas/NPB3.4/NPB3.4-MPI/bin/bt.A.x
```

Κώδικας 3: MPI mpirun example

Στο κάθε αρχείο δηλώνεται όπως φαίνεται και στο παραπάνω σχήμα ένα Rankfile, γίνονται τα απαραίτητα export για να τρέξει το mpi, και χρησιμοποιείται η εντολή mpirun όπου καλεί το εκτελέσιμο.

Παράμετροι στην εντολή mpirun είναι το -np που είναι οι συνολικές διεργασίες που θα τρέξουν, το –rankfile που δηλώνεται ποιο rankfile να χρησιμοποιήσει η εντολή, το –mca btl self,tcp που είναι χρήσιμο για mpi εφαρμογές που είναι σχετικά μεγάλες, για ταχύτητα μεταξύ της ανταλλαγής μηνυμάτων, το --report-binding το οποίο στο αρχείο .err που παράγεται δηλώνει τις δεσμεύσεις των διεργασιών που δημιουργήθηκαν με τα cores που έδεσαν και το –timestam-output που στο αρχείο .err δείχνει την κάθε δέσμευση με την ημερομηνία και την ώρα που αυτή πραγματοποιήθηκε.

Στο σχήμα 11 φαίνεται το αρχείο .err ενός τρεξίματος. Στο αρχείο αυτό παρατηρείται αρχικά η ημερομηνία με την ώρα. Έπειτα φαίνεται σε ποια clones της ουράς έχει αντιστοιχισθεί η κάθε διεργασία. Αναφέρεται με rank η αρίθμηση της εργασίας. Έπειτα σημειώνεται αναλυτικά σε ποιο socket από τα δύο έγινε το δέσιμο και ακόμα πιο συγκεκριμένα σε ποιον πυρήνα. Στα δεξιά φαίνεται και σημειώνεται

με B ο συγκεκριμένος πυρήνας από τους 8 διαθέσιμους συνολικά του εκάστοτε κόμβου.

```
Wed Apr  1 12:13:34 2020<stderr>:[clone13:04492] MCW rank 0 bound to socket 0[core 0[hwt 0]]: [B/./.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone13:04492] MCW rank 1 bound to socket 0[core 1[hwt 0]]: [./B/.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone13:04492] MCW rank 2 bound to socket 0[core 2[hwt 0]]: [././B/.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone13:04492] MCW rank 3 bound to socket 0[core 3[hwt 0]]: [./././B][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone15:31727] MCW rank 10 bound to socket 0[core 2[hwt 0]]: [./.B/.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone14:01869] MCW rank 4 bound to socket 0[core 0[hwt 0]]: [B/./.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone15:31727] MCW rank 11 bound to socket 0[core 3[hwt 0]]: [././.B][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone14:01869] MCW rank 5 bound to socket 0[core 1[hwt 0]]: [./B/./.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone15:31727] MCW rank 8 bound to socket 0[core 0[hwt 0]]: [B/./.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone14:01869] MCW rank 6 bound to socket 0[core 2[hwt 0]]: [./.B/.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone15:31727] MCW rank 9 bound to socket 0[core 1[hwt 0]]: [./B/./.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone14:01869] MCW rank 7 bound to socket 0[core 3[hwt 0]]: [././.B][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone16:06984] MCW rank 14 bound to socket 0[core 2[hwt 0]]: [./.B/.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone16:06984] MCW rank 15 bound to socket 0[core 3[hwt 0]]: [././.B][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone16:06984] MCW rank 12 bound to socket 0[core 0[hwt 0]]: [B/./.][././.]
Wed Apr  1 12:13:34 2020<stderr>:[clone16:06984] MCW rank 13 bound to socket 0[core 1[hwt 0]]: [./B/./.][././.]
```

Σχήμα 11: Error file example

```
rank 0=+n0 slot=0:0
rank 1=+n0 slot=0:1
rank 2=+n0 slot=0:2
rank 3=+n0 slot=0:3
rank 4=+n1 slot=0:0
rank 5=+n1 slot=0:1
rank 6=+n1 slot=0:2
rank 7=+n1 slot=0:3
rank 8=+n2 slot=0:0
rank 9=+n2 slot=0:1
rank 10=+n2 slot=0:2
rank 11=+n2 slot=0:3
rank 12=+n3 slot=0:0
rank 13=+n3 slot=0:1
rank 14=+n3 slot=0:2
rank 15=+n3 slot=0:3
```

Κώδικας 4: Rankfile example

Στον παραπάνω κώδικα φαίνεται το παράδειγμα ενός Rankfile. Το rankfile λέει αναλυτικά με ποια θέση, δηλαδή με ποιον πυρήνα, θα δεθεί η κάθε διεργασία. Έτσι η εντολή *rank 5=+n1 slot=0:1* σημαίνει πως η διεργασία 5 θα τρέξει στο n1 δηλαδή στον δεύτερο διαθέσιμο κόμβο (η αρίθμηση ξεκινάει από το 0), στο slot 0, δηλαδή στο socket 0 και στον πυρήνα 1. Με αυτό τον τρόπο, δημιουργούνται rankfiles που μπορούν να χρησιμοποιούν μόνο το ένα socket από τα δύο του μηχανήματος και να πετυχαίνουμε την πολιτική spread.

Τα αποτελέσματα ενός προβλήματος, γράφονται στο αρχείο .out. Η μορφή ενός τέτοιου αρχείου φαίνεται στο σχήμα 12.

```
Wed Apr  1 12:13:35 2020<stdout>: Time step    1
Wed Apr  1 12:13:37 2020<stdout>: Time step   20
Wed Apr  1 12:13:39 2020<stdout>: Time step   40
Wed Apr  1 12:13:40 2020<stdout>: Time step   60
Wed Apr  1 12:13:42 2020<stdout>: Time step   80
Wed Apr  1 12:13:44 2020<stdout>: Time step  100
Wed Apr  1 12:13:45 2020<stdout>: Time step  120
Wed Apr  1 12:13:47 2020<stdout>: Time step  140
Wed Apr  1 12:13:49 2020<stdout>: Time step  160
Wed Apr  1 12:13:50 2020<stdout>: Time step  180
Wed Apr  1 12:13:52 2020<stdout>: Time step  200
Wed Apr  1 12:13:52 2020<stdout>: Verification being performed for class A
Wed Apr  1 12:13:52 2020<stdout>: accuracy setting for epsilon =  0.1000000000000E-07
Wed Apr  1 12:13:52 2020<stdout>: Comparison of RMS-norms of residual
Wed Apr  1 12:13:52 2020<stdout>:           1 0.1080634671464E+03 0.1080634671464E+03 0.8021787200786E-14
Wed Apr  1 12:13:52 2020<stdout>:           2 0.1131973090122E+02 0.1131973090122E+02 0.9415542762816E-15
Wed Apr  1 12:13:52 2020<stdout>:           3 0.2597435451158E+02 0.2597435451158E+02 0.3282665917769E-14
Wed Apr  1 12:13:52 2020<stdout>:           4 0.2366562254468E+02 0.2366562254468E+02 0.8106549417537E-14
Wed Apr  1 12:13:52 2020<stdout>:           5 0.2527896321175E+03 0.2527896321175E+03 0.1292970980304E-13
Wed Apr  1 12:13:52 2020<stdout>: Comparison of RMS-norms of solution error
Wed Apr  1 12:13:52 2020<stdout>:           1 0.4234841604053E+01 0.4234841604053E+01 0.1048656009767E-14
Wed Apr  1 12:13:52 2020<stdout>:           2 0.4439028249700E+00 0.4439028249700E+00 0.5627361811846E-14
Wed Apr  1 12:13:52 2020<stdout>:           3 0.9669248013635E+00 0.9669248013635E+00 0.8037399766163E-15
Wed Apr  1 12:13:52 2020<stdout>:           4 0.8830206303977E+00 0.8830206303977E+00 0.1508761611776E-14
Wed Apr  1 12:13:52 2020<stdout>:           5 0.9737990177083E+01 0.9737990177083E+01 0.4925208774258E-14
Wed Apr  1 12:13:52 2020<stdout>: Verification Successful
Wed Apr  1 12:13:52 2020<stdout>:
Wed Apr  1 12:13:52 2020<stdout>:
Wed Apr  1 12:13:52 2020<stdout>: BT Benchmark Completed.
Wed Apr  1 12:13:52 2020<stdout>: Class           =                        A
Wed Apr  1 12:13:52 2020<stdout>: Size            =             64x  64x  64
Wed Apr  1 12:13:52 2020<stdout>: Iterations      =                      200
Wed Apr  1 12:13:52 2020<stdout>: Time in seconds =                    17.00
Wed Apr  1 12:13:52 2020<stdout>: Total processes =                       16
Wed Apr  1 12:13:52 2020<stdout>: Active procs    =                       16
Wed Apr  1 12:13:52 2020<stdout>: Mop/s total     =                  9896.75
Wed Apr  1 12:13:52 2020<stdout>: Mop/s/process   =                   618.55
Wed Apr  1 12:13:52 2020<stdout>: Operation type  =           floating point
Wed Apr  1 12:13:52 2020<stdout>: Verification    =               SUCCESSFUL
Wed Apr  1 12:13:52 2020<stdout>: Version         =                      3.4
Wed Apr  1 12:13:52 2020<stdout>: Compile date    =              01 Apr 2020
Wed Apr  1 12:13:52 2020<stdout>:
Wed Apr  1 12:13:52 2020<stdout>: Compile options:
Wed Apr  1 12:13:52 2020<stdout>:    MPIFC        = mpif90
Wed Apr  1 12:13:52 2020<stdout>:    FLINK        = $(MPIFC)
Wed Apr  1 12:13:52 2020<stdout>:    FMPI_LIB     = (none)
Wed Apr  1 12:13:52 2020<stdout>:    FMPI_INC     = (none)
Wed Apr  1 12:13:52 2020<stdout>:    FFLAGS       = -O3
```

Σχήμα 12: Output file example

Στο αρχείο αυτό φαίνονται πληροφορίες για το πως τρέχει το Nas Parallel Problem όπως και τα αποτελέσματα του τρεξίματος του. Το Verification είναι SUCCESSFUL συνεπώς το πρόβλημα έτρεξε και ολοκληρώθηκε ορθώς και με σωστά αποτελέσματα. Σημειώνεται ποιο είναι το πρόβλημα και ποια η κλάση του όπως και βασικά του χαρακτηριστικά. Η μελέτη της εργασίας αυτής αφορά τον χρόνο Time in seconds, που δείχνει τη συνολική διάρκεια εκτέλεσης του προβλήματος.

Στα αριστερά του αρχείου φαίνονται όπως και στο .err η ημερομηνία και η ώρα που το πρόβλημα αυτό έτρεξε στο cluster.

```
qsub -q parlab -l nodes=clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.A.1x8
qsub -q parlab -l nodes=clone15:ppn=8+clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.A.2x4
qsub -q parlab -l nodes=clone13:ppn=8+clone14:ppn=8+clone15:ppn=8+clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.A.4x8
qsub -q parlab -l nodes=clone1:ppn=8+clone2:ppn=8+clone4:ppn=8+clone5:ppn=8+clone13:ppn=8+clone14:ppn=8+clone15:ppn=8+clone16:ppn=8
 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.A.8x4
qsub -q parlab -l nodes=clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.B.1x8
qsub -q parlab -l nodes=clone15:ppn=8+clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.B.2x4
qsub -q parlab -l nodes=clone13:ppn=8+clone14:ppn=8+clone15:ppn=8+clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.B.4x8
qsub -q parlab -l nodes=clone1:ppn=8+clone2:ppn=8+clone4:ppn=8+clone5:ppn=8+clone13:ppn=8+clone14:ppn=8+clone15:ppn=8+clone16:ppn=8
 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.B.8x4
qsub -q parlab -l nodes=clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.C.1x8
qsub -q parlab -l nodes=clone15:ppn=8+clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.C.2x4
qsub -q parlab -l nodes=clone13:ppn=8+clone14:ppn=8+clone15:ppn=8+clone16:ppn=8 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.C.4x8
qsub -q parlab -l nodes=clone1:ppn=8+clone2:ppn=8+clone4:ppn=8+clone5:ppn=8+clone13:ppn=8+clone14:ppn=8+clone15:ppn=8+clone16:ppn=8
 /home/users/ppeppas/NPB3.4/NPB3.4-MPI/mpirun/mpirun.bt.C.8x4
```

Κώδικας 5: Qsub file example

Για το τρέξιμο των αρχείων mpirun του σχήματος 3 χρησιμοποιείται η εντολή qsub όπως φαίνεται στο παραπάνω σχήμα. Στην εντολή αυτή δίνονται σαν παράμετροι η ουρά του συστήματος της σχολής που θα χρησιμοποιηθεί, τα clones που θα δεσμευθούν και το path για το εκτελέσιμο mpirun αρχείο.

5.3 Αποτελέσματα NPBs

Στο κεφάλαιο αυτό θα παρουσιαστούν αναλυτικά τα αποτελέσματα μέσω γραφικών παραστάσεων των Nas Parallel Benchmark προβλημάτων.

Τα προβλήματα εκτελέστηκαν στις κλάσεις A,B,C,D και για αριθμό πυρήνων τα OMP 1, ενώ τα MPI 8,16,32 και 64.

Στις γραφικές παραστάσεις παρουσιάζονται για κάθε πρόβλημα και για κάθε κλάση 3 γραφήματα. Το πρώτο είναι ο χρόνος εκτέλεσης σε συνάρτηση με τον αριθμό πυρήνων, το δεύτερο είναι το speedup που επιτυγχάνεται σε σχέση με την σειριακή εκτέλεση σε συνάρτηση με τον αριθμό πυρήνων και το τρίτο είναι το efficiency που επιτυγχάνεται ομοίως σε συνάρτηση με τον αριθμό των πυρήνων.

Αρχικά παρατίθενται οι πίνακες των αποτελεσμάτων των NPBs για τις τρεις κλάσεις και στη συνέχεια οι γραφικές παραστάσεις.

A										
Policy	Processes	Topology	BT	CG	EP	FT	IS	LU	MG	SP
Compact	8	1x8	52,00	1,35	4,28	4,36	0,62	23,93	1,92	92,91
Spread	8	2x4	52,00	1,26	4,28	4,68	1,13	21,16	1,48	92,60
Compact	16	2x8	18,45	0,73	2,16	3,97	0,95	11,77	1,14	25,65
Spread	16	4x4	17,00	0,72	2,16	3,36	0,83	11,76	1,49	25,32
Compact	32	4x8	27,70	1,96	1,35	3,73	1,95	20,64	2,05	43,54
Spread	32	8x4	31,31	0,77	1,13	1,89	0,93	6,64	1,13	67,00
Compact	64	8x8	15,76	4,66	0,80	8,13	1,99	29,49	1,28	28,31
Spread	64	16x4	31,31	0,74	0,56	2,57	0,29	6,61	0,50	64,72

Σχήμα 13: CLASS A

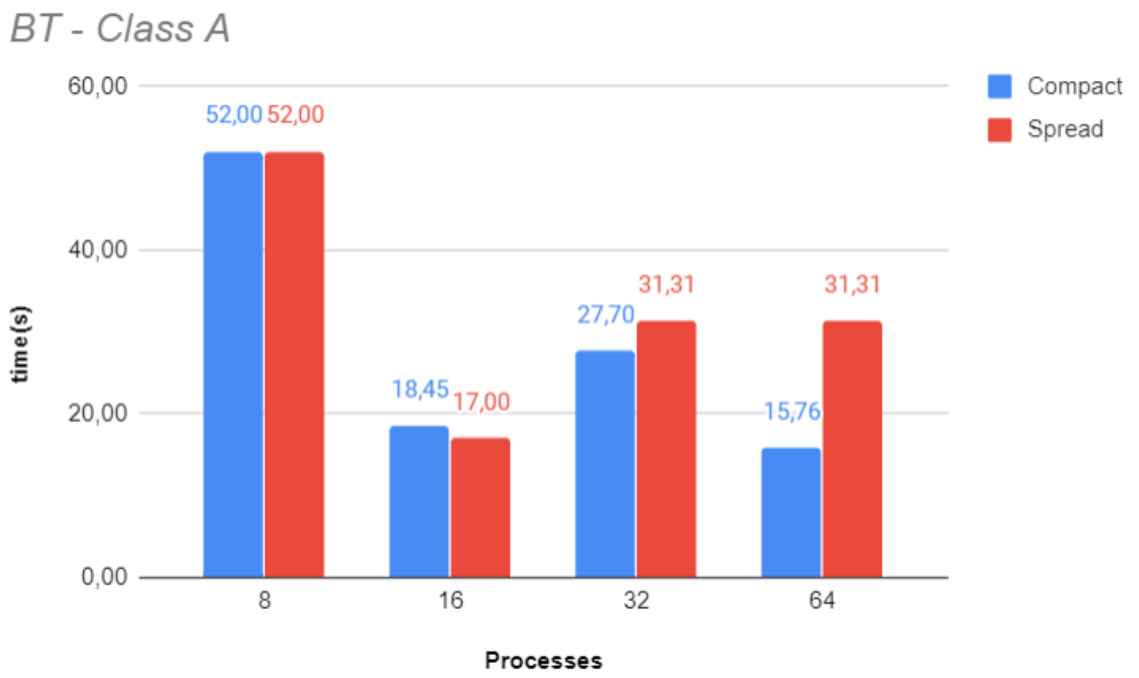
B										
Policy	Processes	Topology	BT	CG	EP	FT	IS	LU	MG	SP
Compact	8	1x8	226,90	59,02	17,13	48,40	2,65	105,85	8,99	452,77
Spread	8	2x4	226,58	53,97	17,17	54,01	4,30	85,71	6,95	452,51
Compact	16	2x8	72,11	49,52	8,54	45,68	3,76	55,88	5,45	143,07
Spread	16	4x4	88,59	48,06	8,61	42,54	4,94	51,86	7,73	153,24
Compact	32	4x8	87,35	98,26	5,58	56,50	4,45	55,10	10,51	144,29
Spread	32	8x4	82,76	34,36	4,27	51,33	2,30	31,40	5,37	140,07
Compact	64	8x8	68,94	151,26	2,68	43,23	10,42	55,73	5,51	126,08
Spread	64	16x4	62,06	53,03	2,17	16,17	2,73	24,43	1,82	112,22

Σχήμα 14: CLASS B

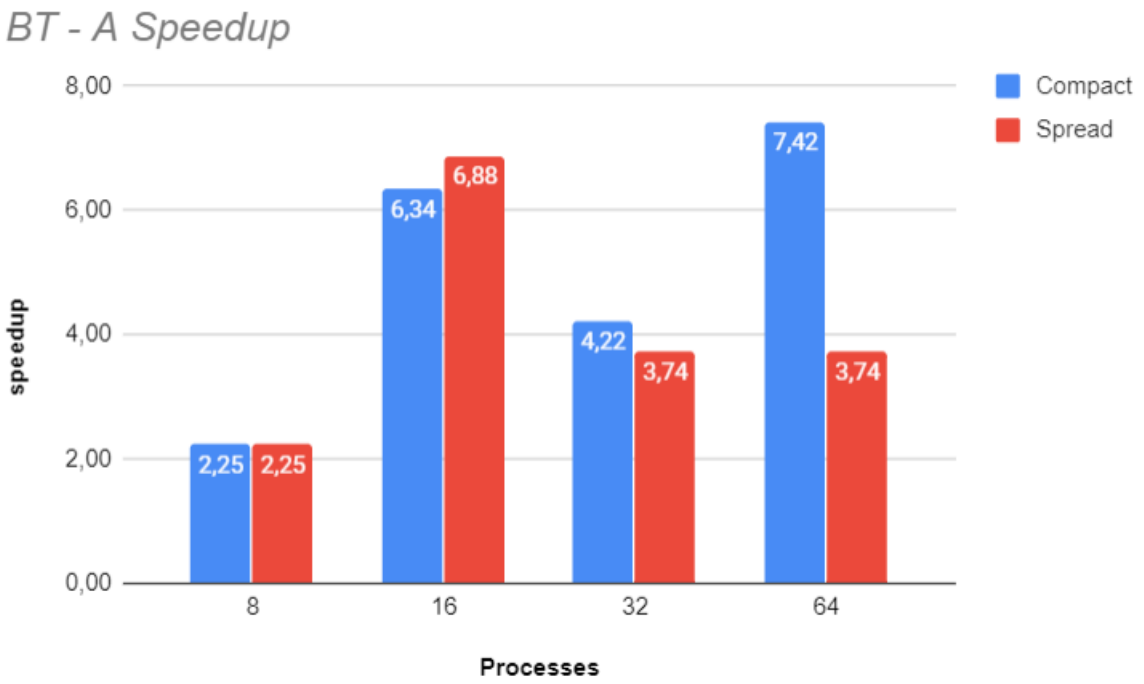
C										
Policy	Processes	Topology	BT	CG	EP	FT	IS	LU	MG	SP
Compact	8	1x8	950,19	154,15	68,02	202,38	10,86	426,84	87,32	1.843,35
Spread	8	2x4	950,57	134,30	69,50	231,24	17,39	344,30	64,34	1.844,10
Compact	16	2x8	306,14	115,17	34,24	183,12	15,64	219,82	48,66	684,40
Spread	16	4x4	257,65	99,19	34,45	152,18	16,13	186,48	41,73	470,60
Compact	32	4x8	227,68	145,94	18,92	143,91	21,29	159,97	35,17	442,37
Spread	32	8x4	191,26	93,97	17,08	107,45	18,65	106,34	25,10	329,86
Compact	64	8x8	154,79	199,14	10,59	148,81	17,54	125,48	24,13	260,09
Spread	64	16x4	110,15	153,30	8,66	126,76	4,99	76,64	19,51	187,72

Σχήμα 15: CLASS C

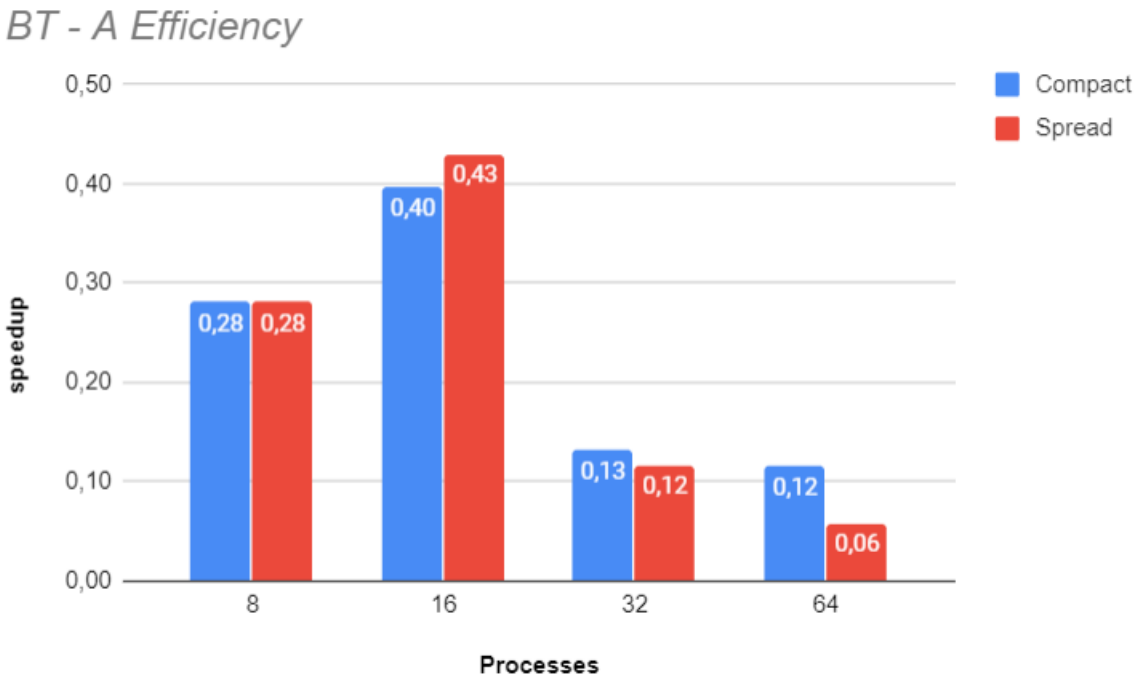
5.3.1 Class A



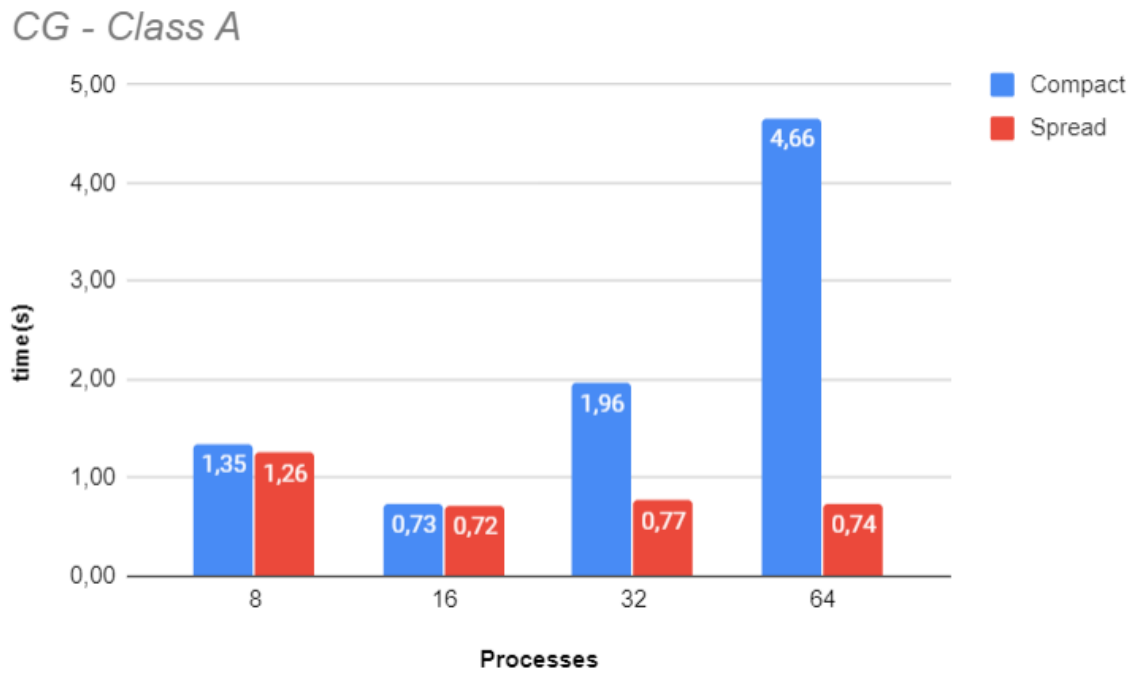
Γραφικές 1: BT - CLASS A



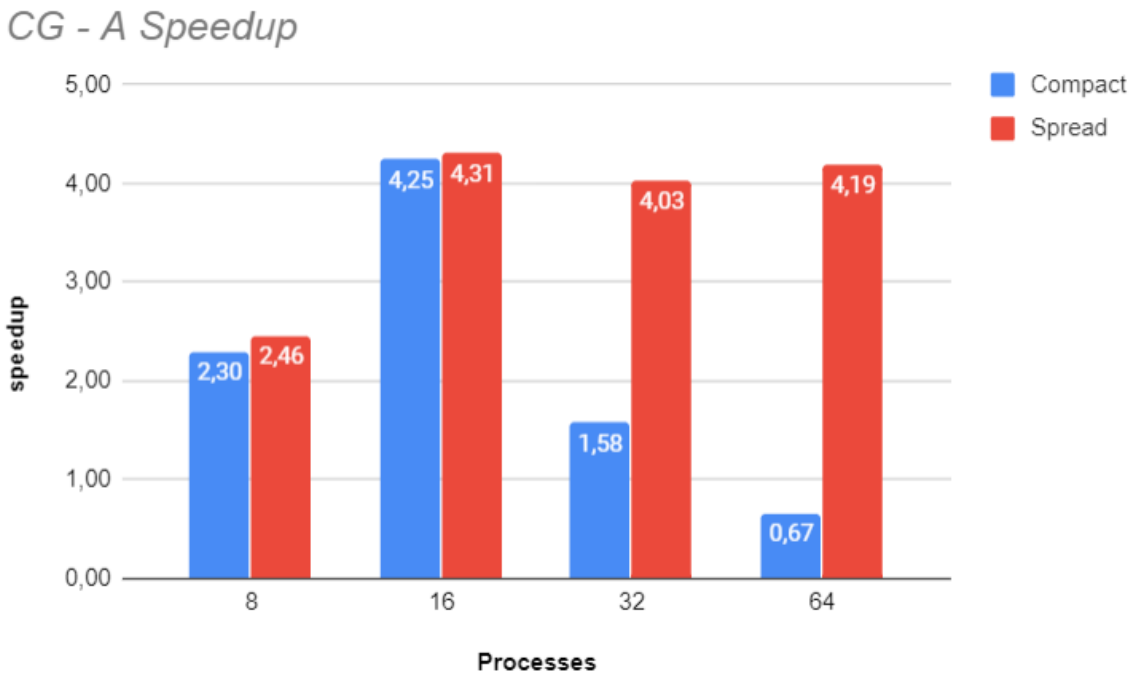
Γραφικές 2: BT - A Speedup



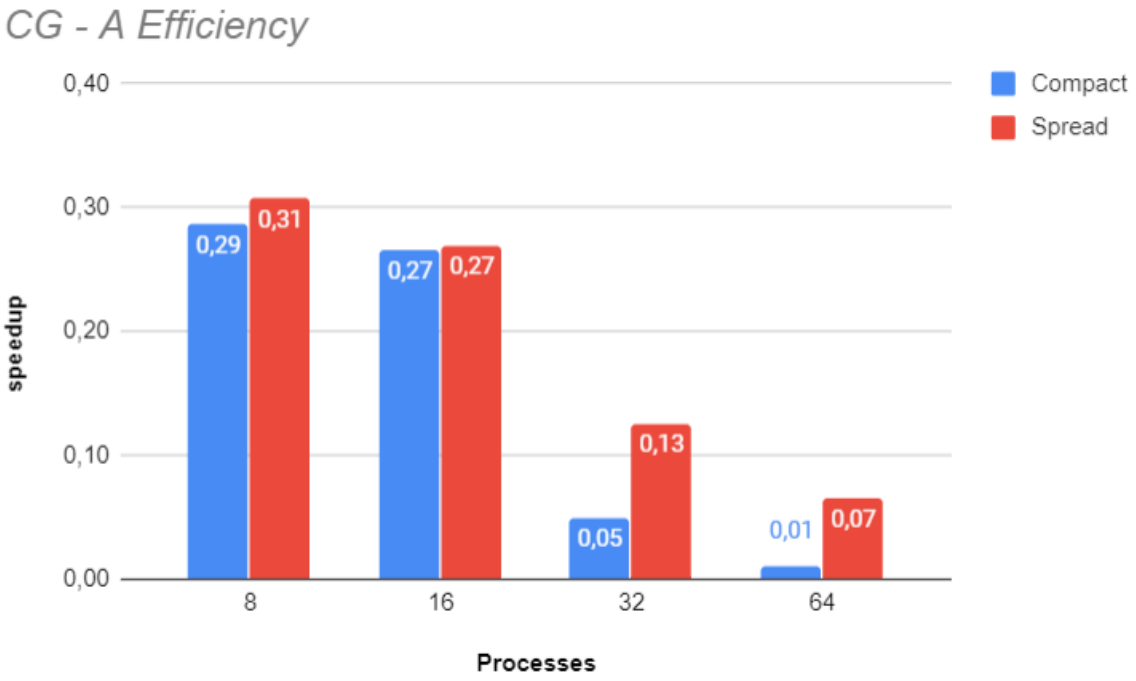
Γραφικές 3: BT - A Efficiency



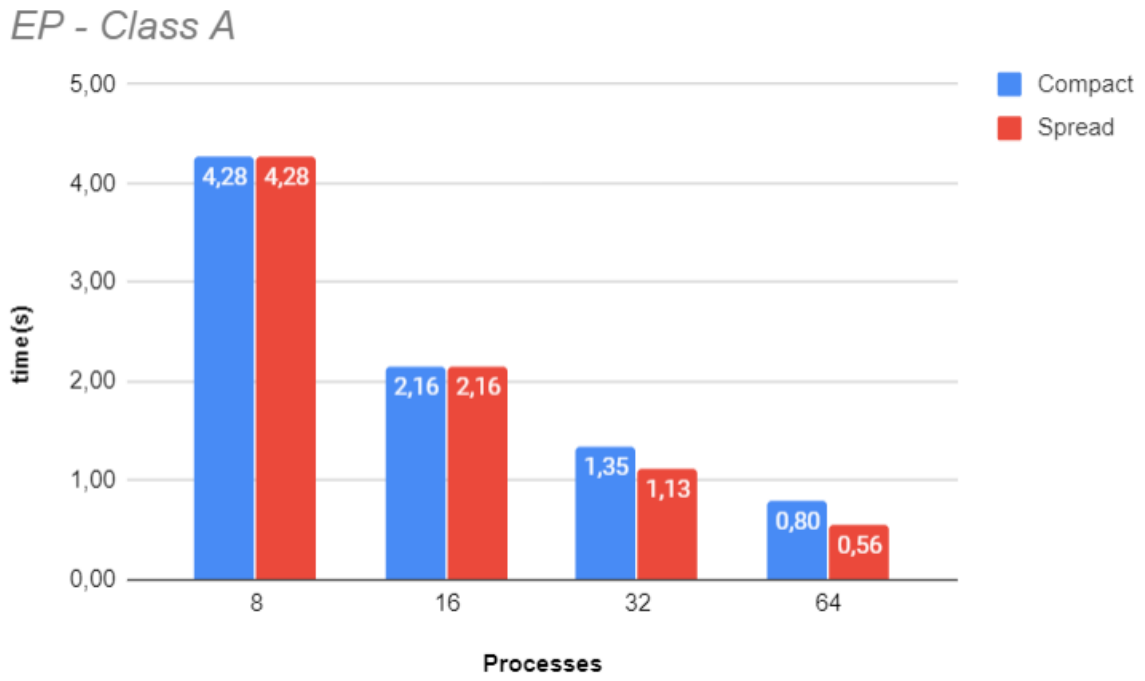
Γραφικές 4: CG - CLASS A



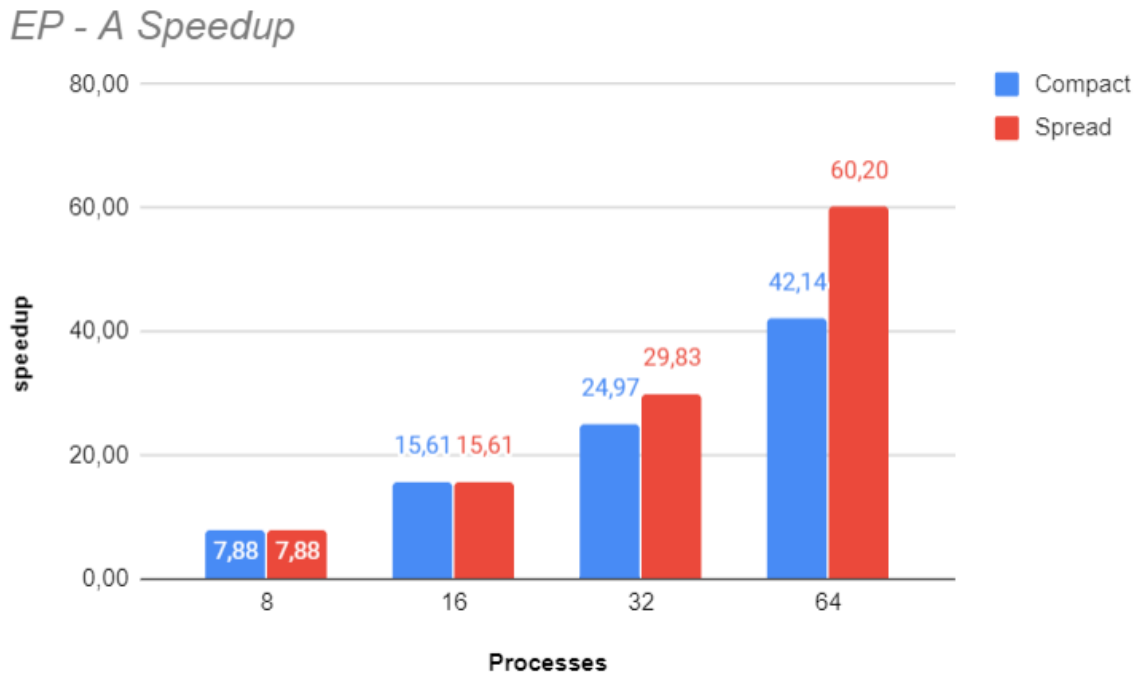
Γραφικές 5: CG - A Speedup



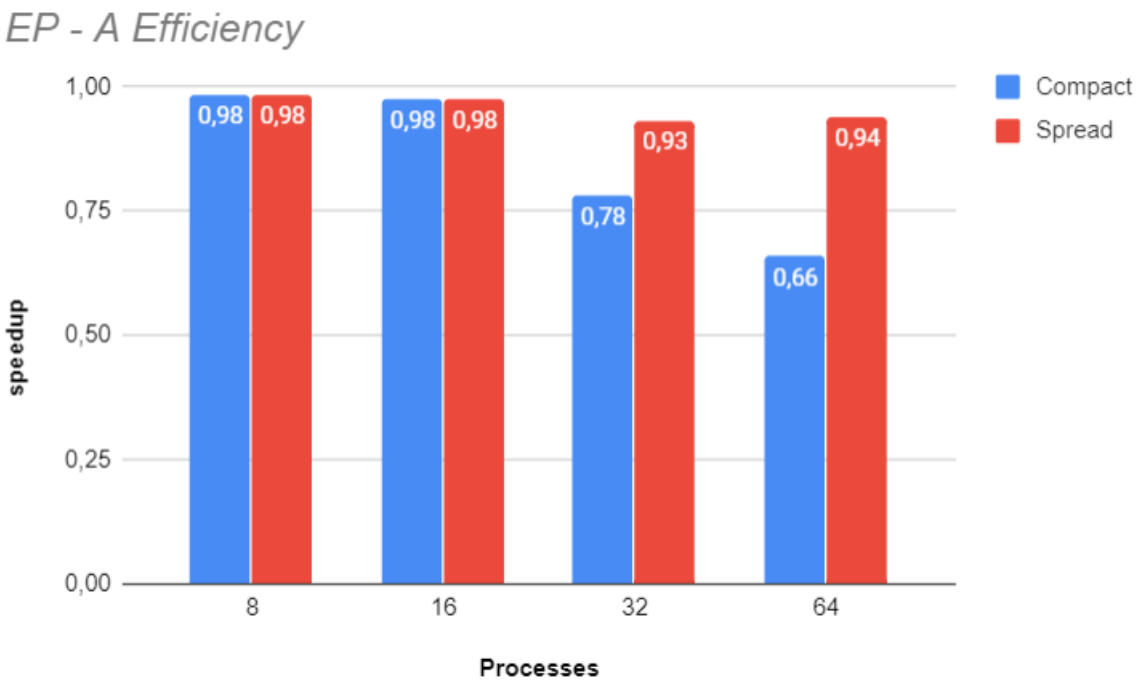
Γραφικές 6: CG - A Efficiency



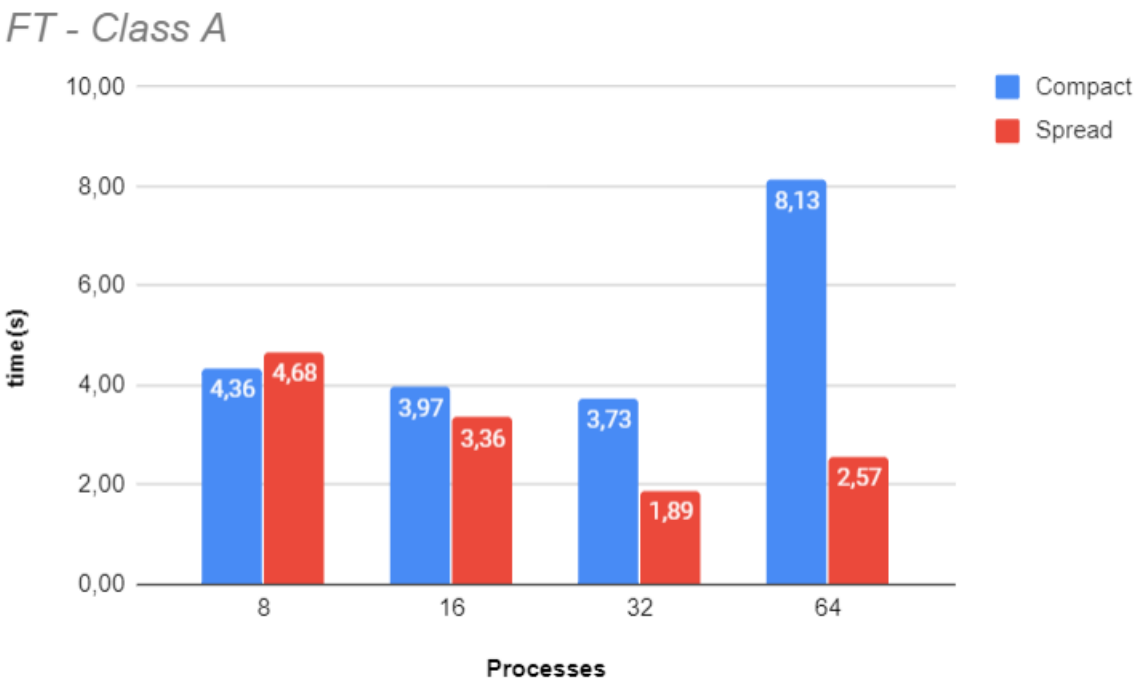
Γραφικές 7: EP - CLASS A



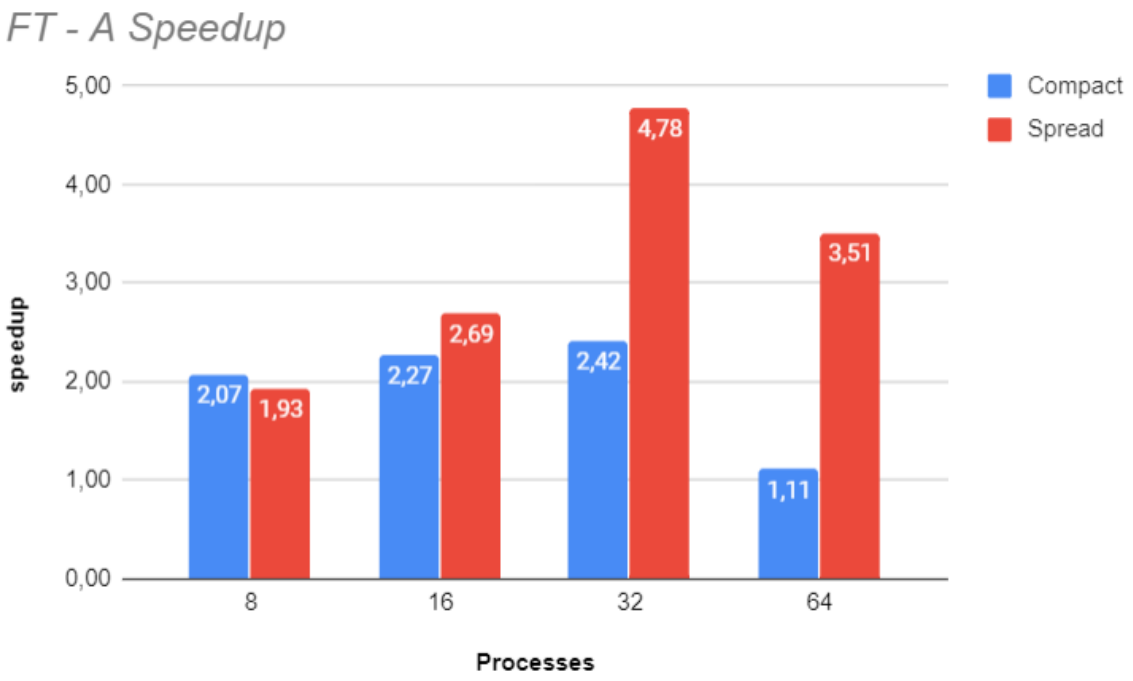
Γραφικές 8: EP - A Speedup



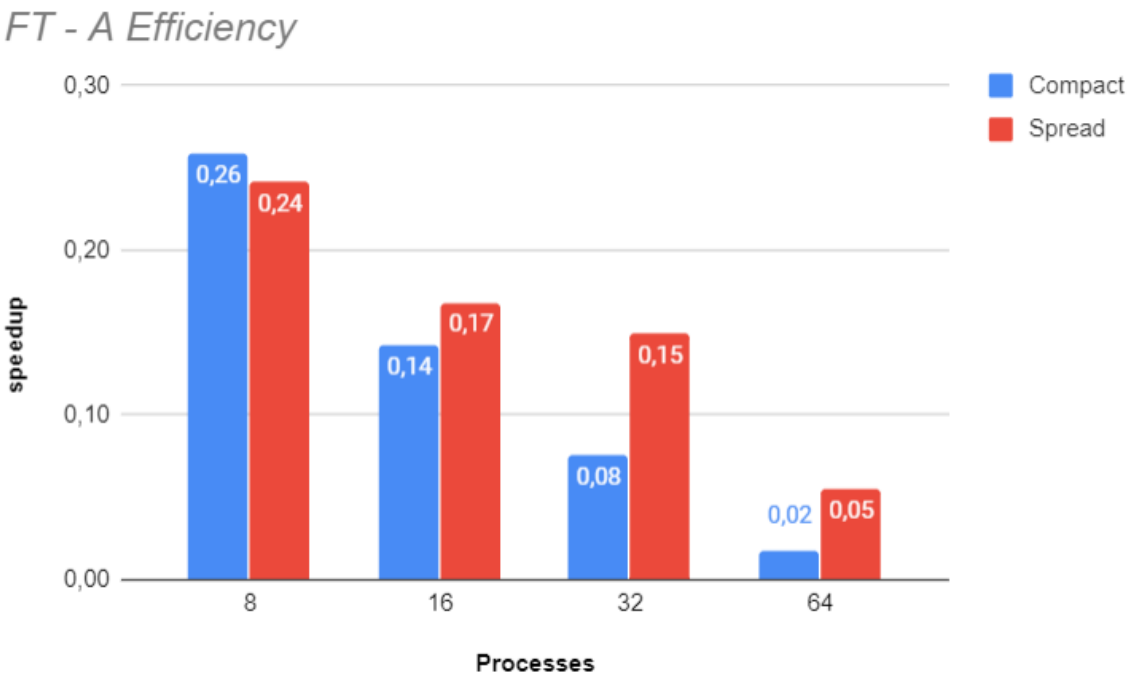
Γραφικές 9: EP - A Efficiency



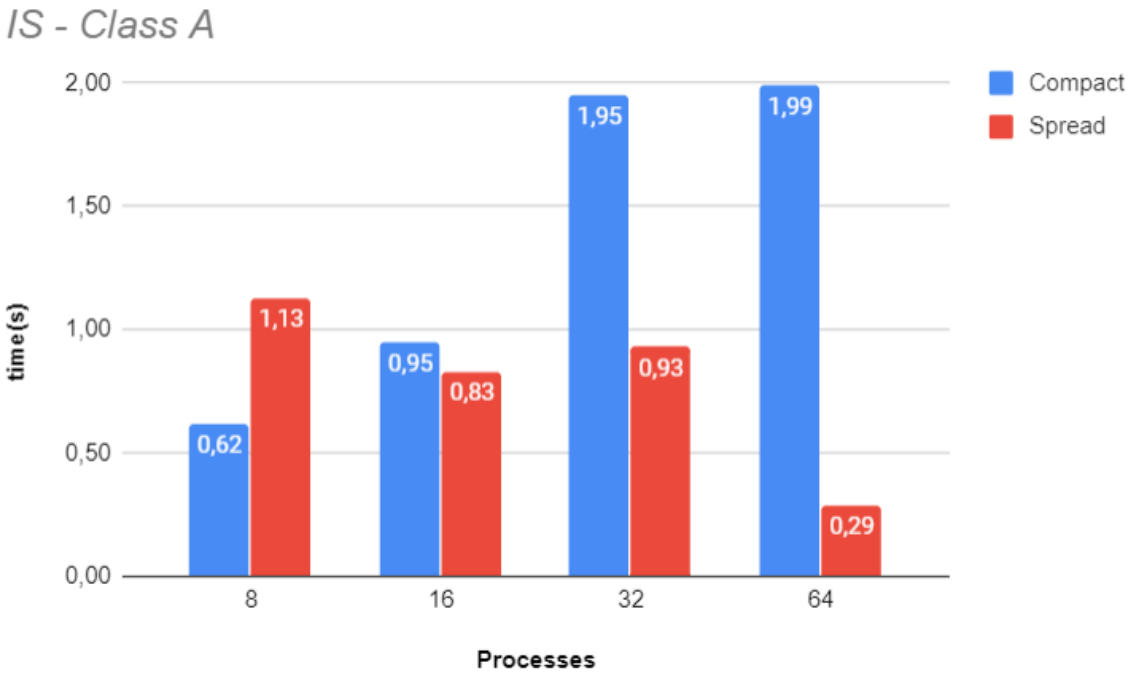
Γραφικές 10: FT - CLASS A



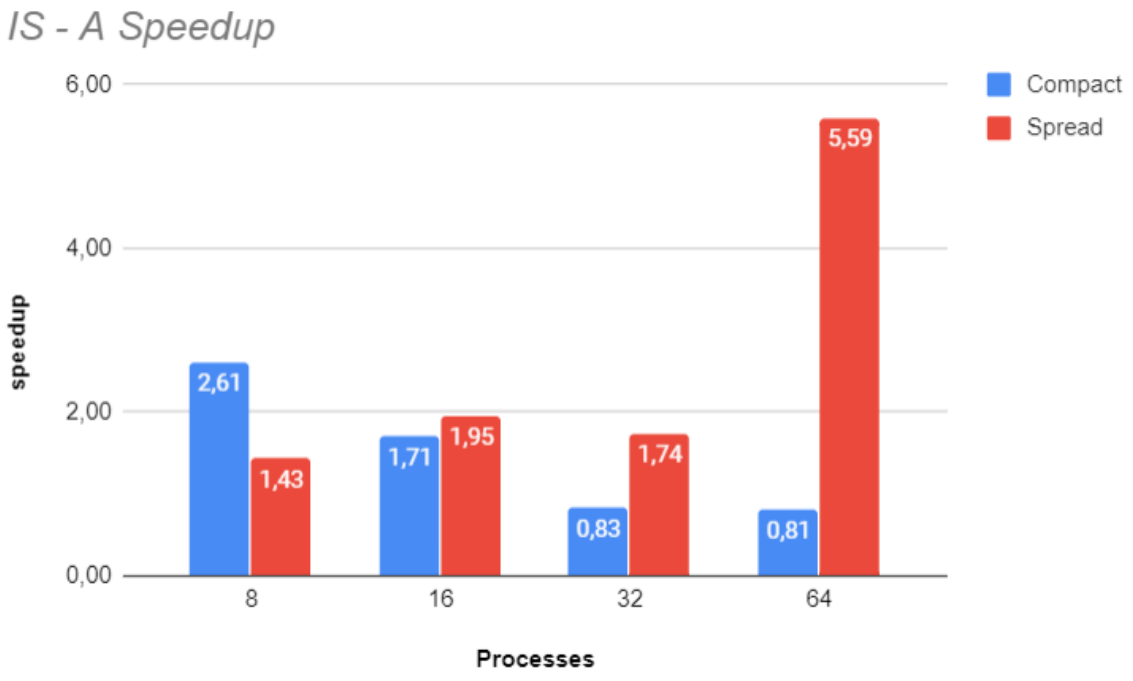
Γραφικές 11: FT - A Speedup



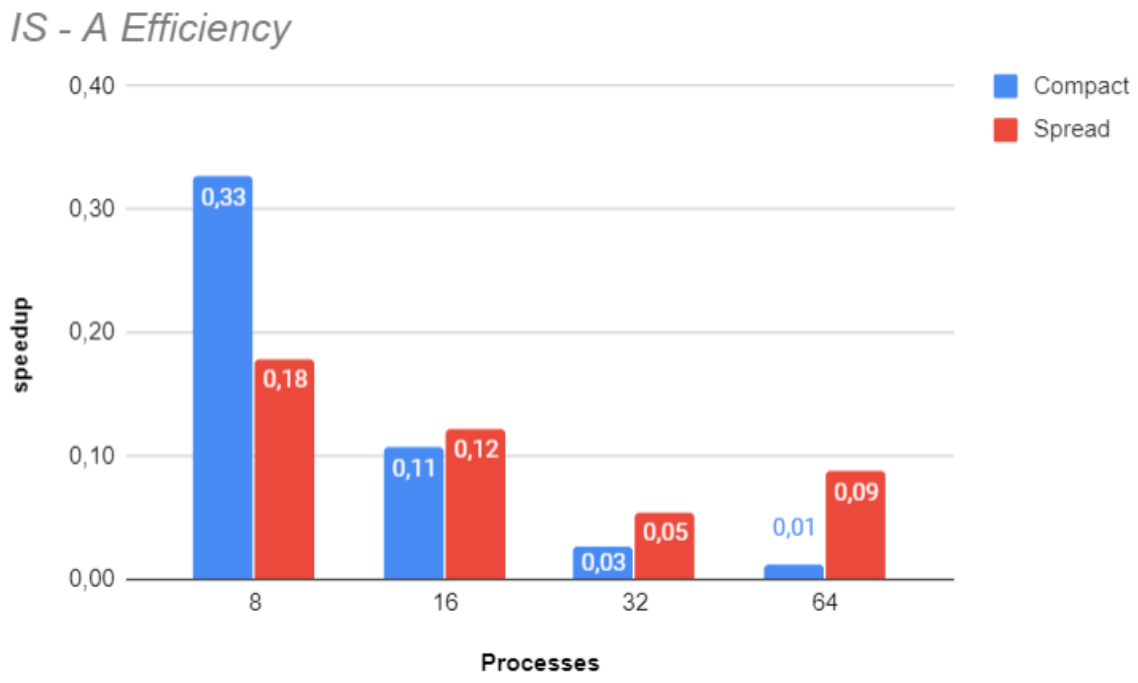
Γραφικές 12: FT - A Efficiency



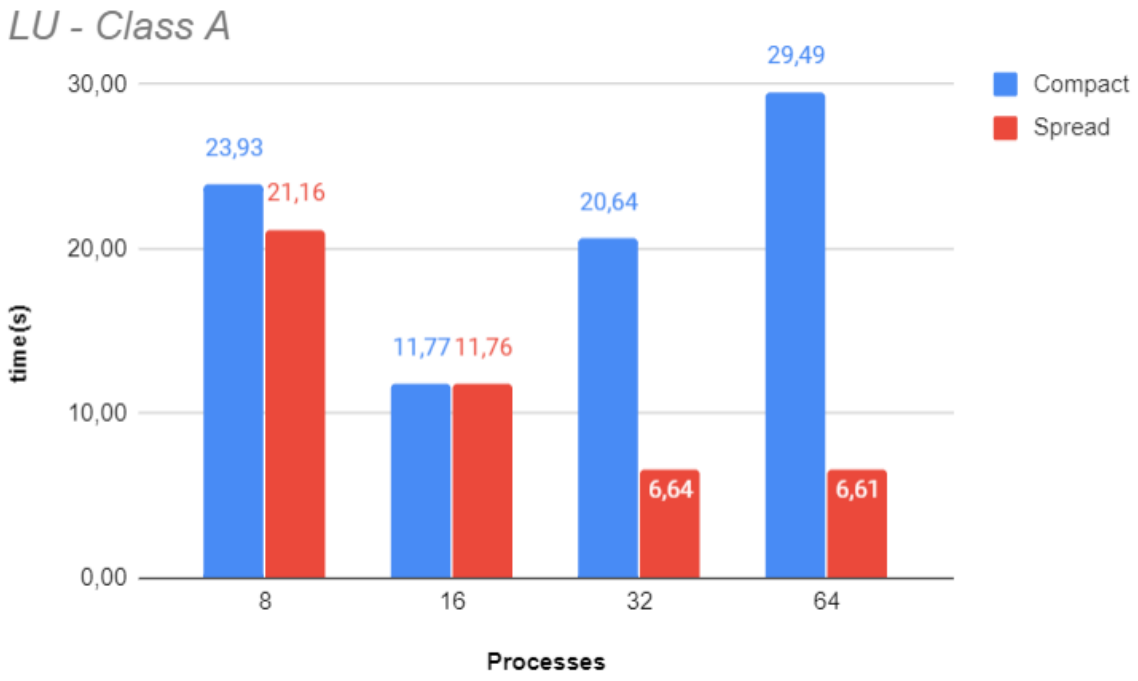
Γραφικές 13: IS - CLASS A



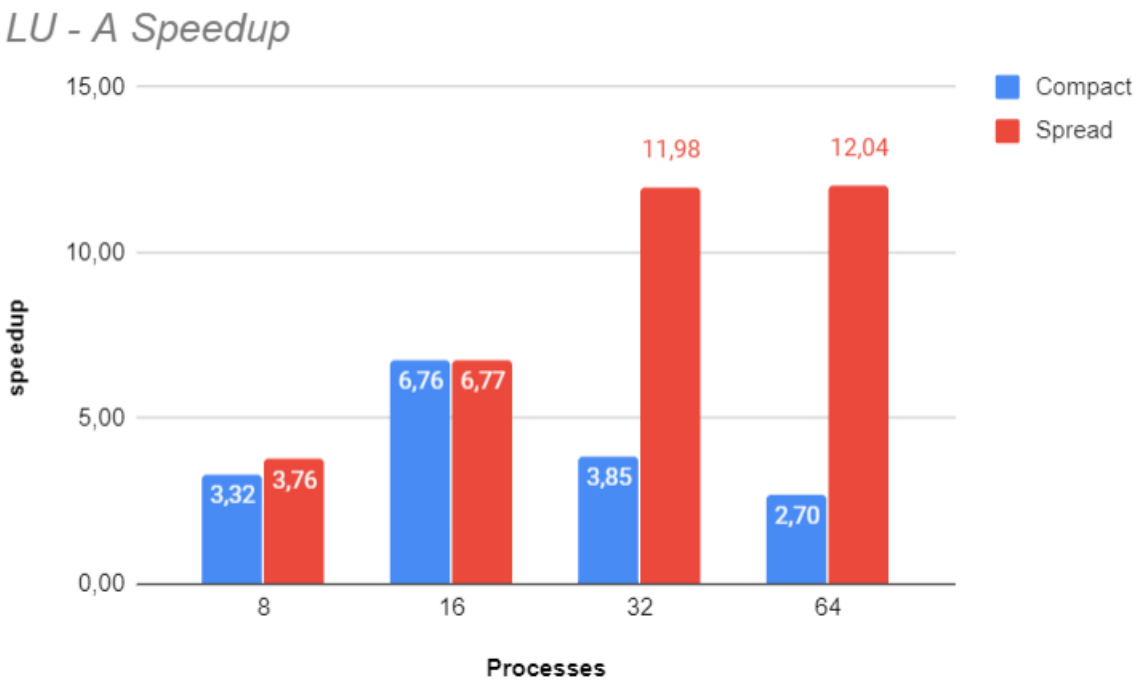
Γραφικές 14: IS - A Speedup



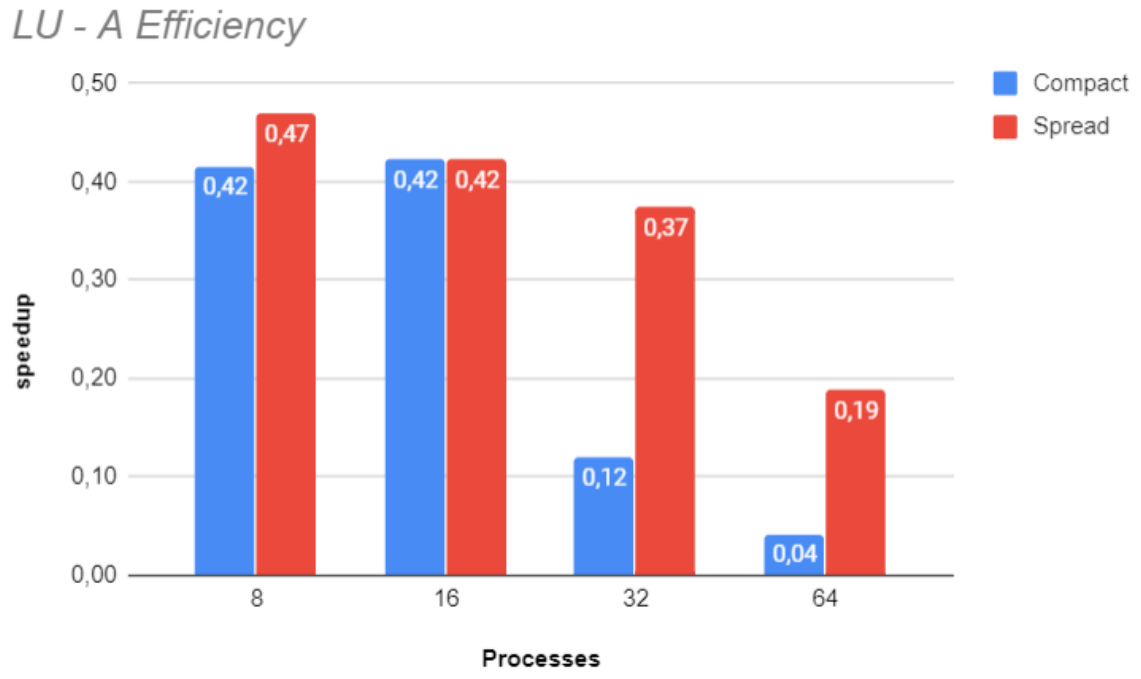
Γραφικές 15: IS - A Efficiency



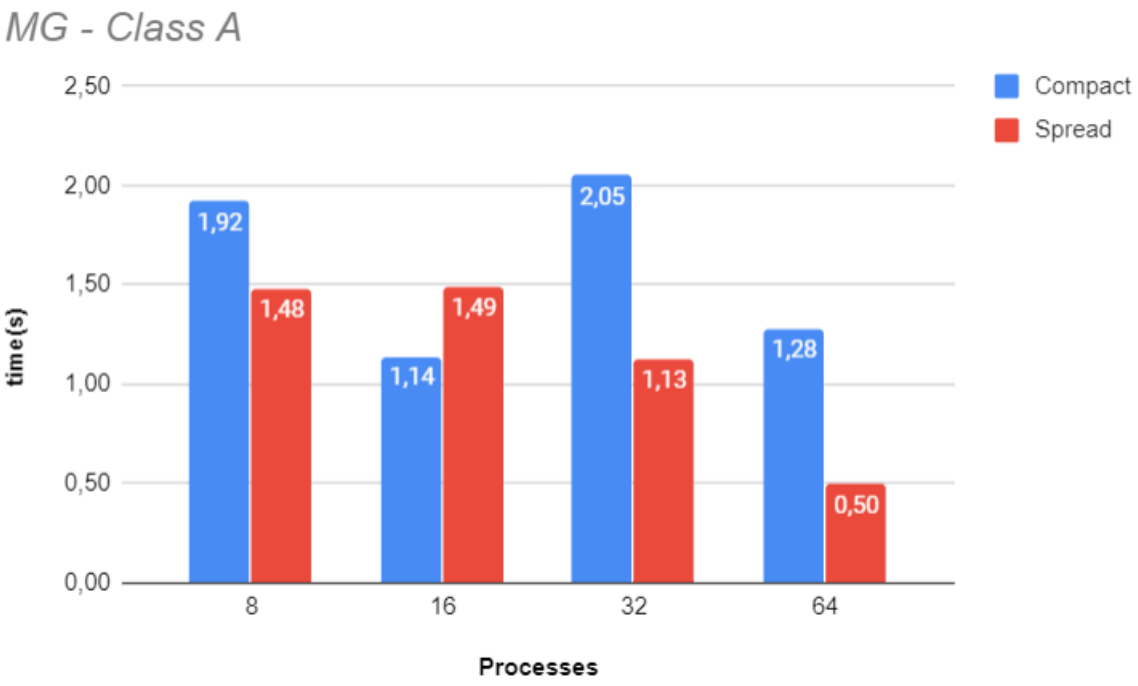
Γραφικές 16: LU - CLASS A



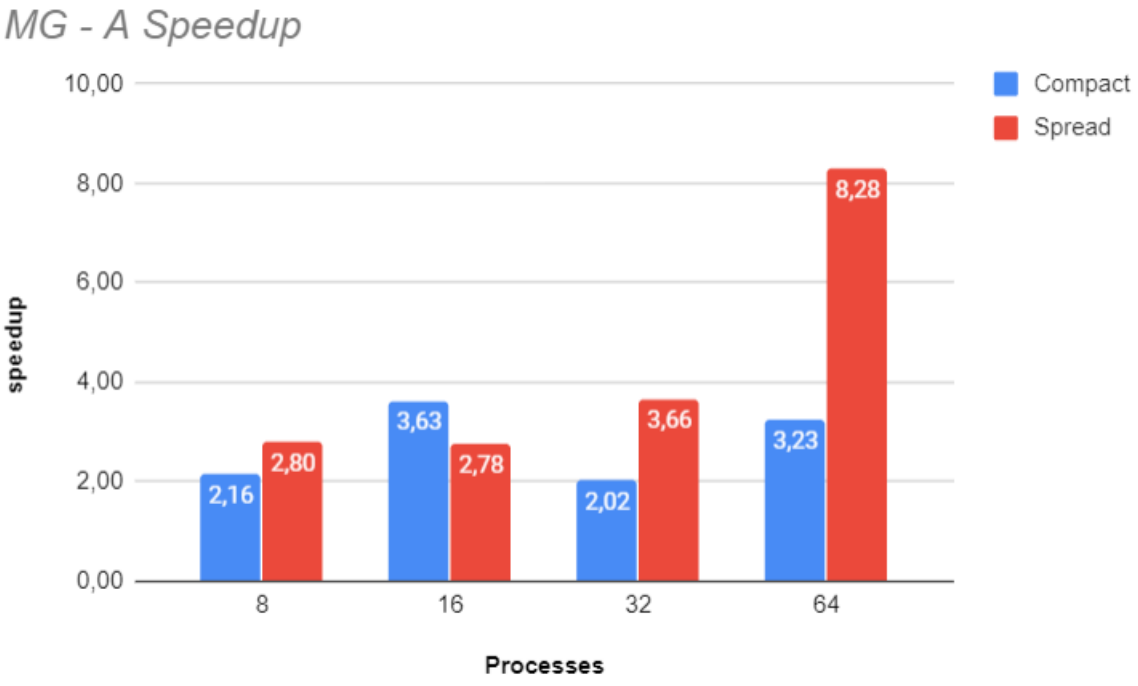
Γραφικές 17: LU – A Speedup



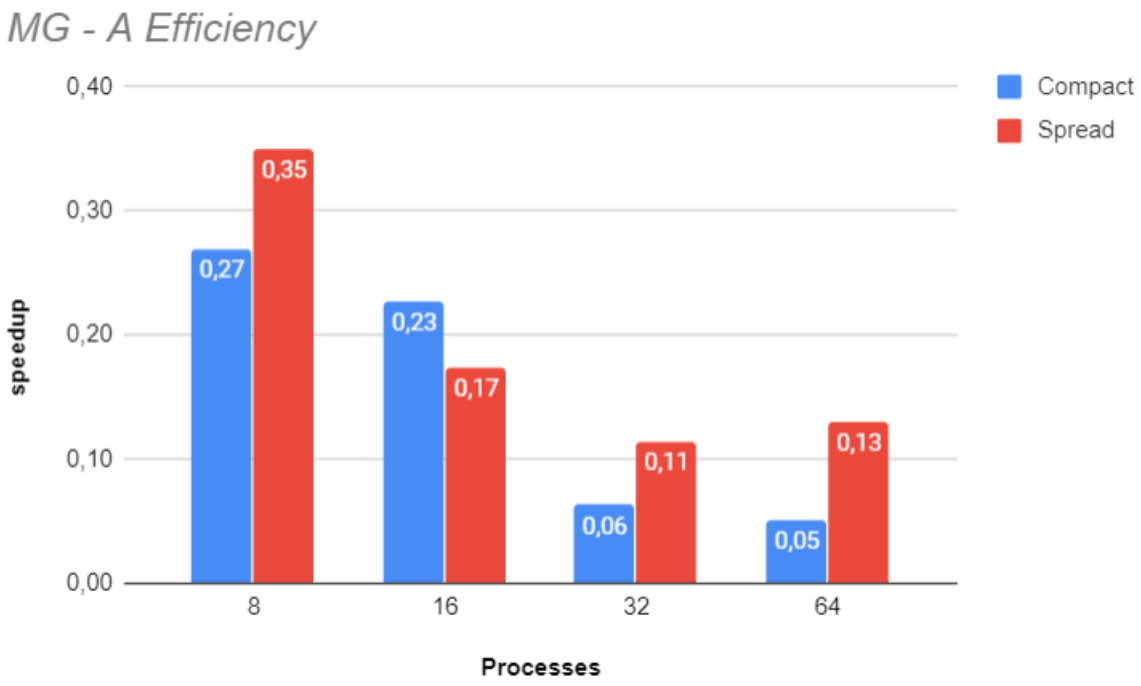
Γραφικές 18: LU - A Efficiency



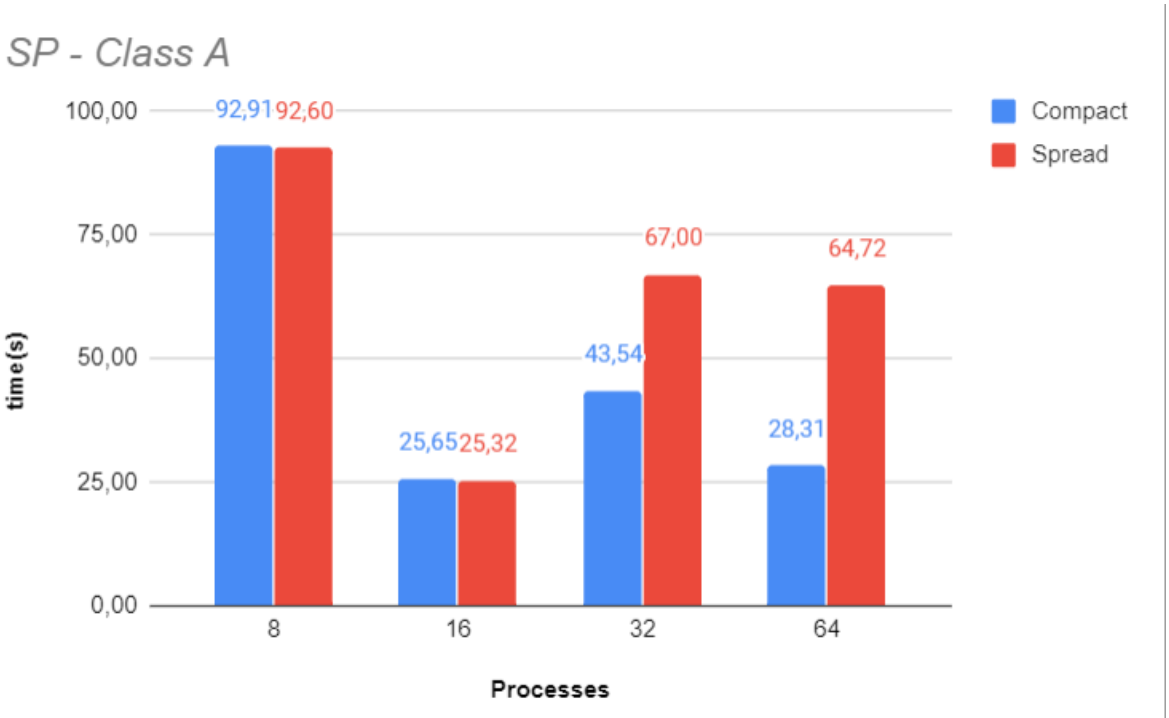
Γραφικές 19: MG - CLASS A



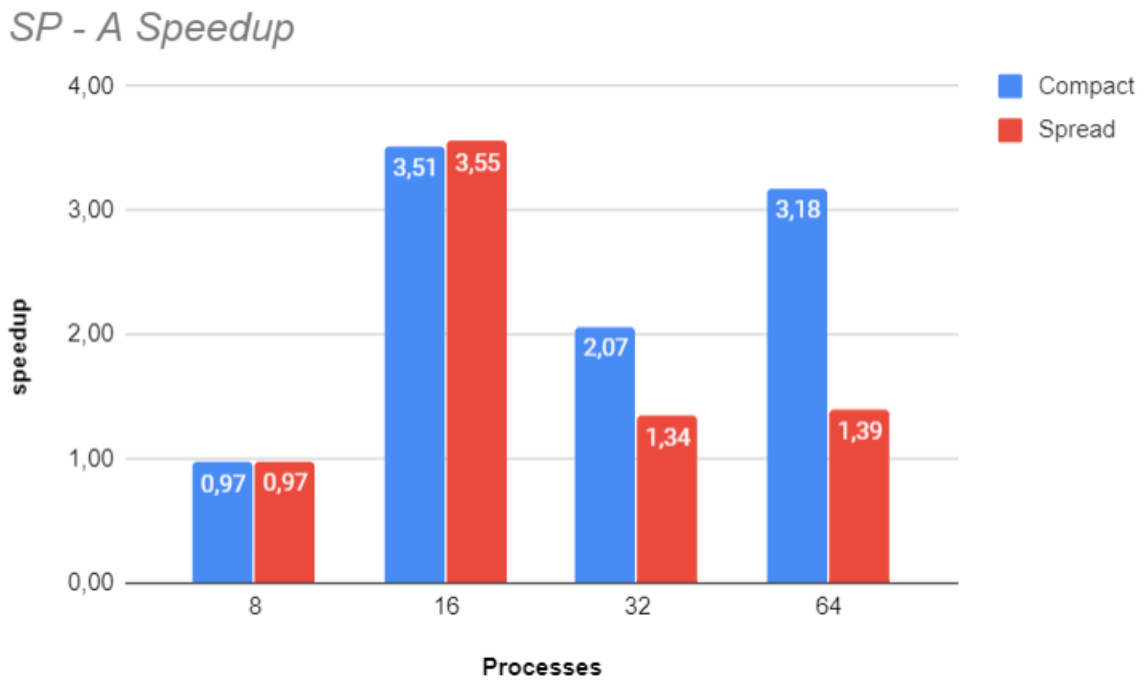
Γραφικές 20: MG - A Speedup



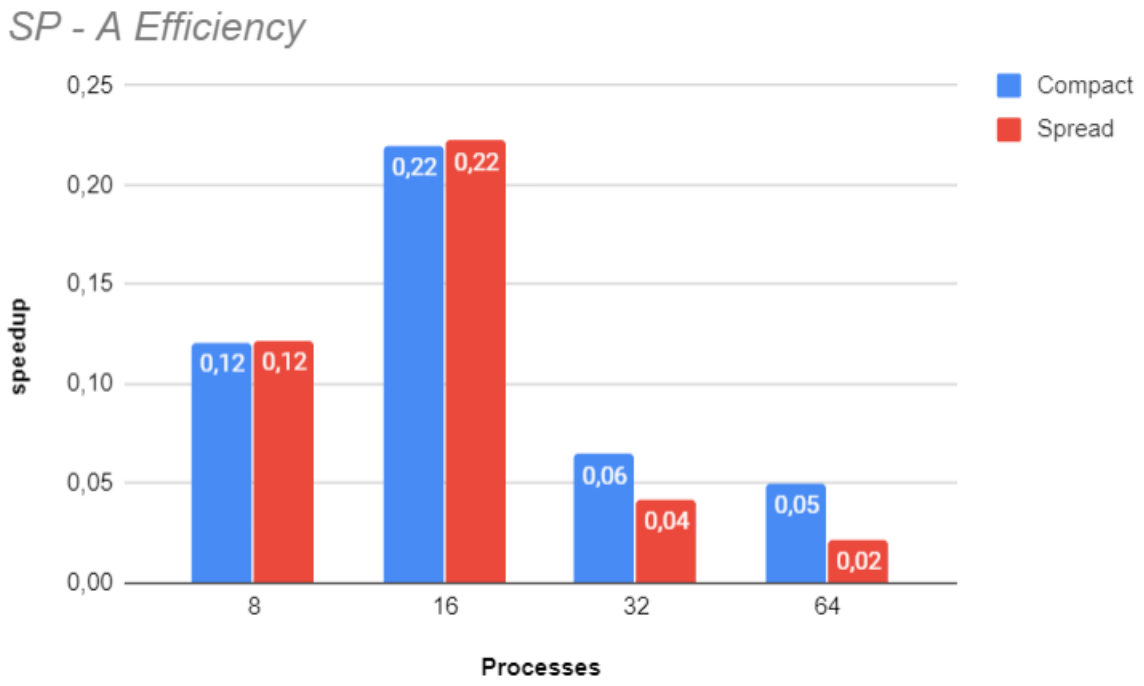
Γραφικές 21: MG - A Efficiency



Γραφικές 22: SP - CLASS A

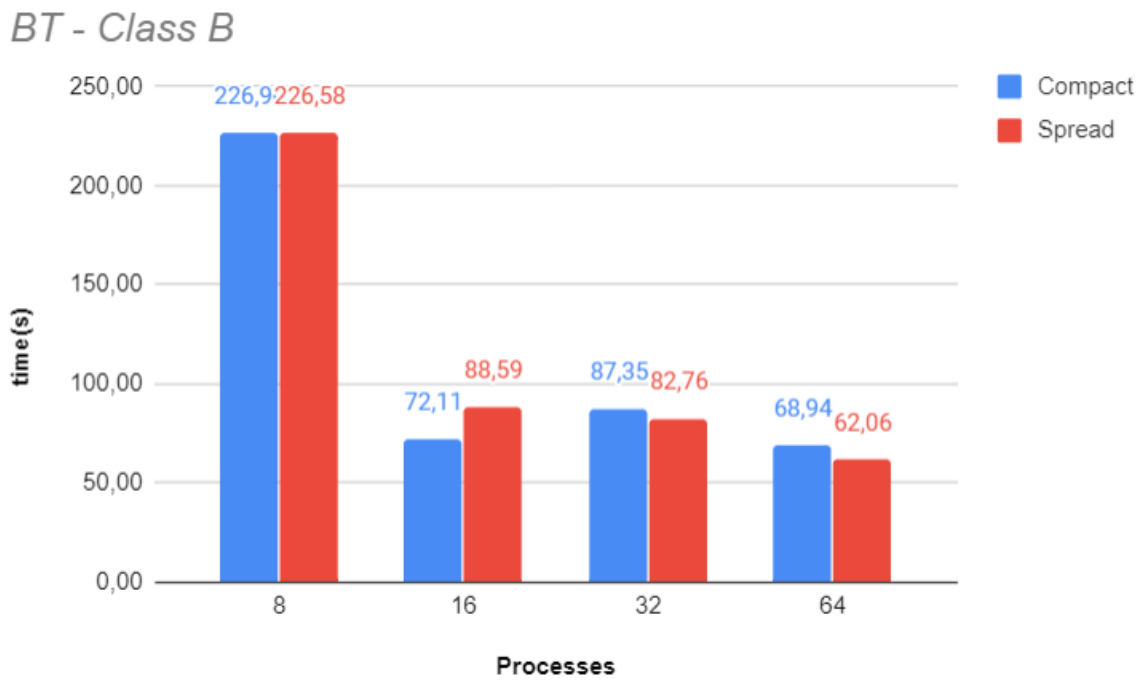


Γραφικές 23: SP - A Speedup

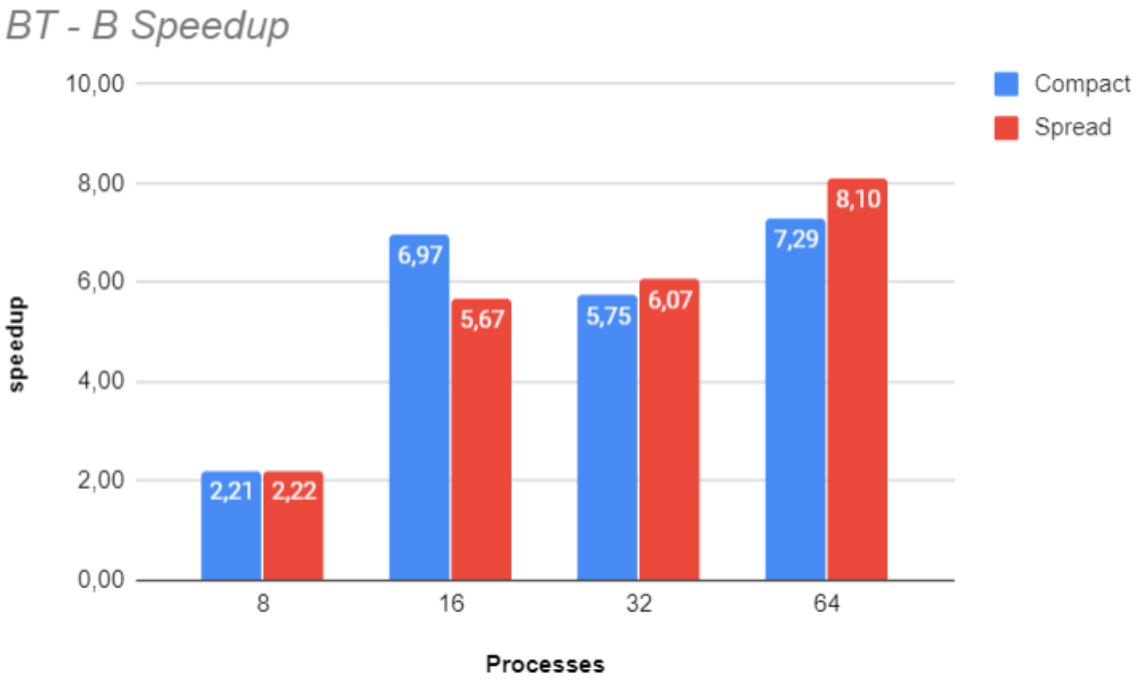


Γραφικές 24: SP - A Efficiency

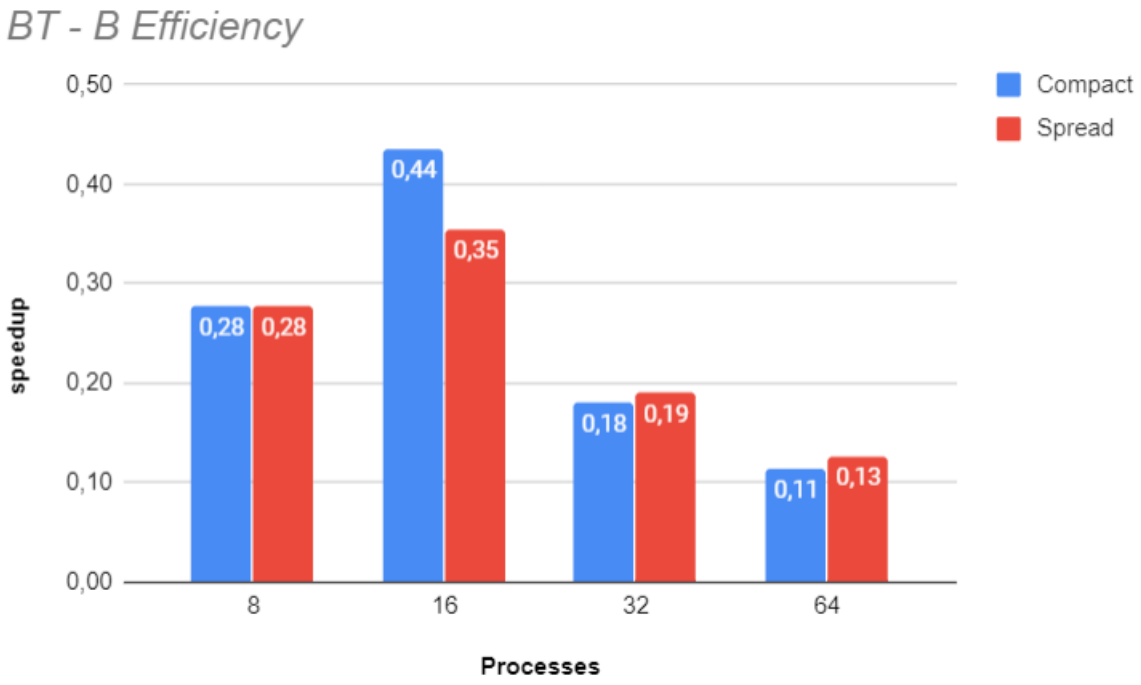
5.3.2 Class B



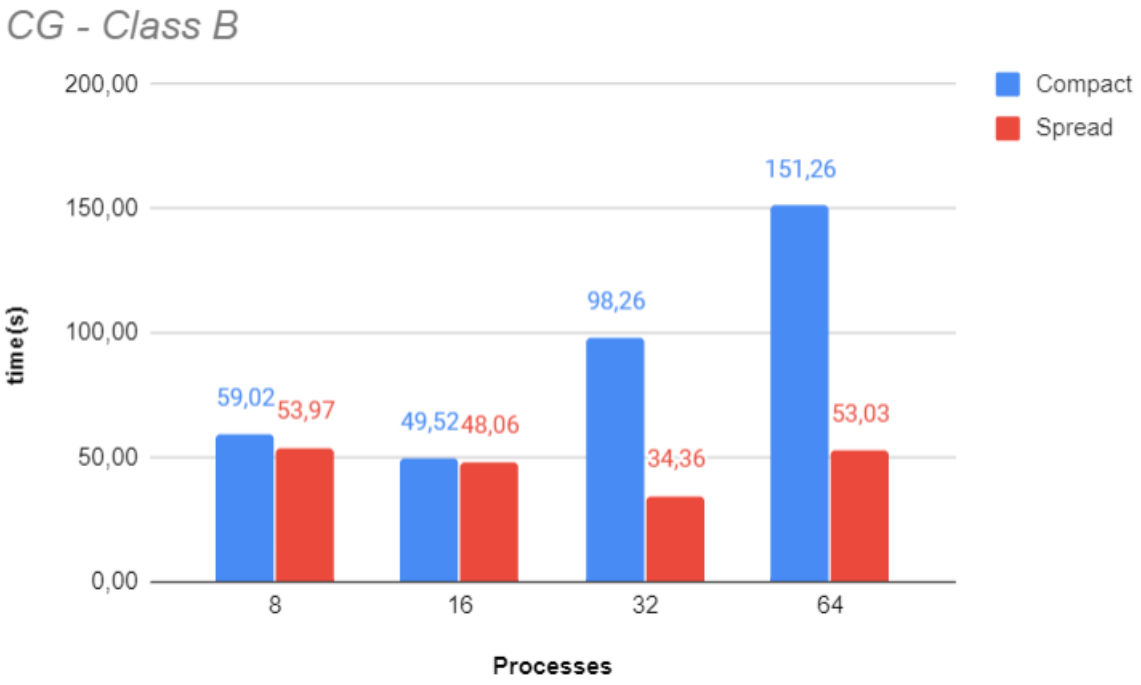
Γραφικές 25: BT - CLASS B



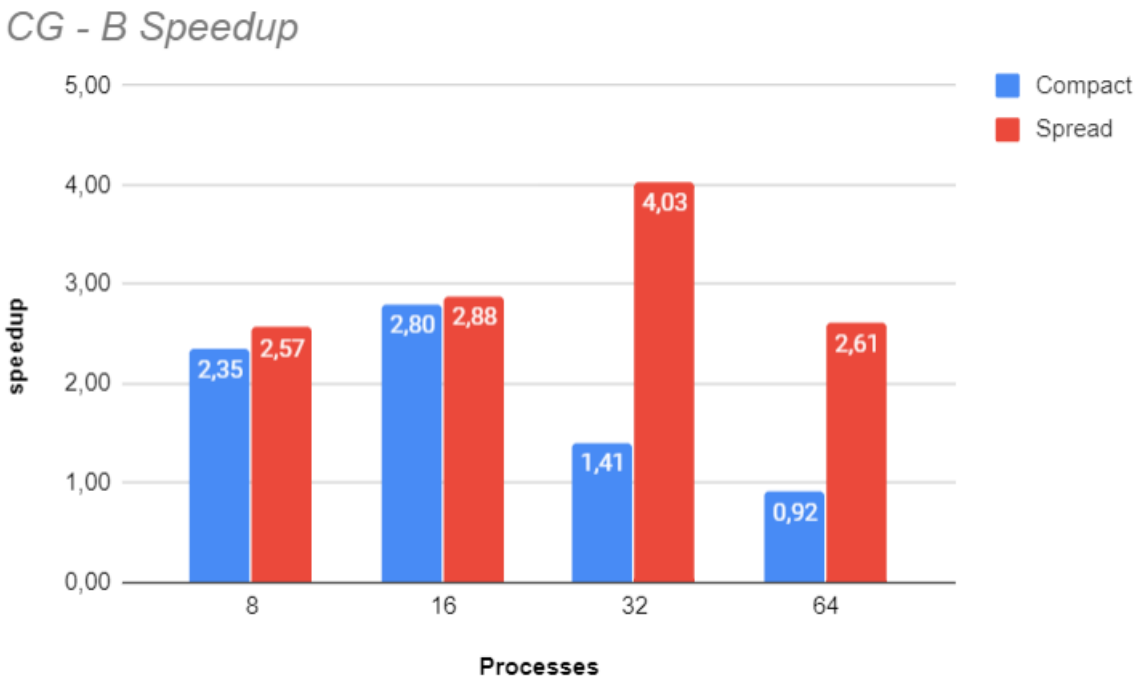
Γραφικές 26: BT - B Speedup



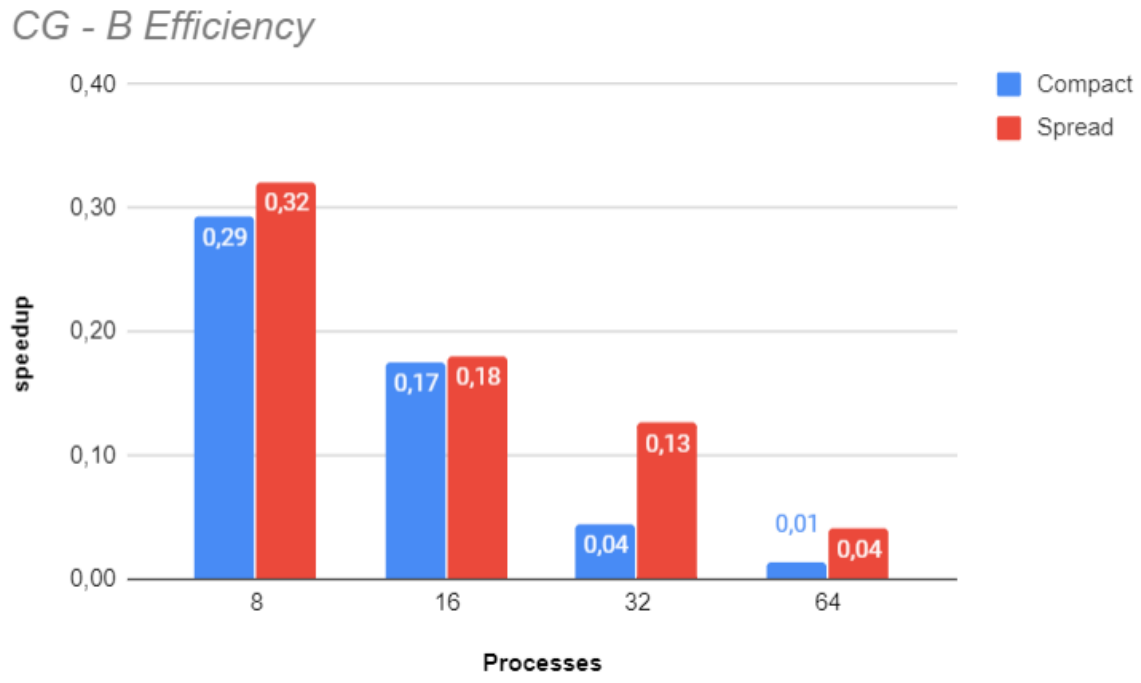
Γραφικές 27: BT - B Efficiency



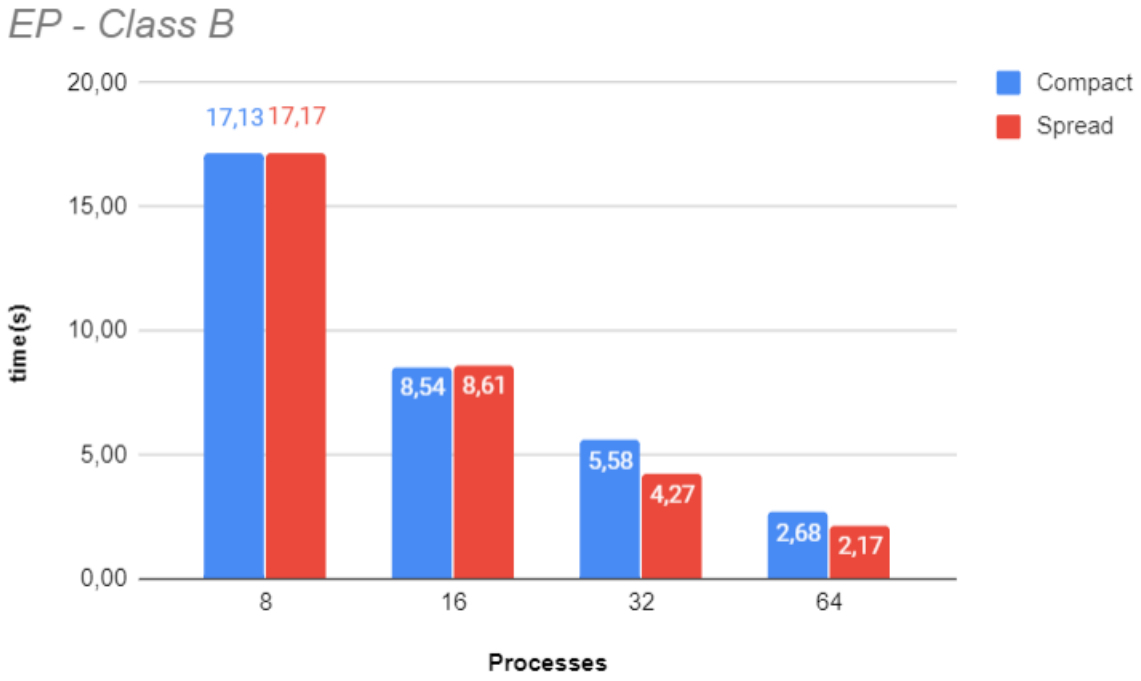
Γραφικές 28: CG - CLASS B



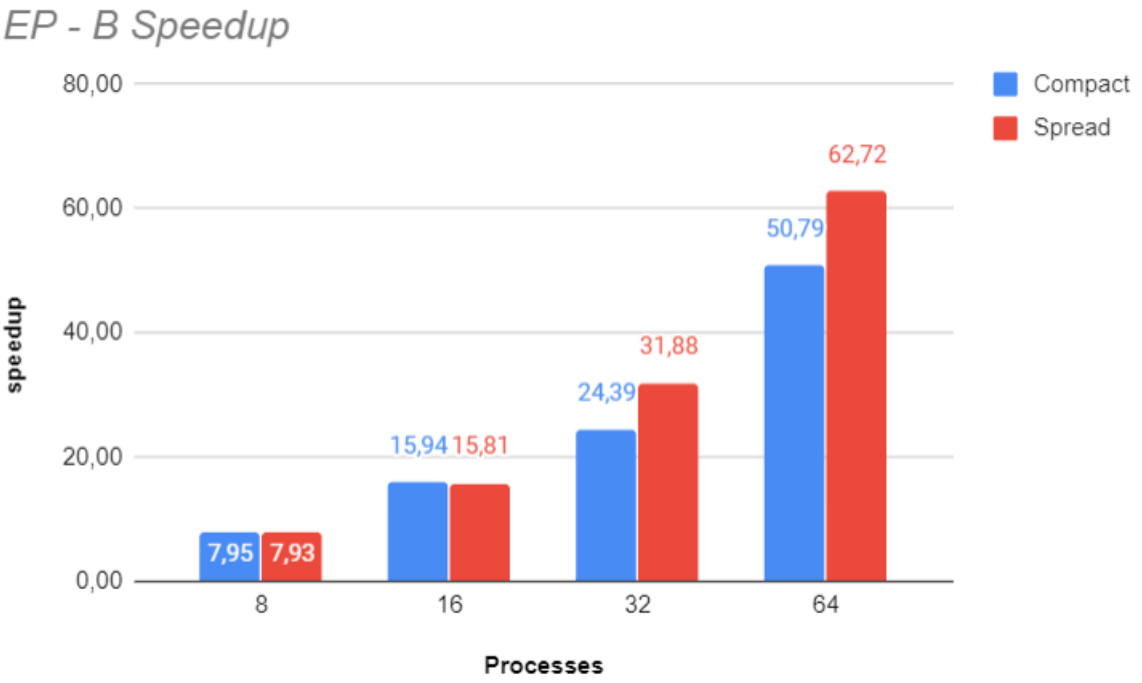
Γραφικές 29: CG - B Speedup



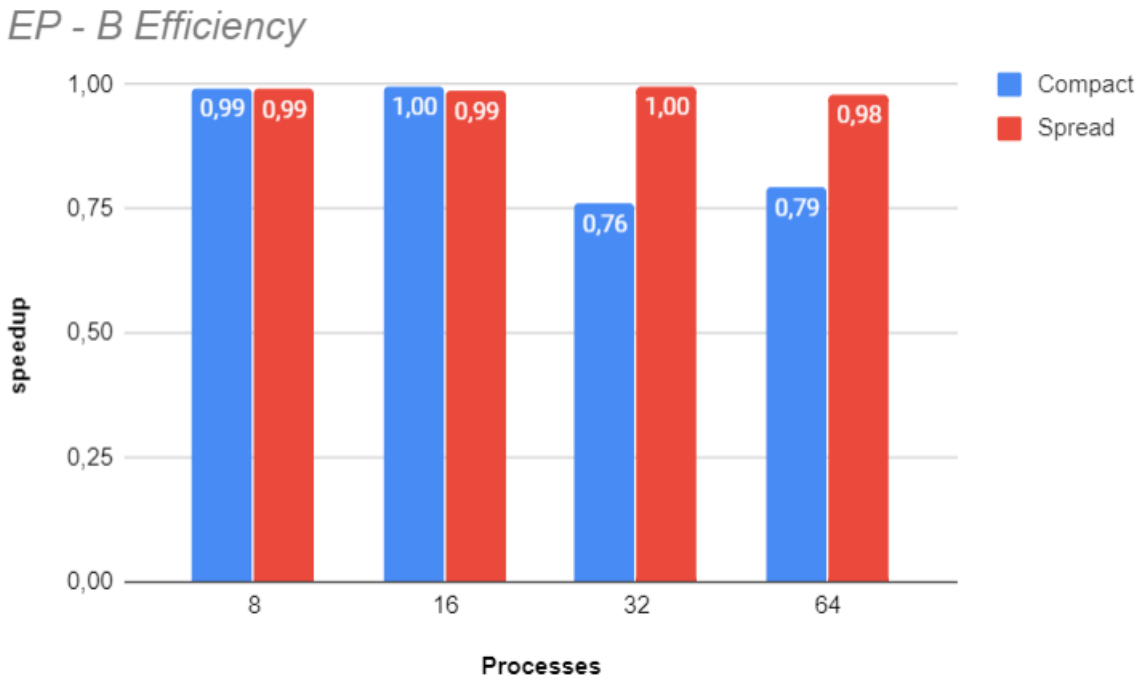
Γραφικές 30: CG - B Efficiency



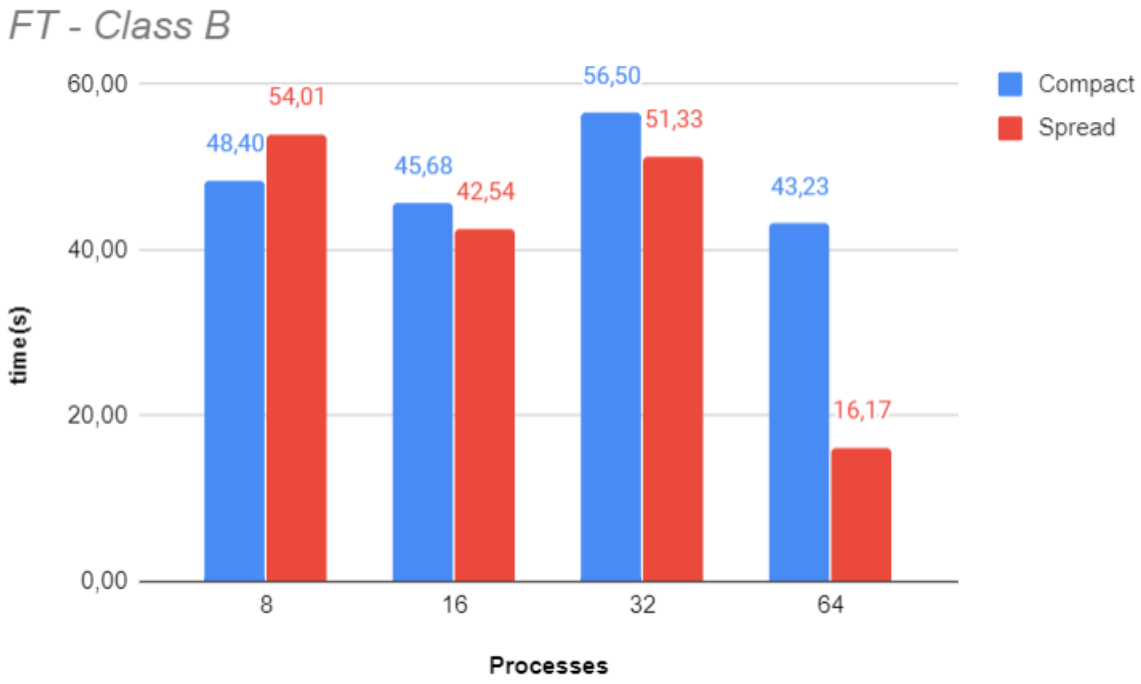
Γραφικές 31: EP - CLASS B



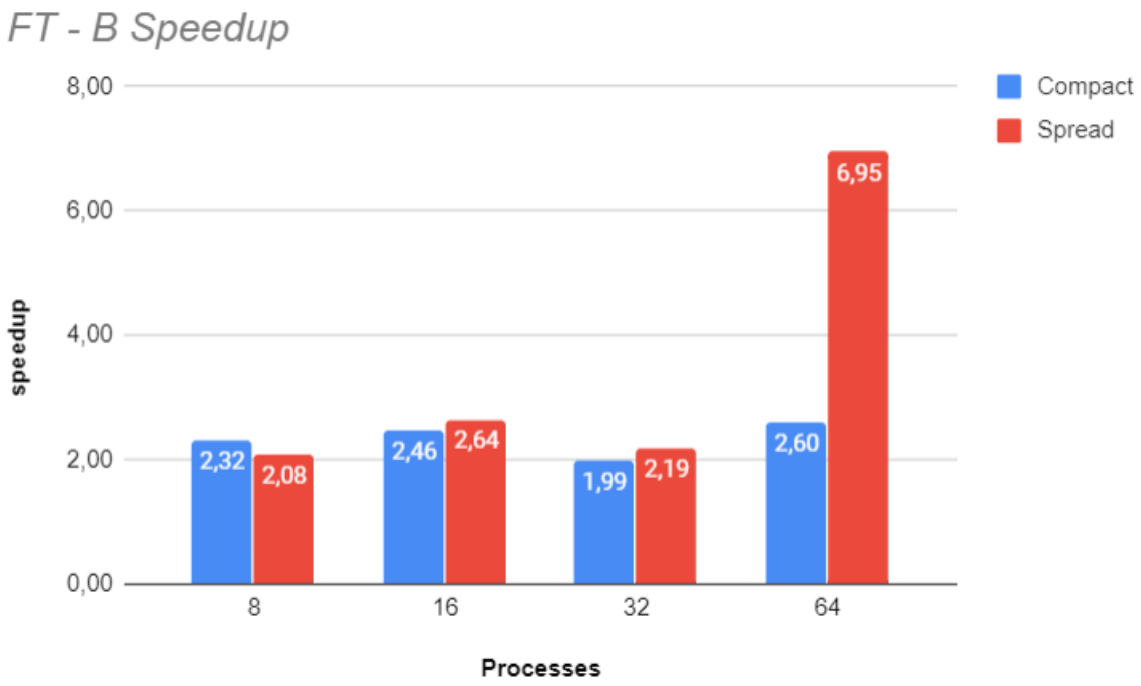
Γραφικές 32: EP - B Speedup



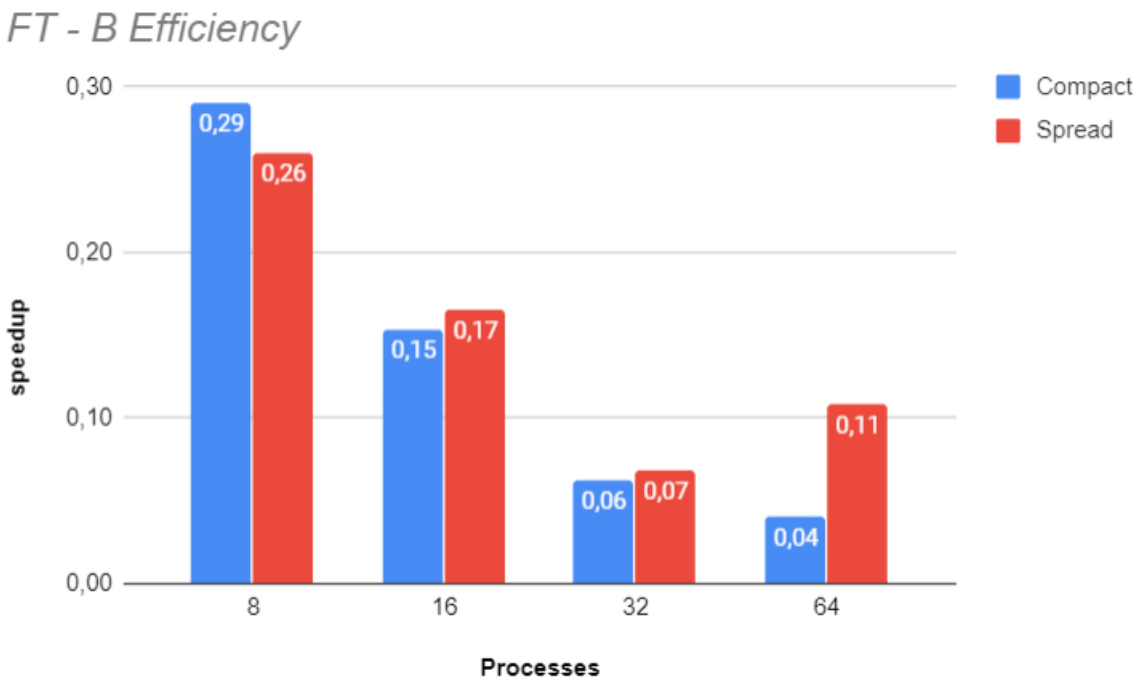
Γραφικές 33: EP - B Efficiency



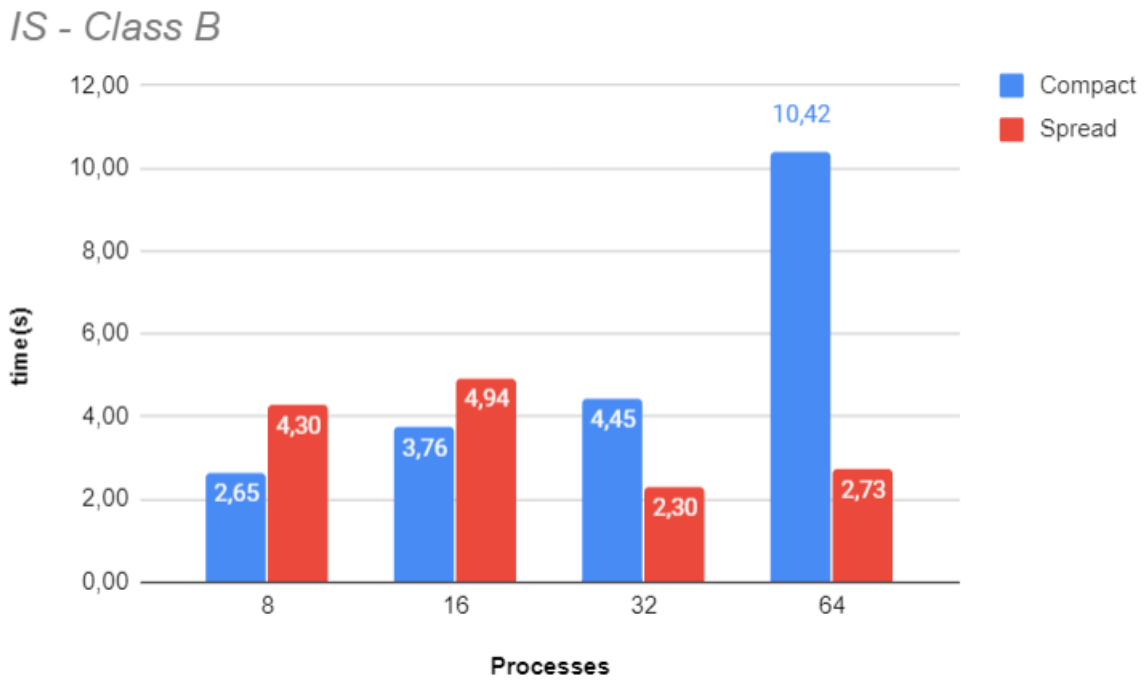
Γραφικές 34: FT - CLASS B



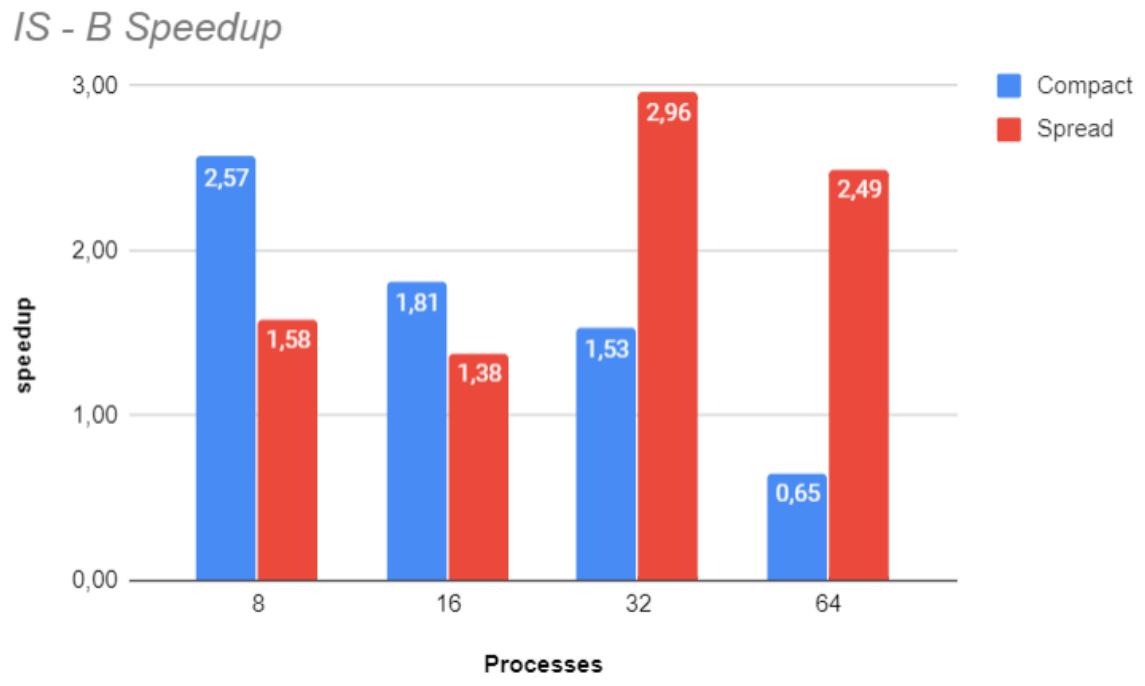
Γραφικές 35: FT - B Speedup



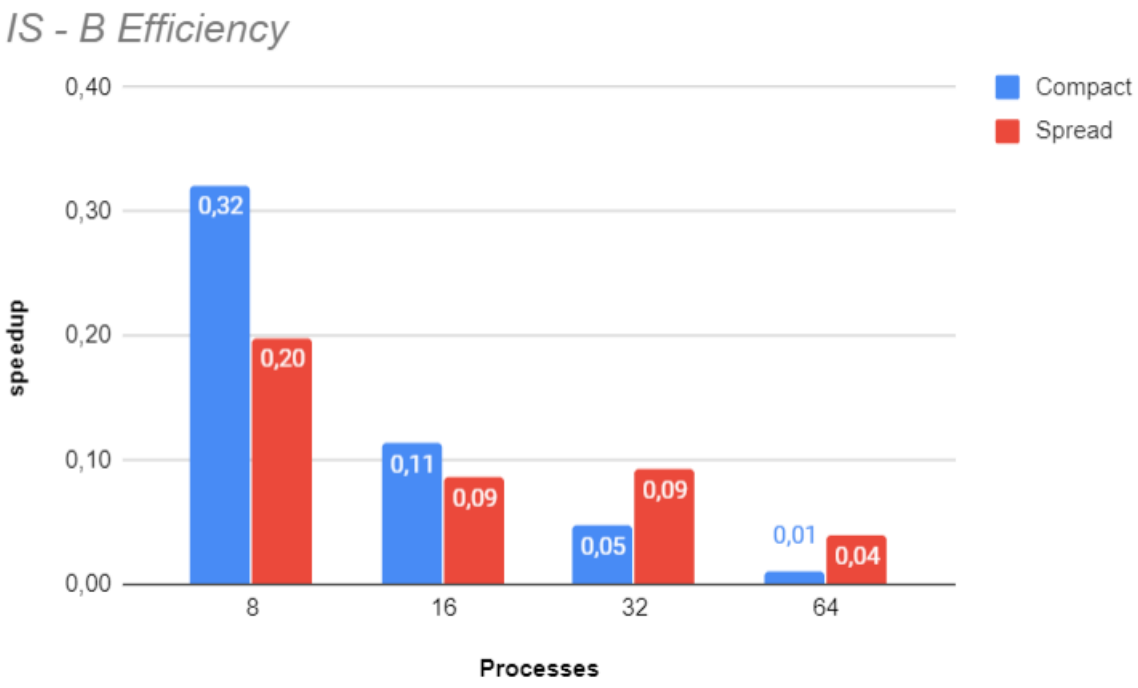
Γραφικές 36: FT - B Efficiency



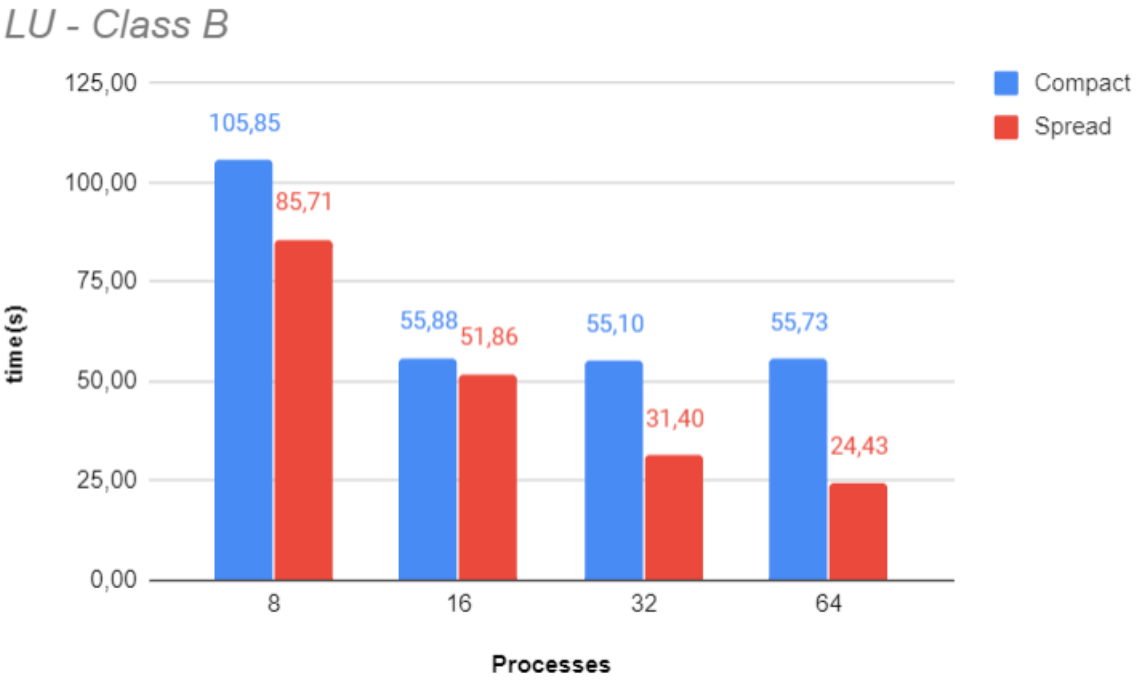
Γραφικές 37: IS - CLASS B



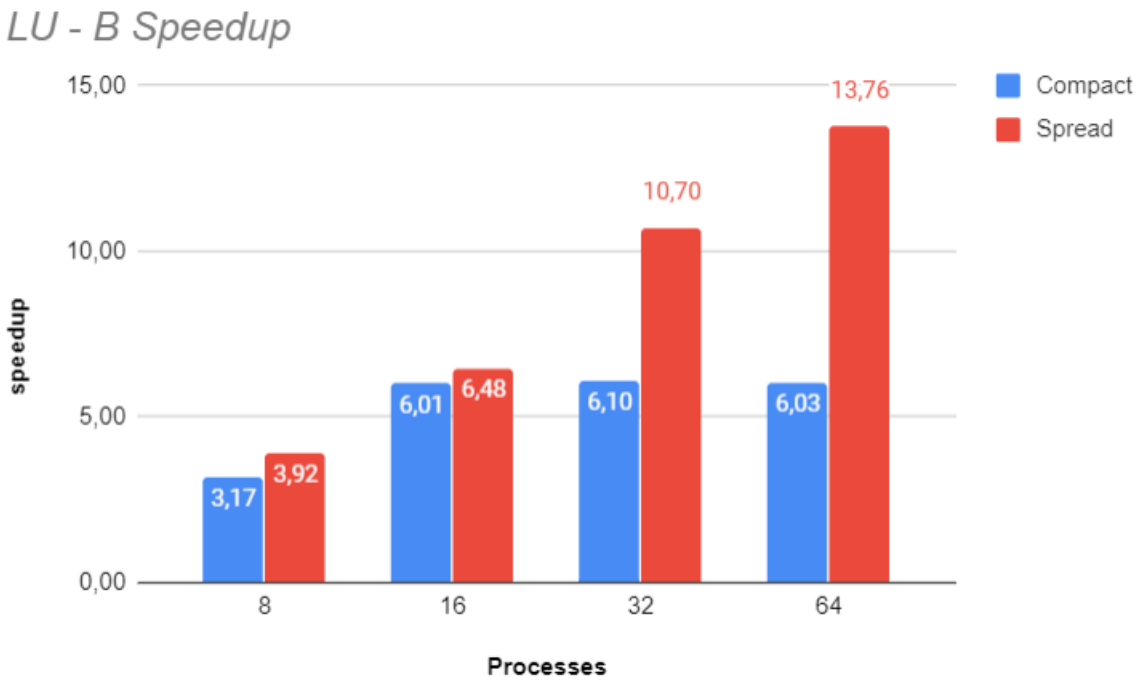
Γραφικές 38: IS - B Speedup



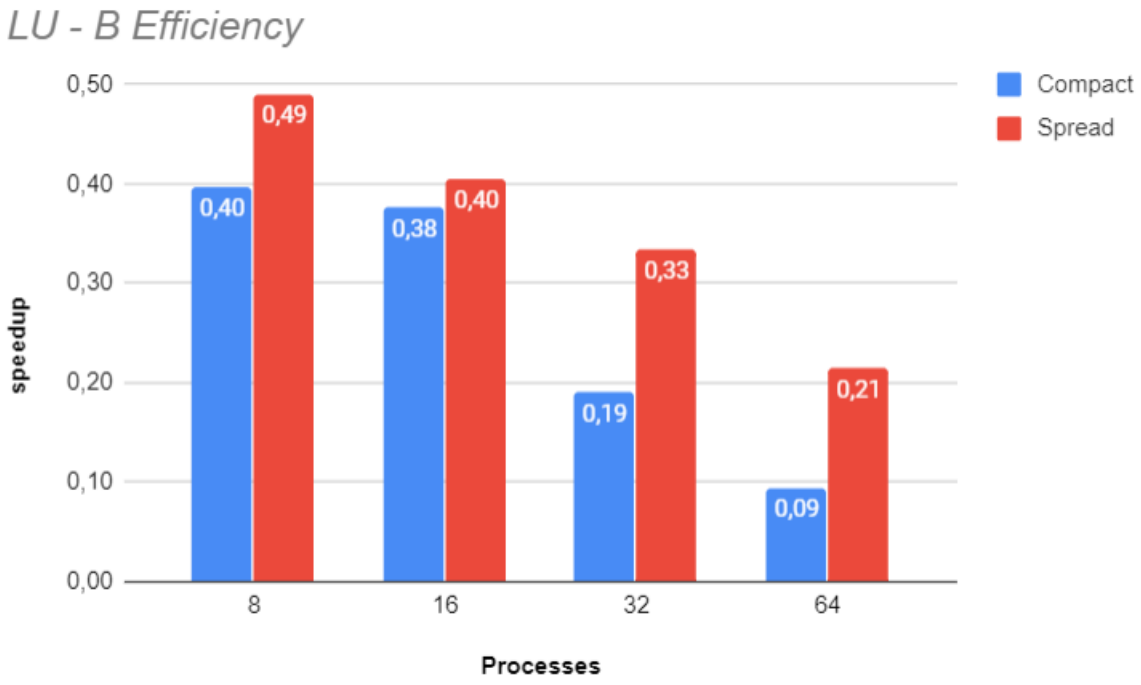
Γραφικές 39: IS - B Efficiency



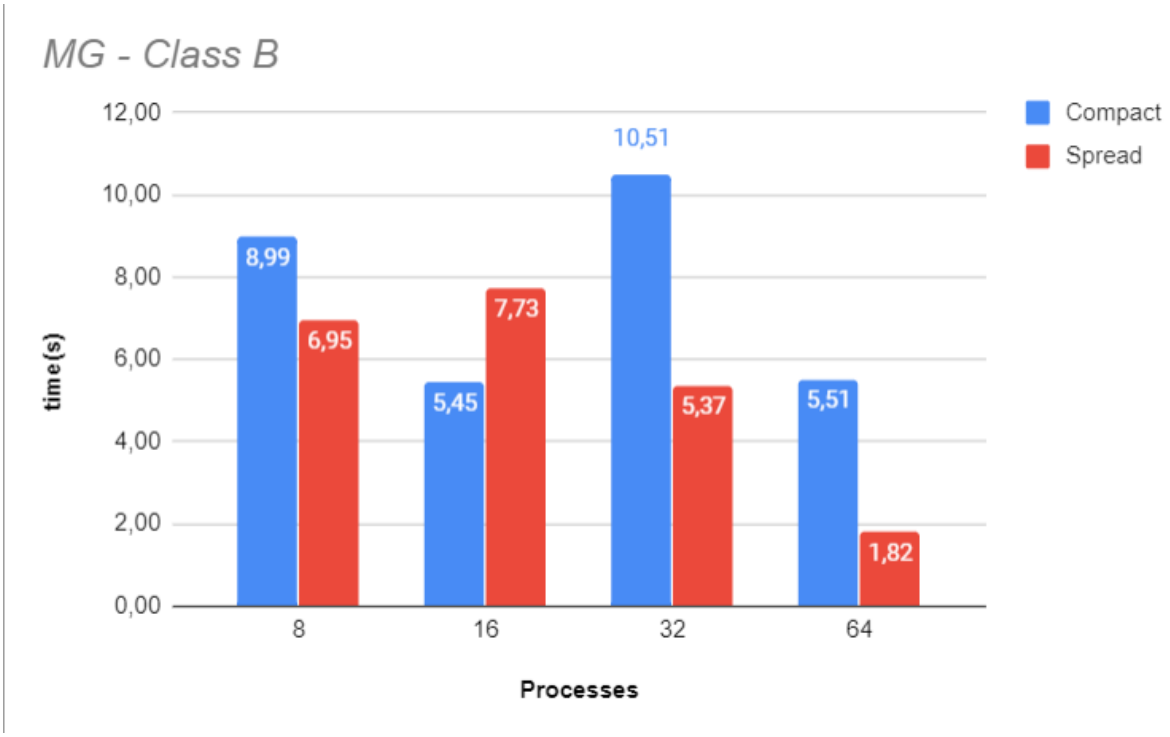
Γραφικές 40: LU - CLASS B



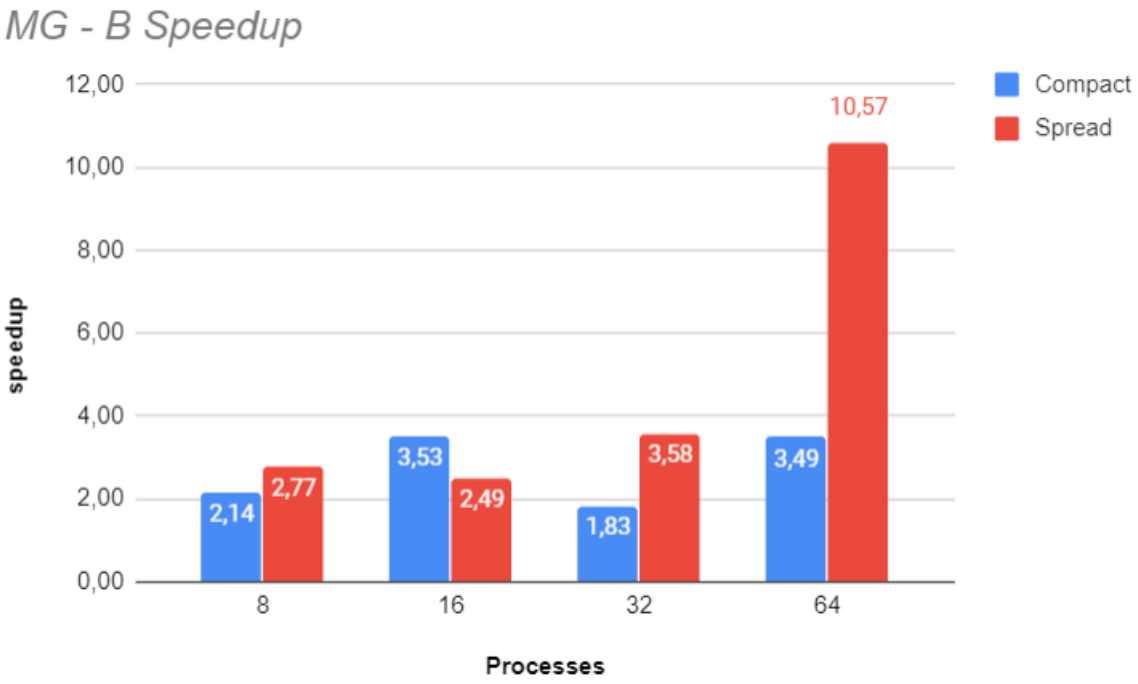
Γραφικές 41: LU - B Speedup



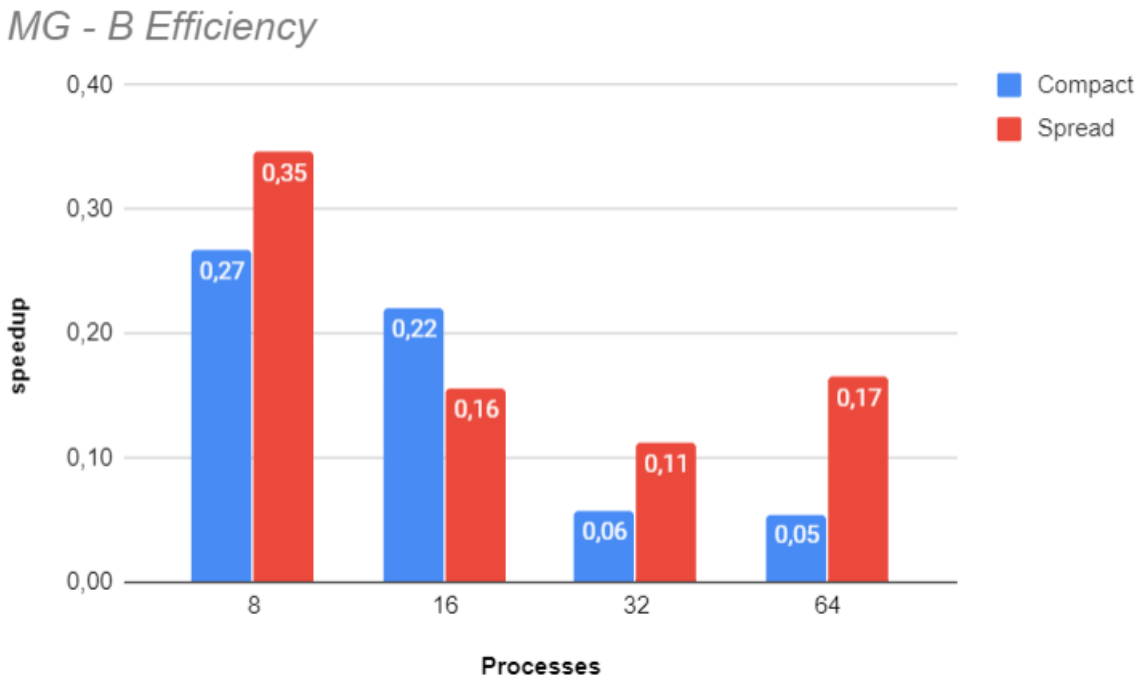
Γραφικές 42: LU - B Efficiency



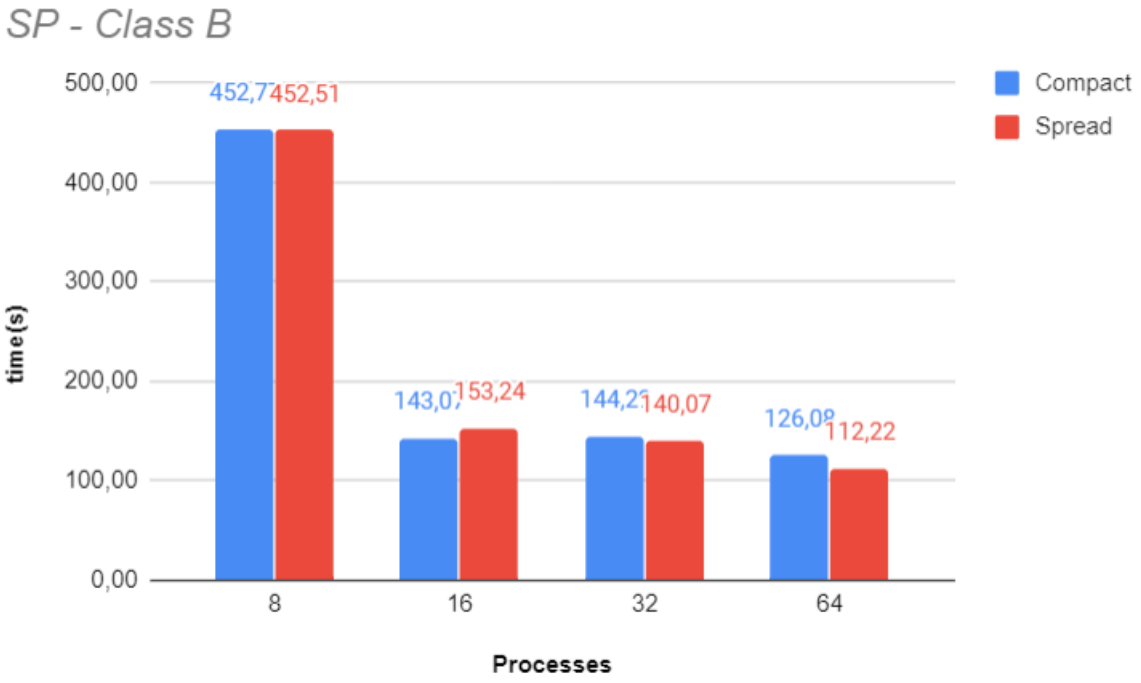
Γραφικές 43: MG - CLASS B



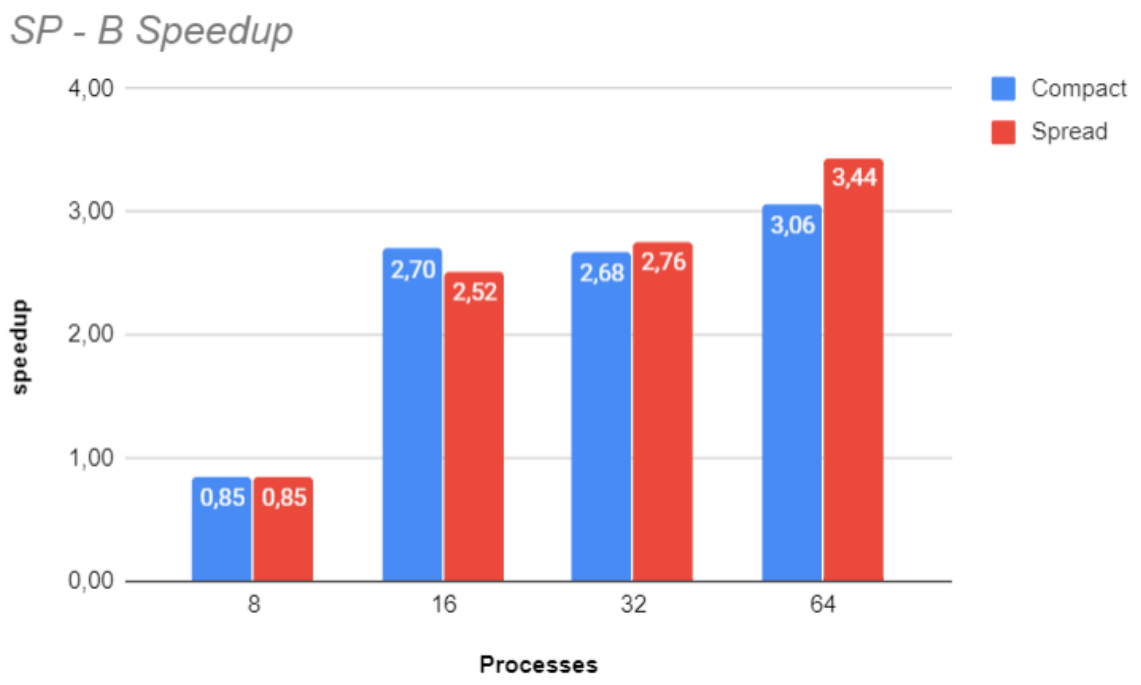
Γραφικές 44: MG - B Speedup



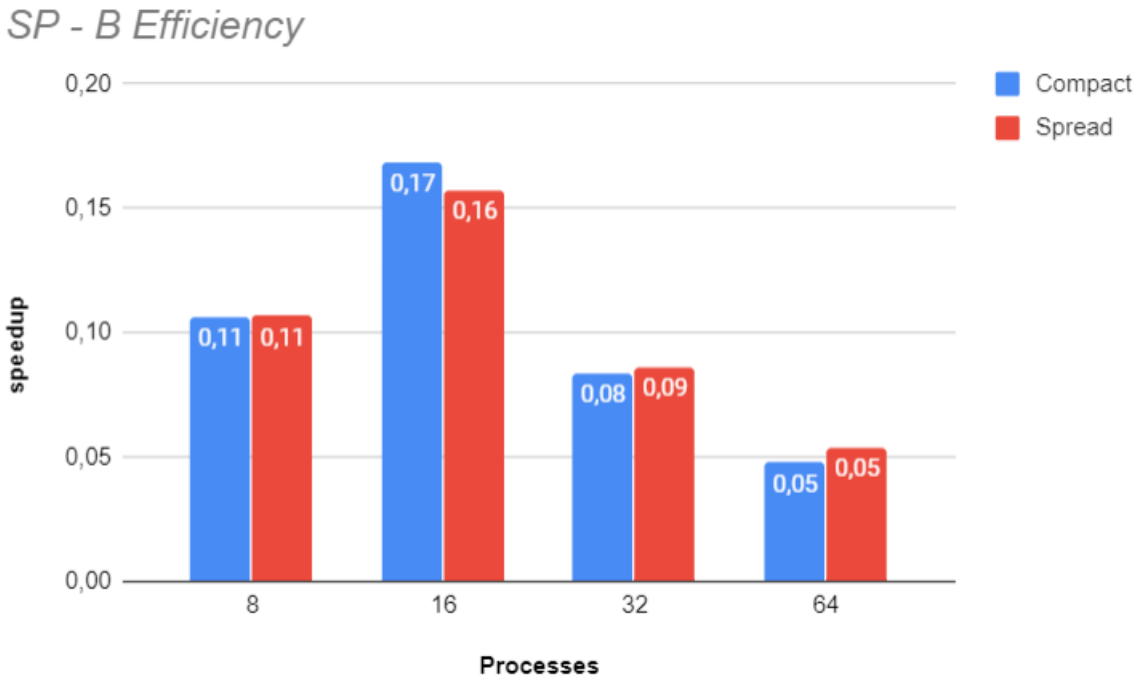
Γραφικές 45: MG - B Efficiency



Γραφικές 46: SP - CLASS B

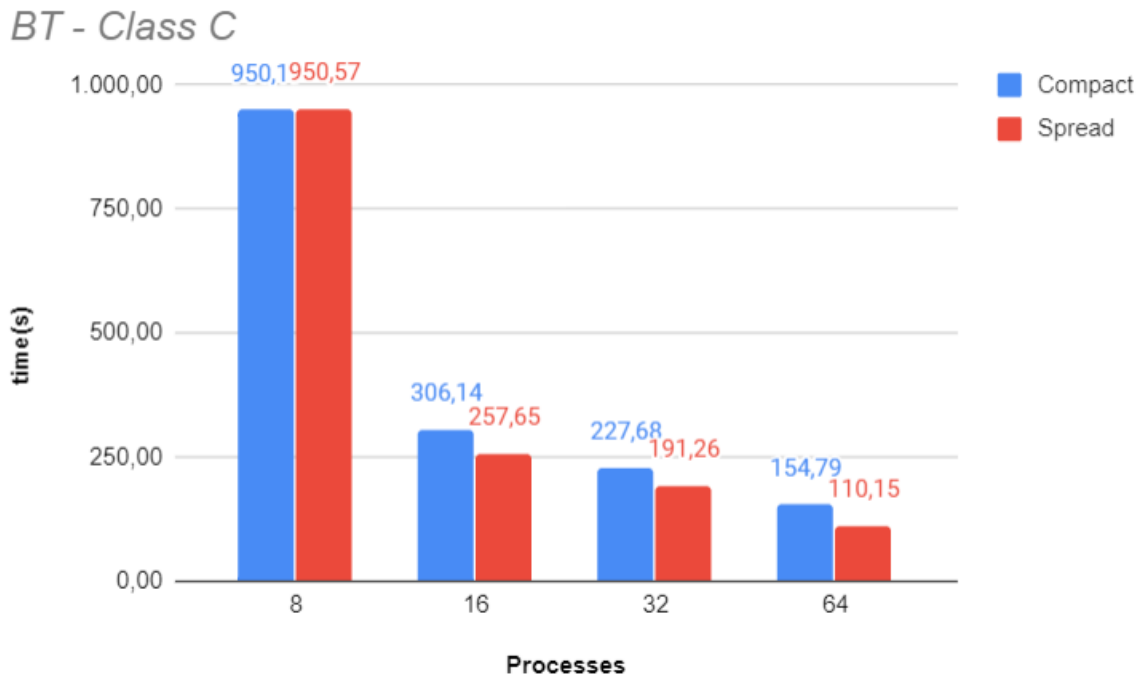


Γραφικές 47: SP - B Speedup

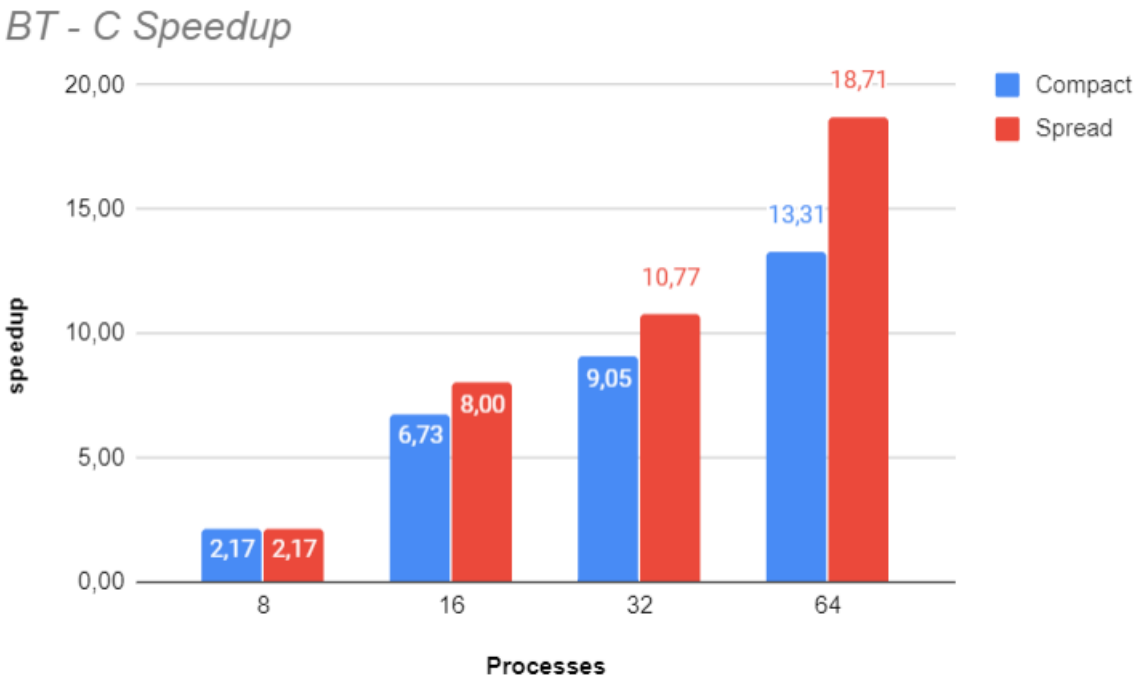


Γραφικές 48: SP - B Efficiency

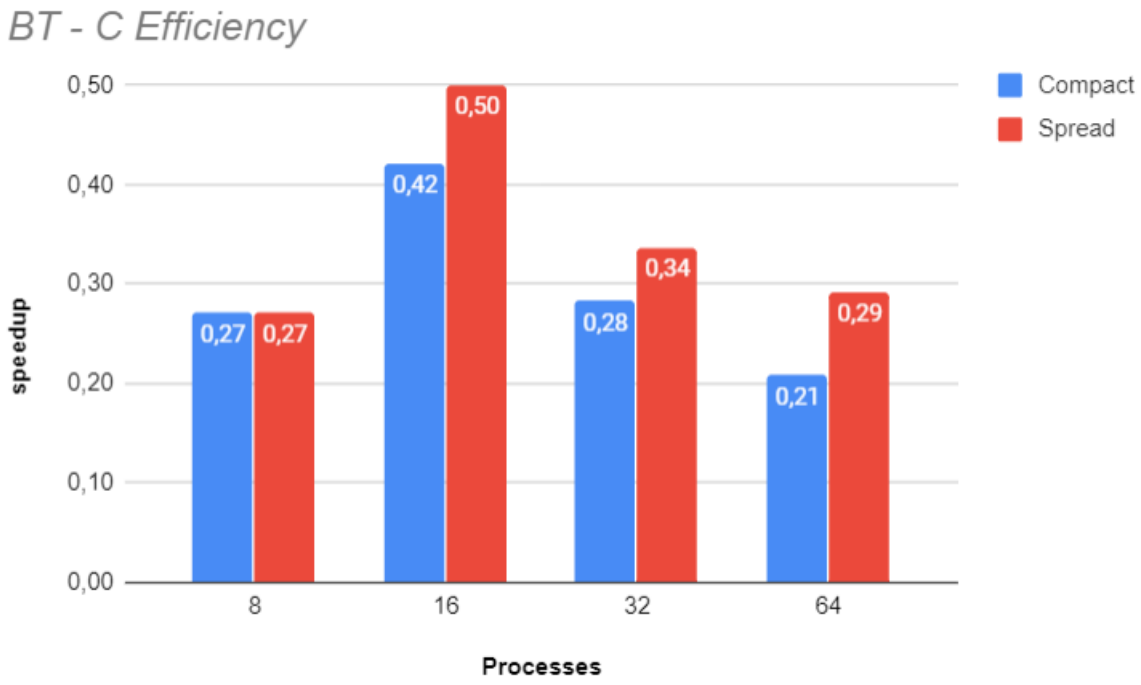
5.3.3 Class C



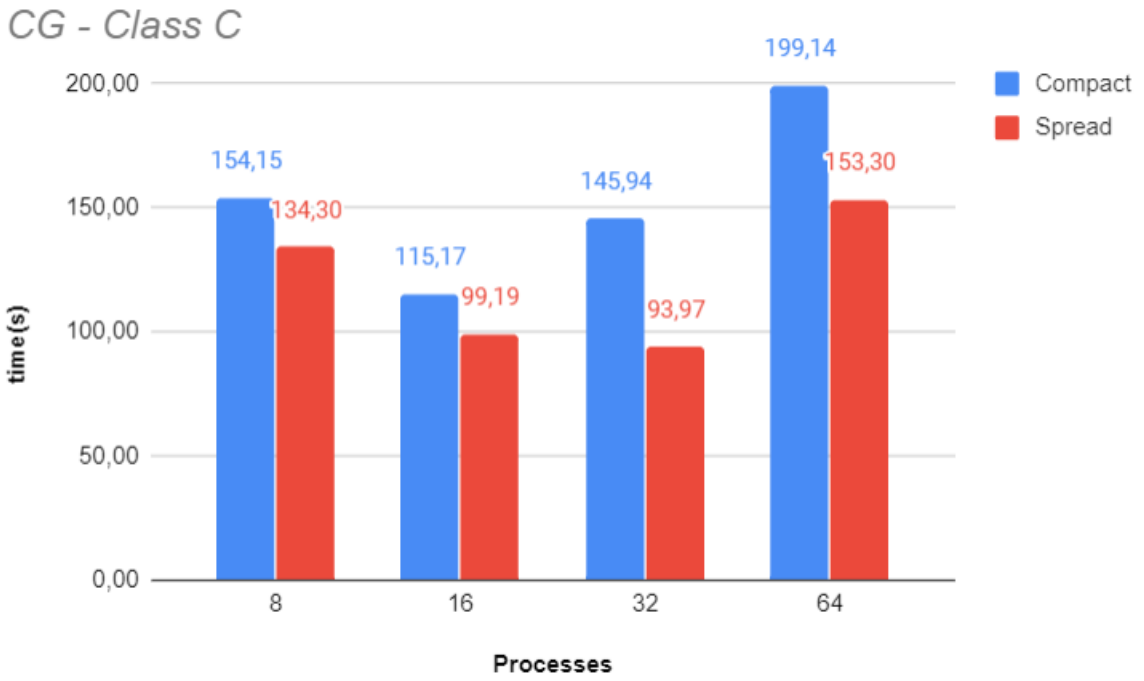
Γραφικές 49: BT - CLASS C



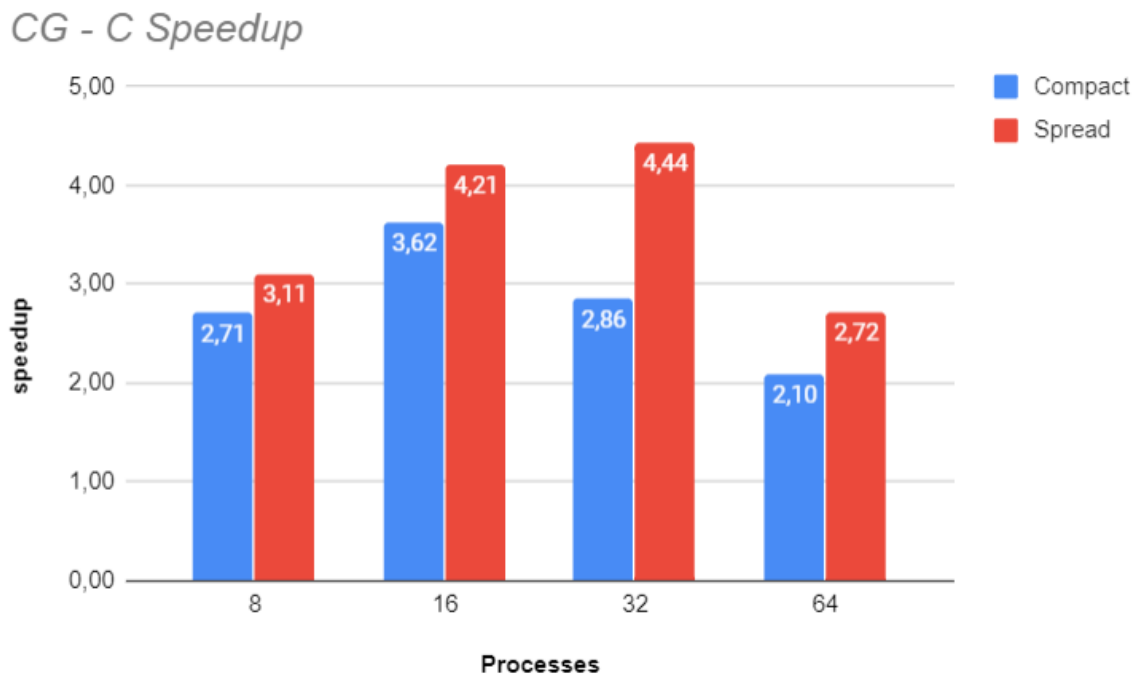
Γραφικές 50: BT - C Speedup



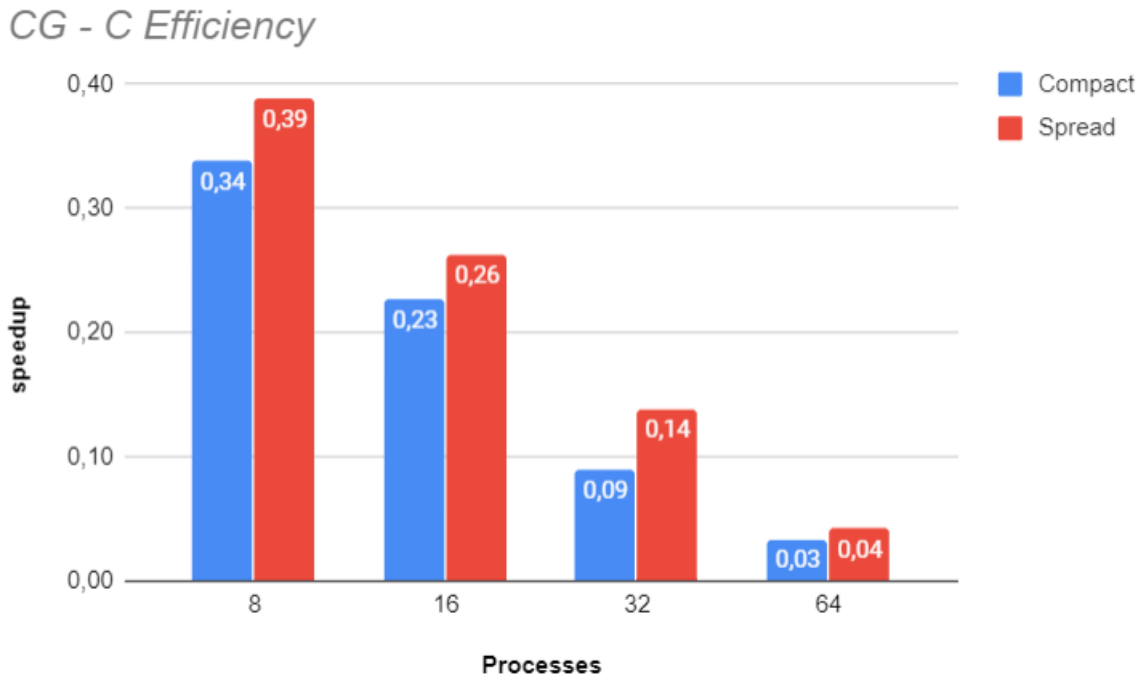
Γραφικές 51: BT - C Efficiency



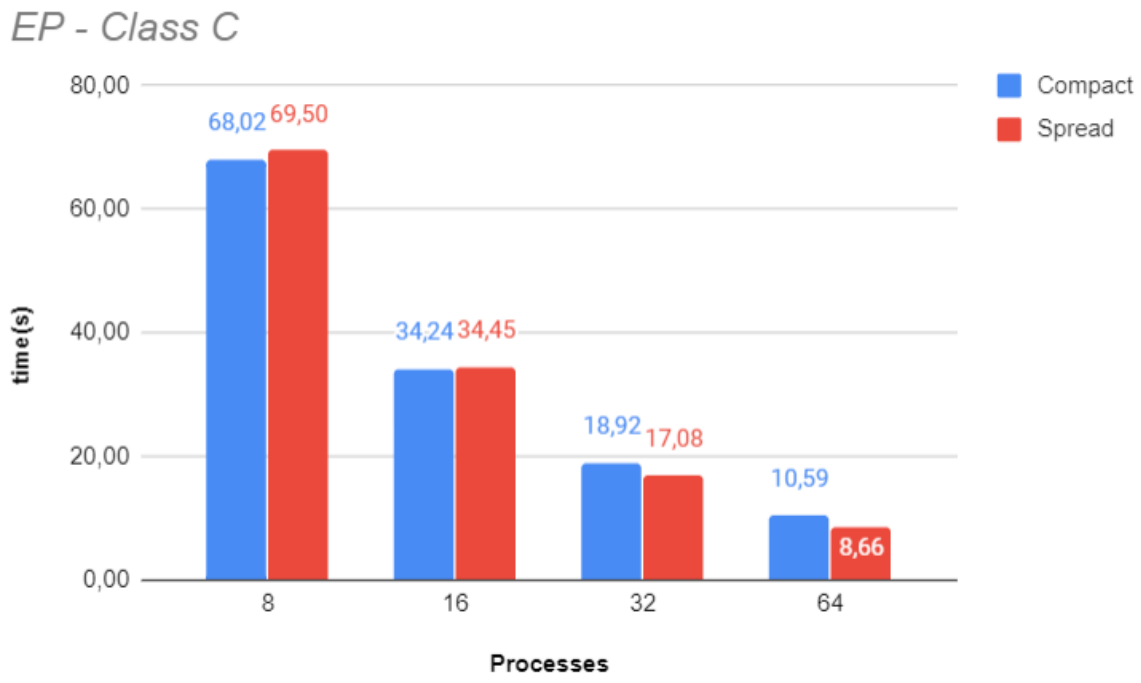
Γραφικές 52: CG - CLASS C



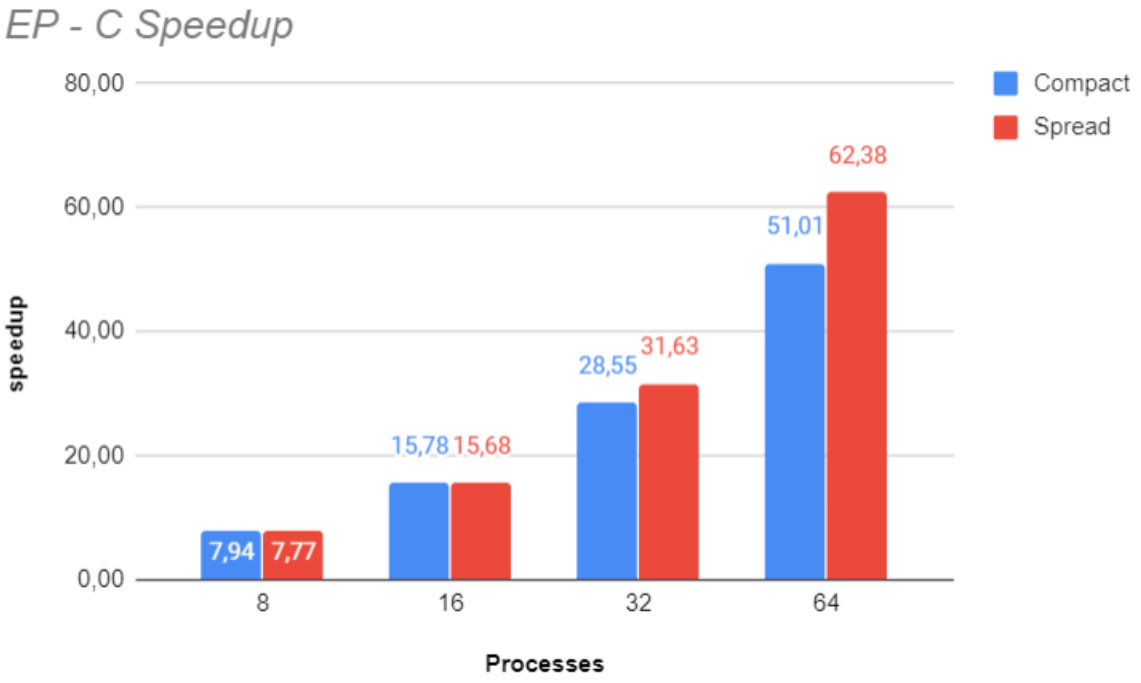
Γραφικές 53: CG - C Speedup



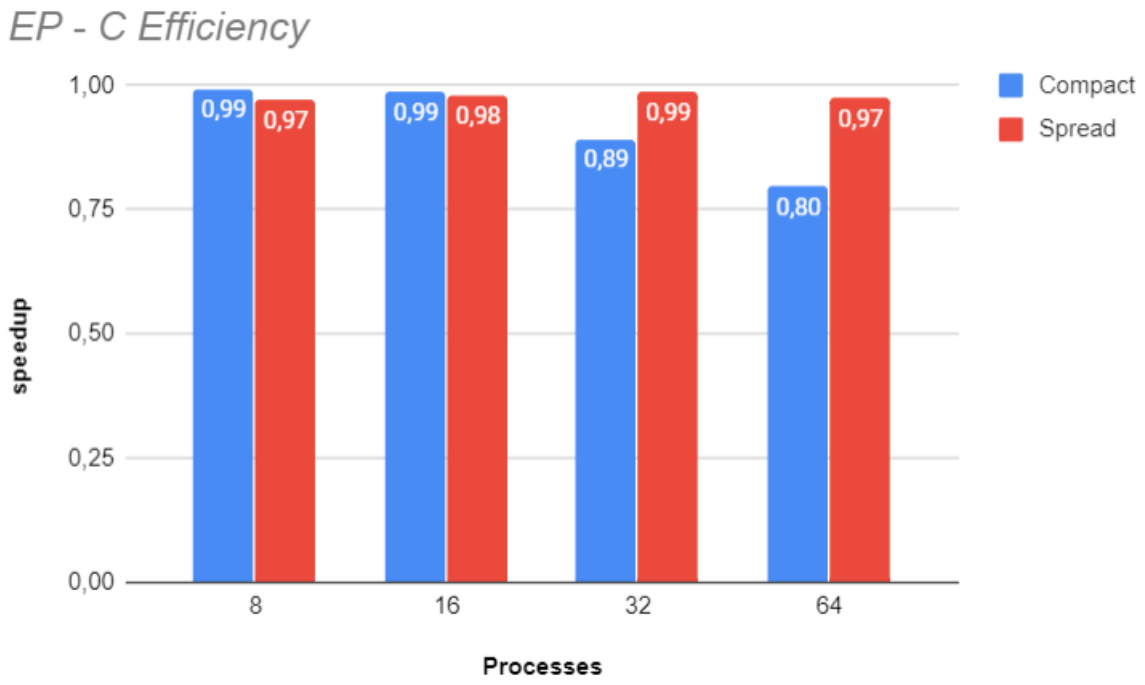
Γραφικές 54: CG - C Efficiency



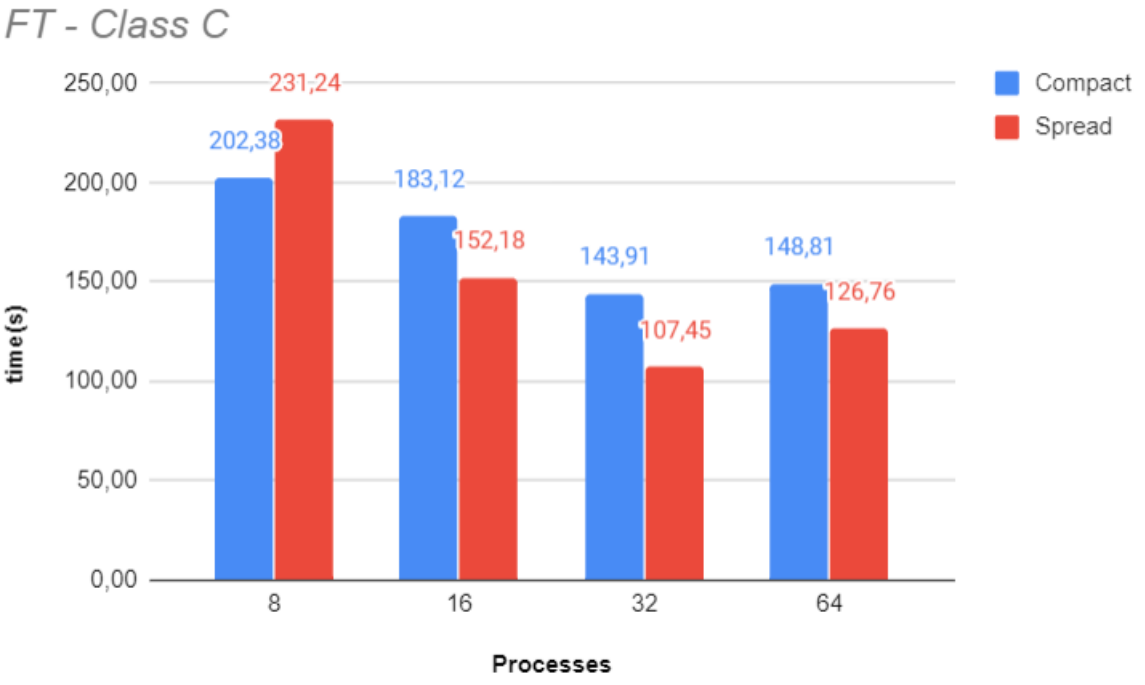
Γραφικές 55: EP - CLASS C



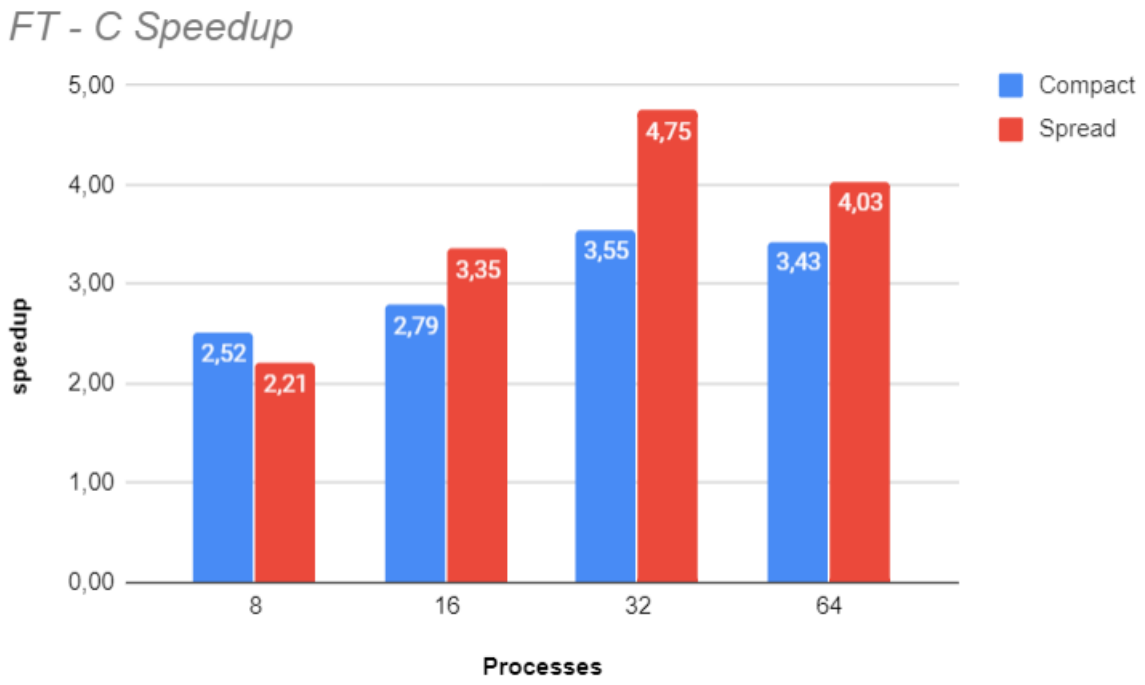
Γραφικές 56: EP - C Speedup



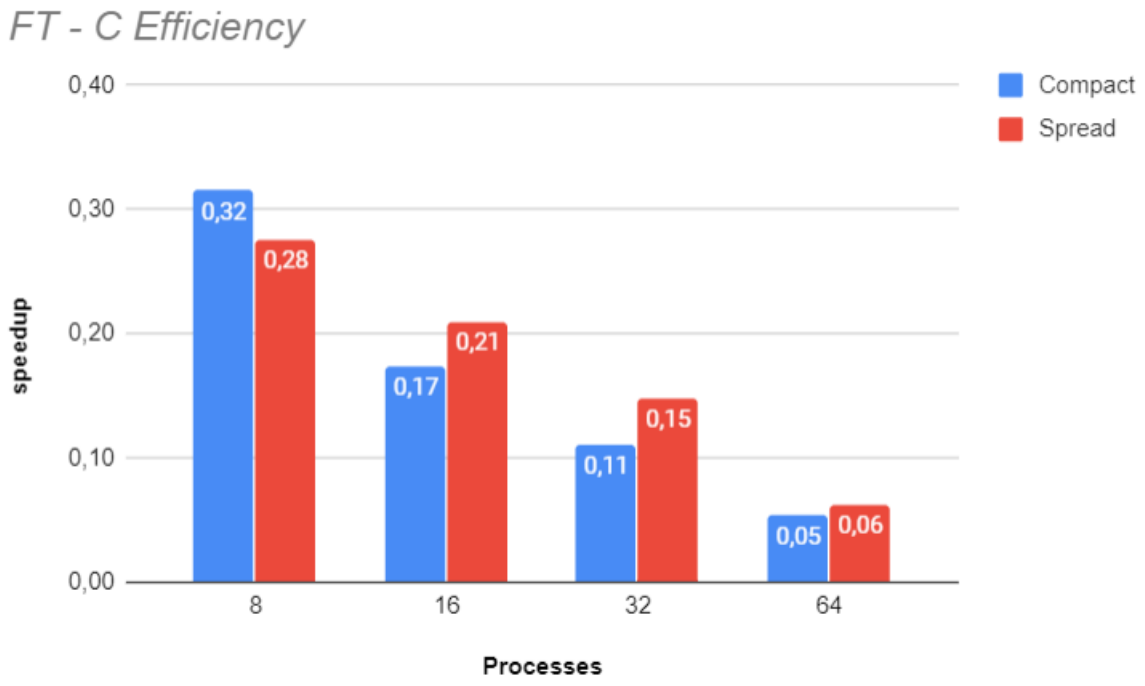
Γραφικές 57: EP - C Efficiency



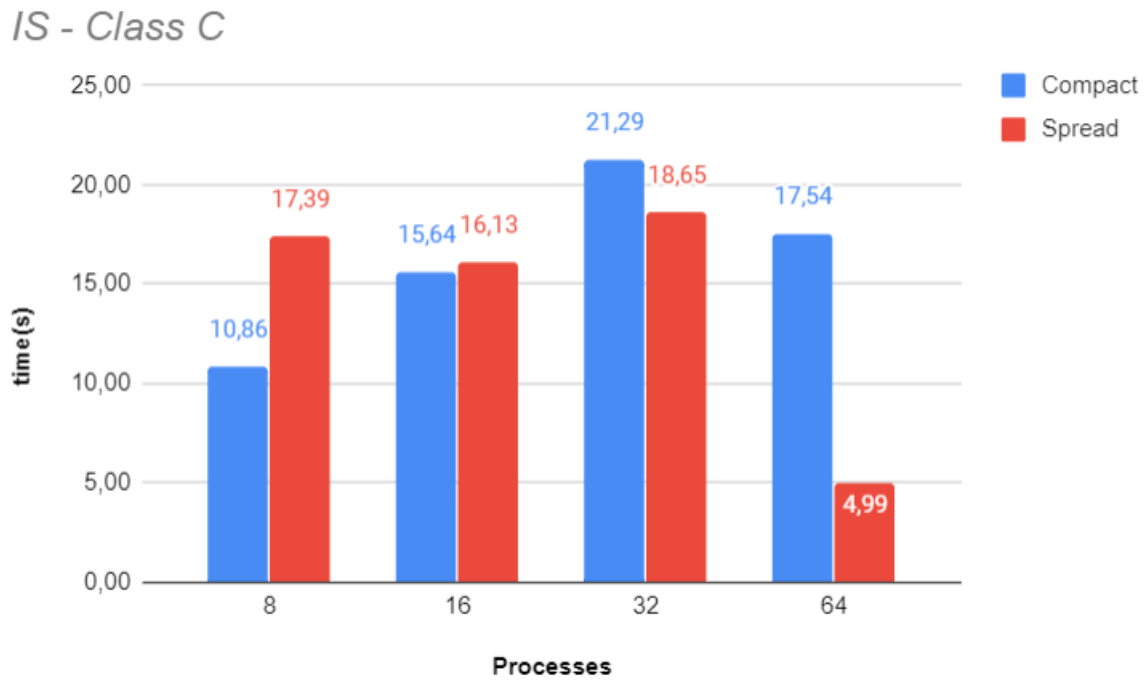
Γραφικές 58: FT - CLASS C



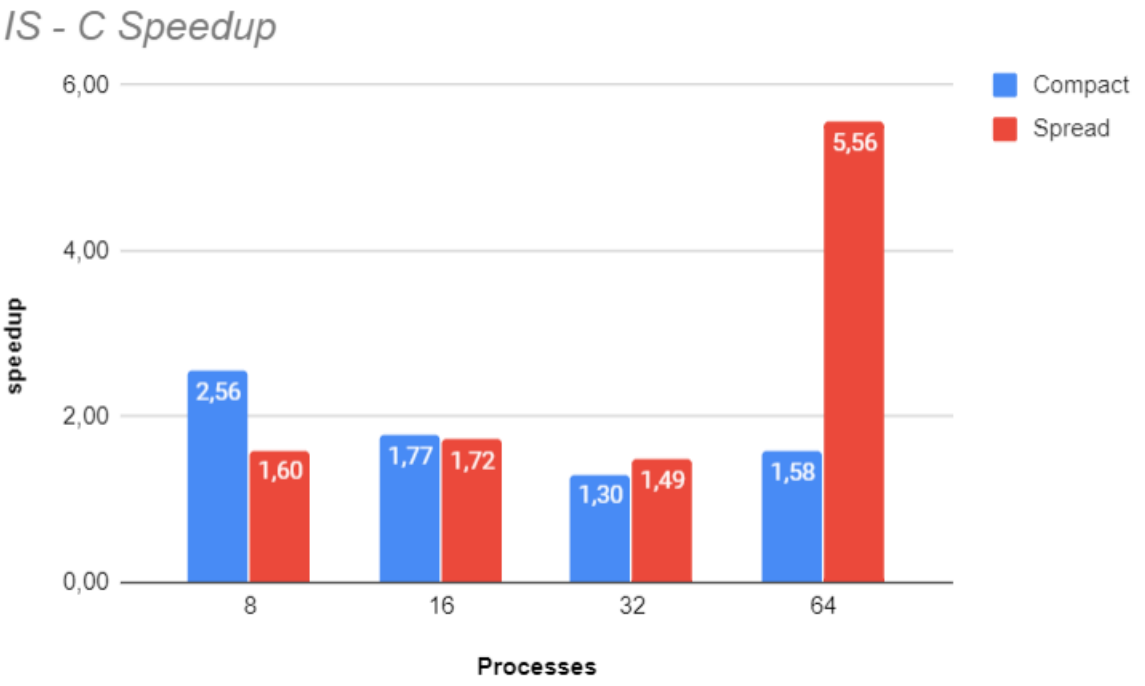
Γραφικές 59: FT - C Speedup



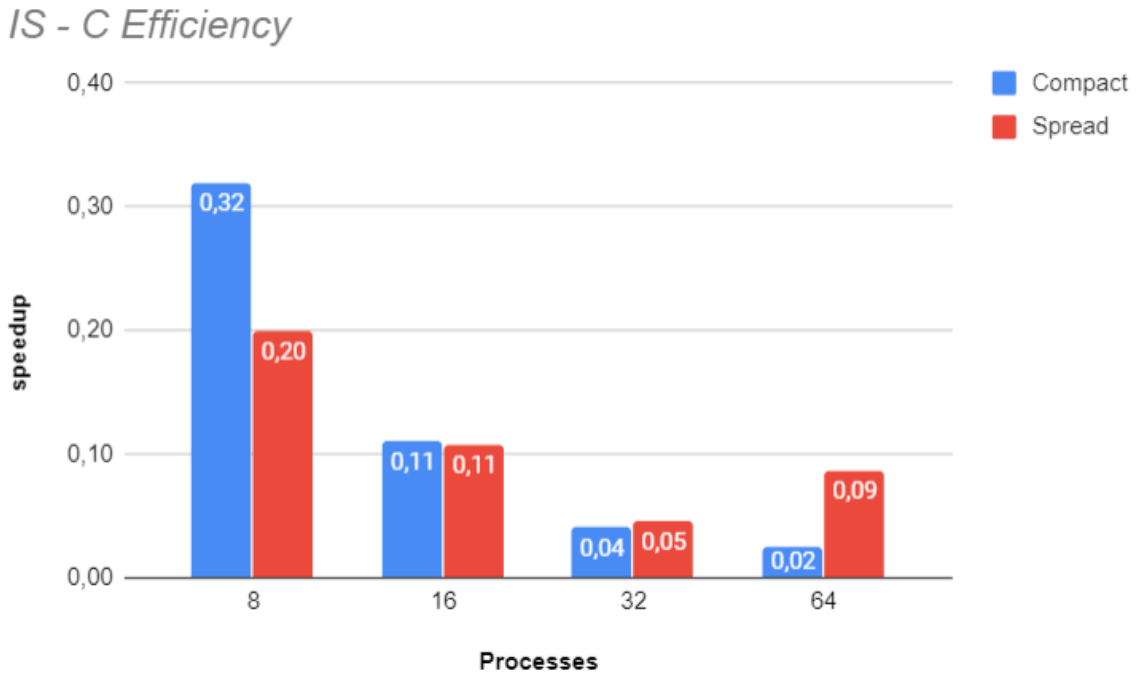
Γραφικές 60: FT - C Efficiency



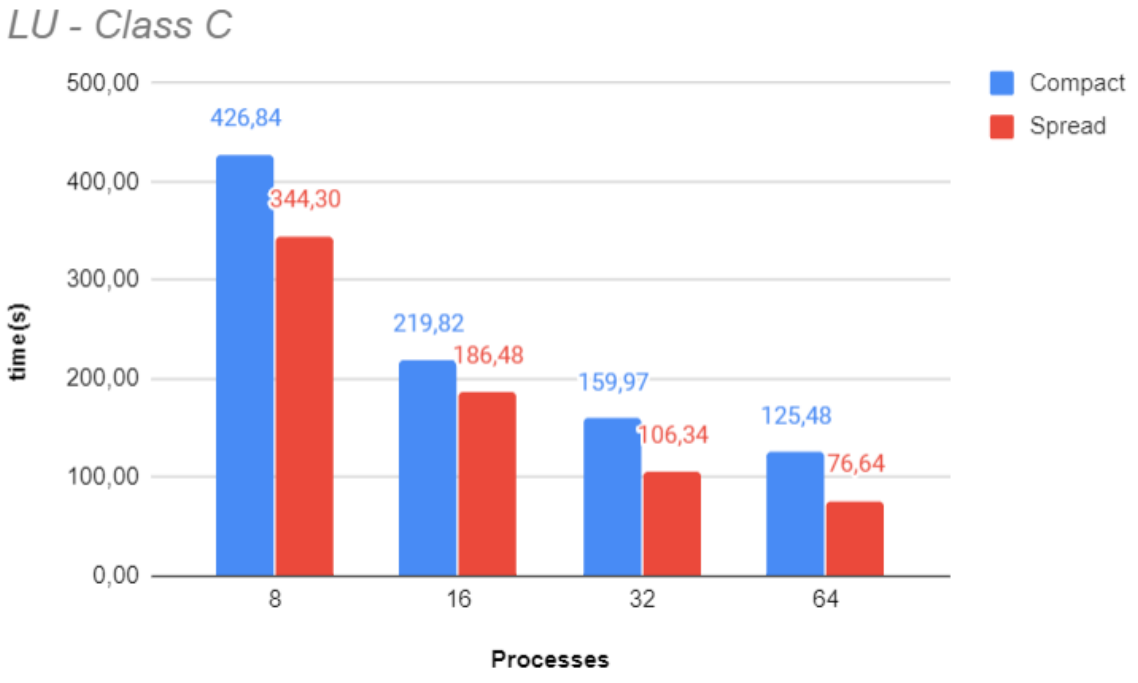
Γραφικές 61: IS - CLASS C



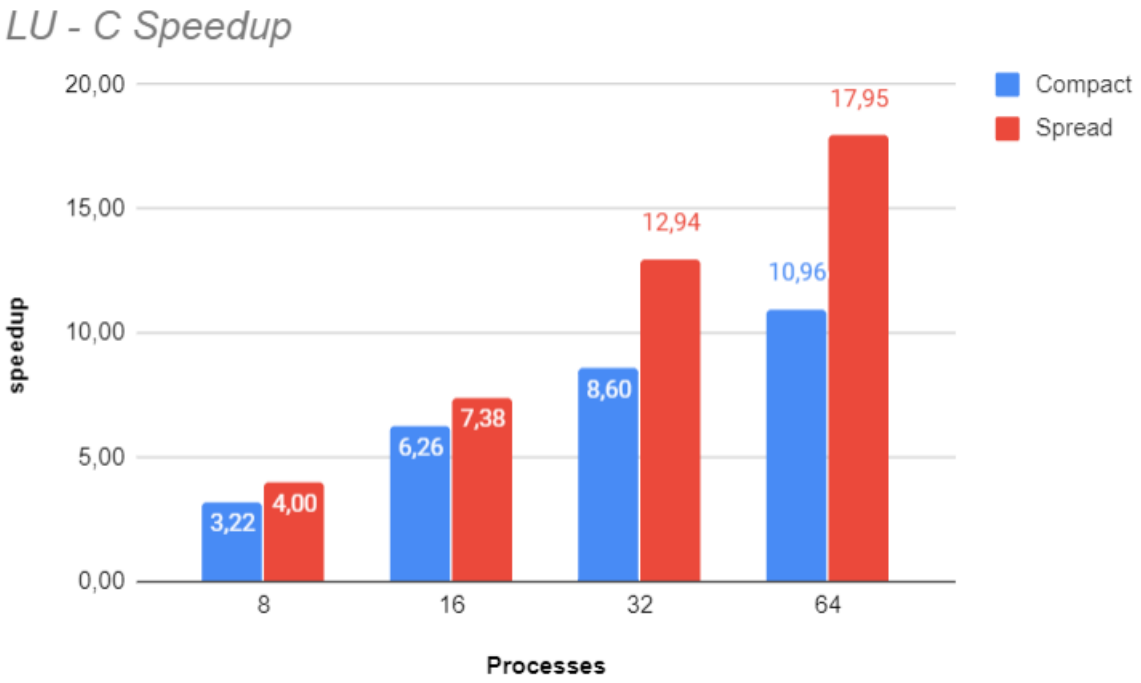
Γραφικές 62: IS - C Speedup



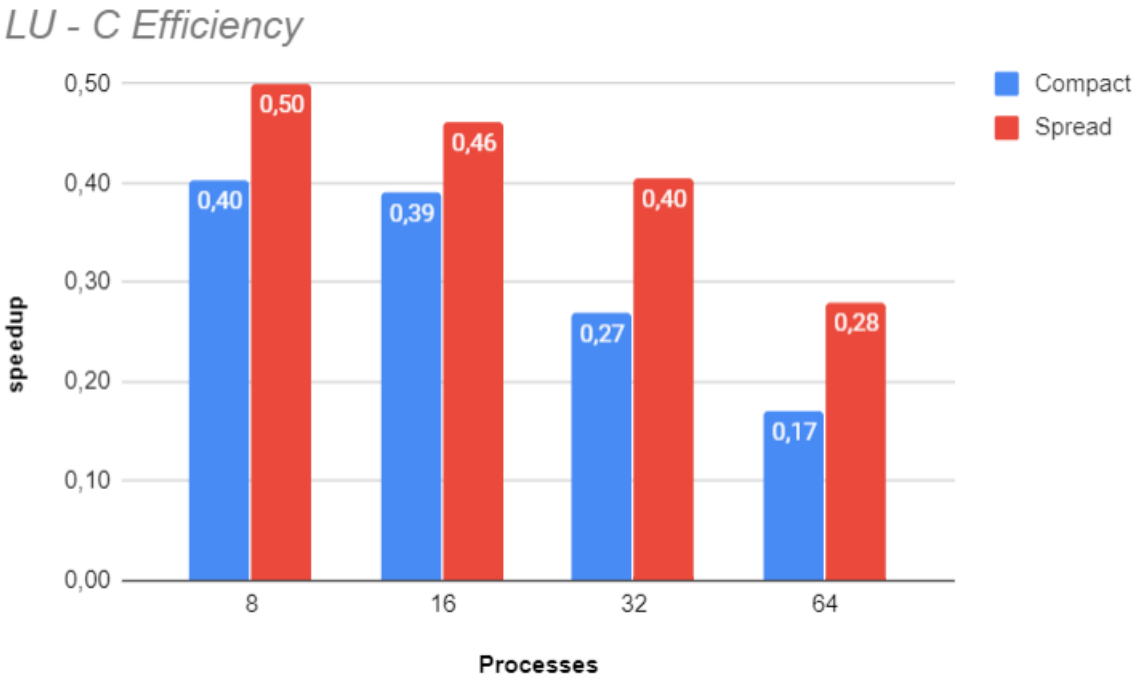
Γραφικές 63: IS - C Efficiency



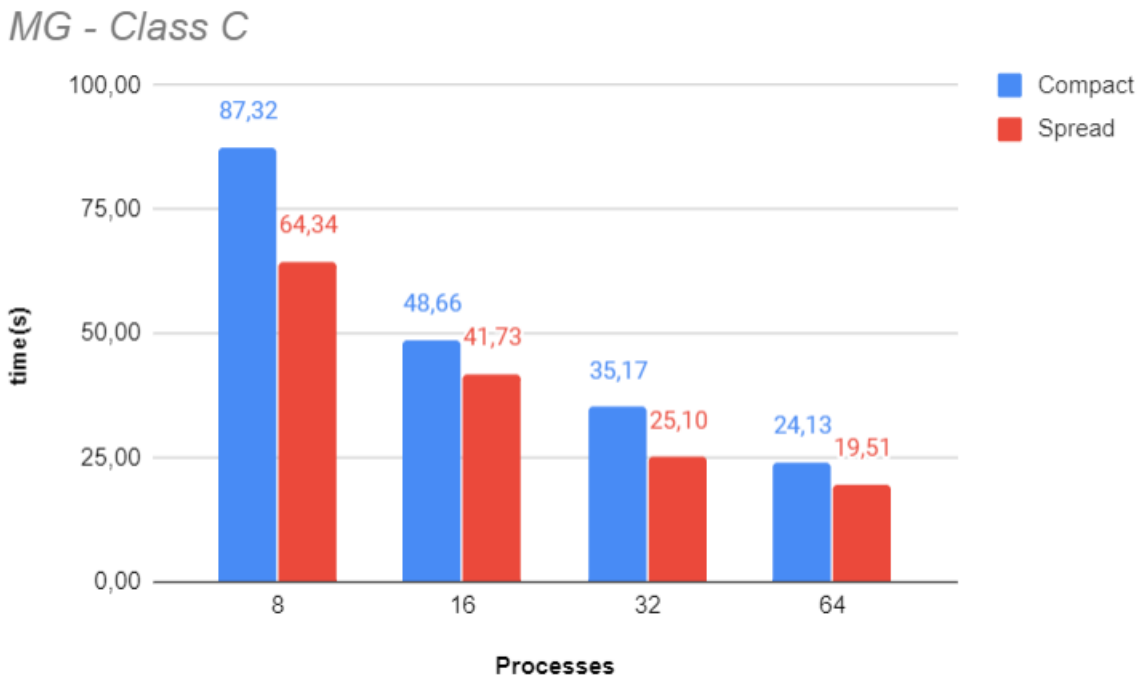
Γραφικές 64: LU - CLASS C



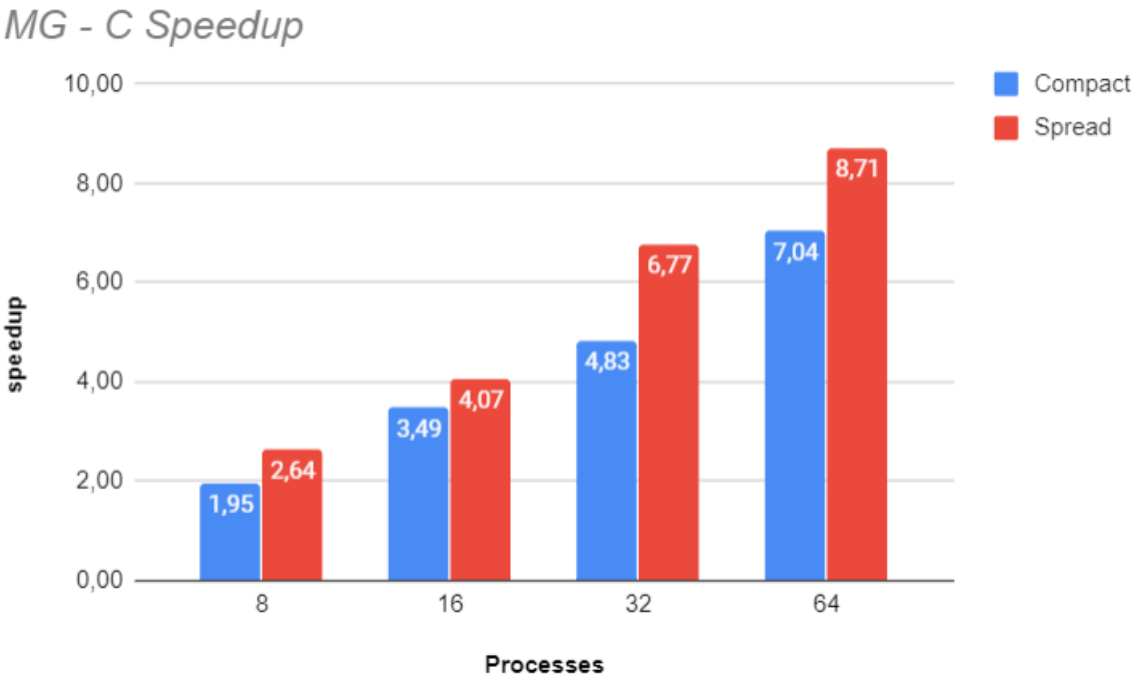
Γραφικές 65: LU - C Speedup



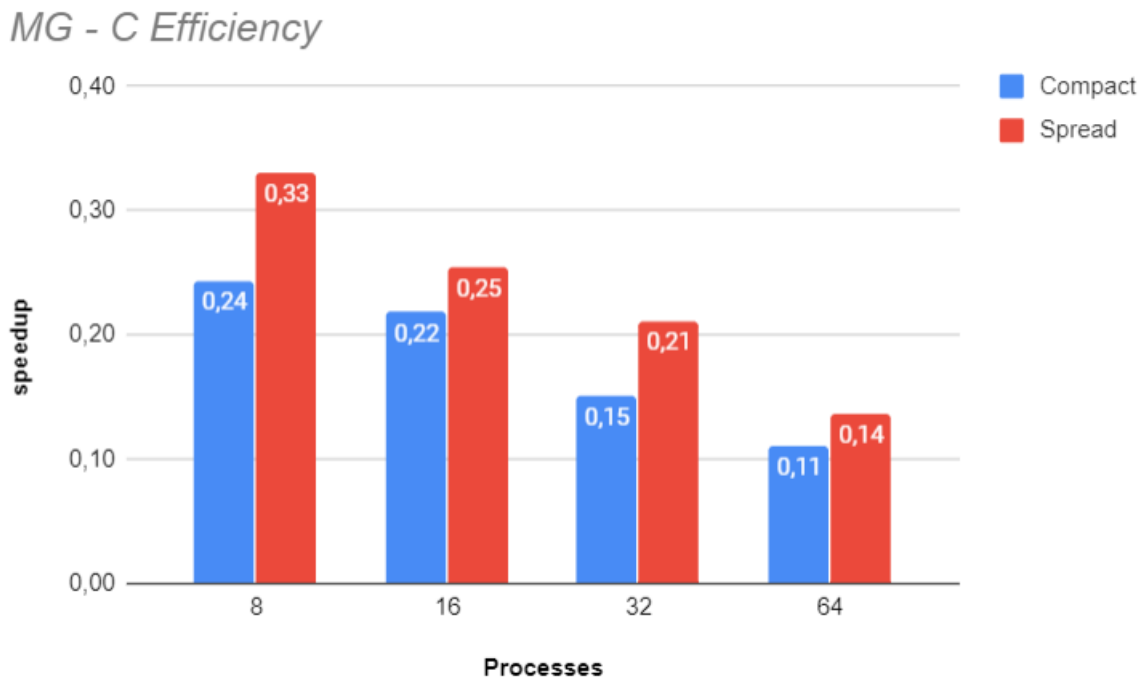
Γραφικές 66: LU - C Efficiency



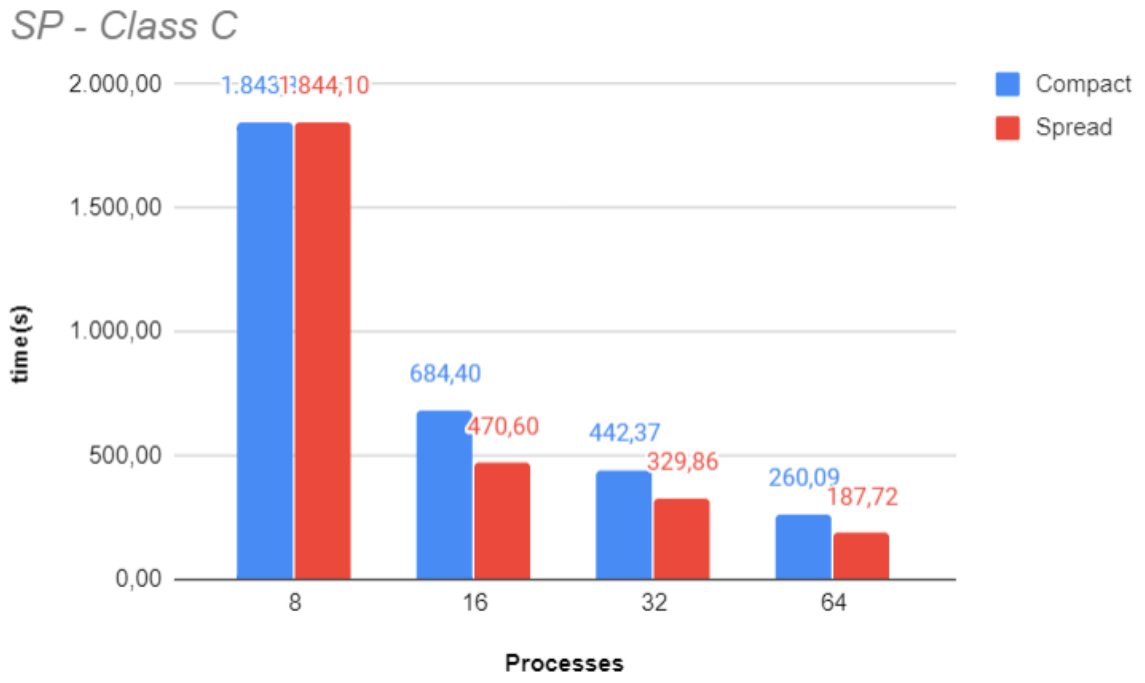
Γραφικές 67: MG - CLASS C



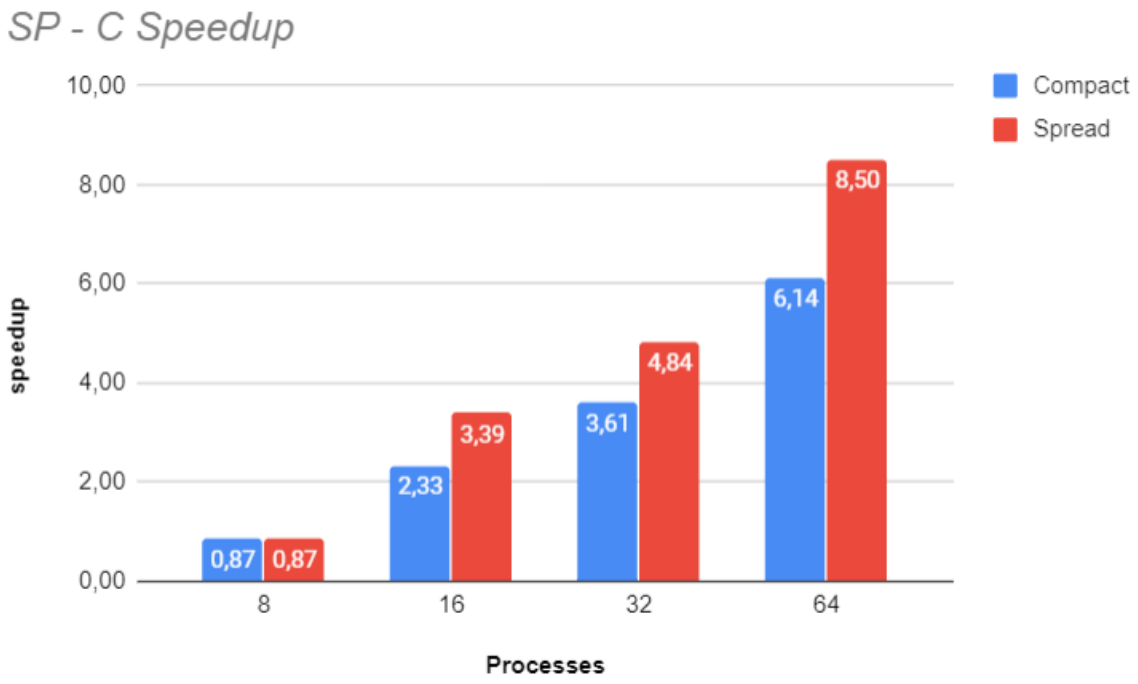
Γραφικές 68: MG - C Speedup



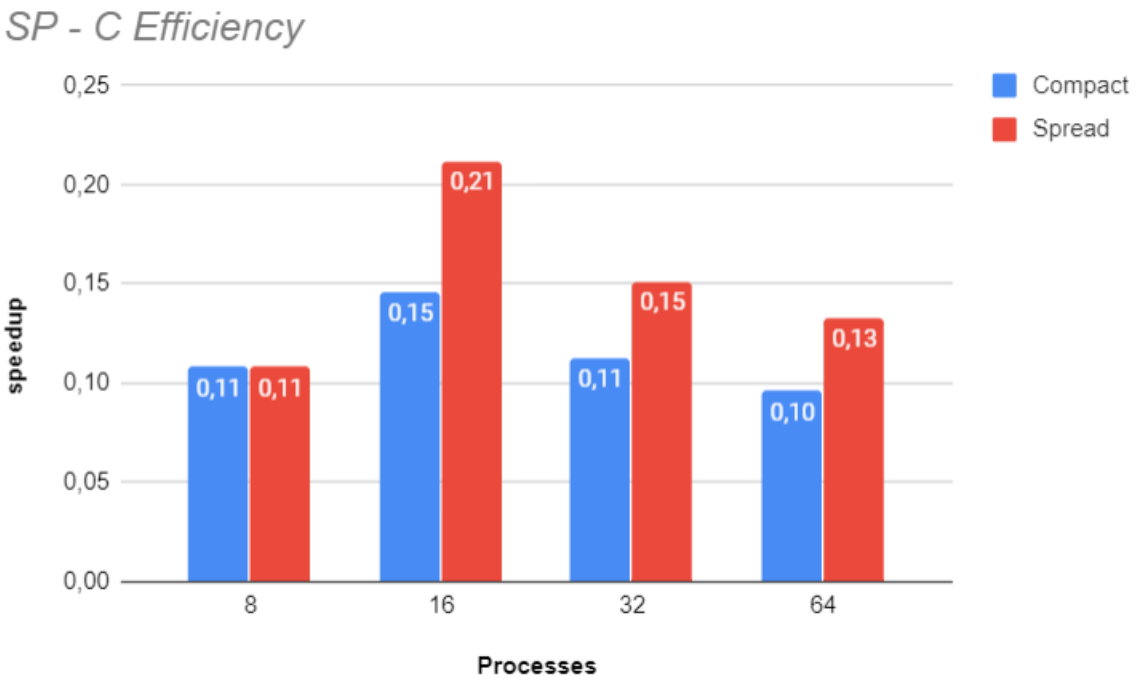
Γραφικές 69: MG - C Efficiency



Γραφικές 70: SP - CLASS C

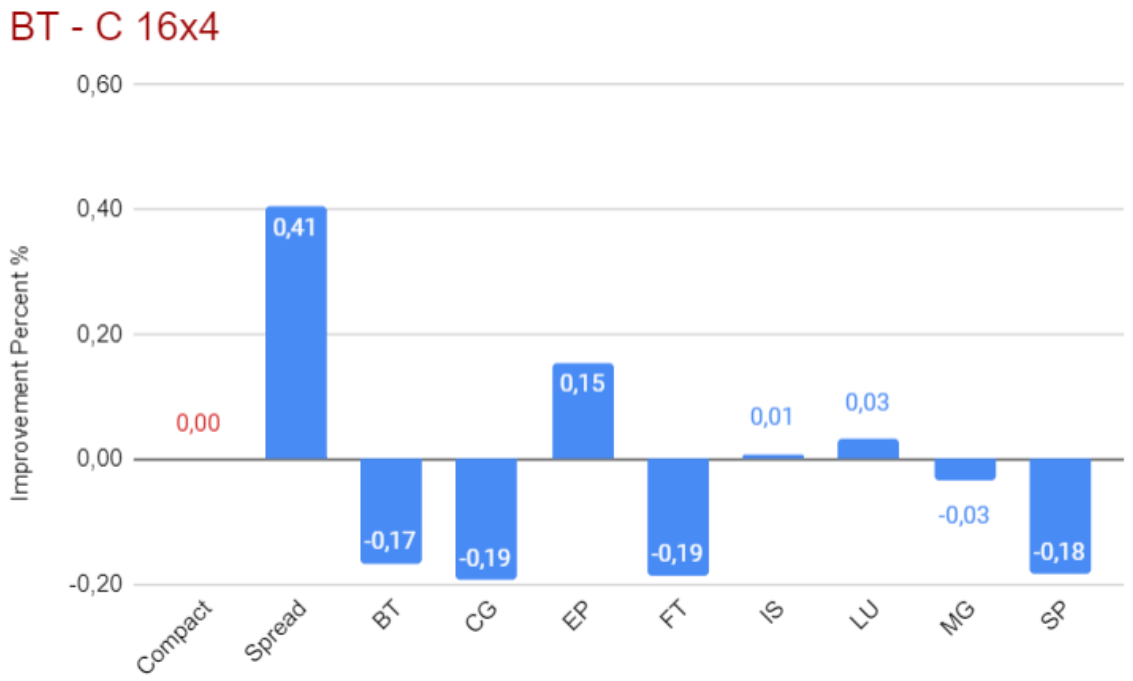


Γραφικές 71: SP - C Speedup

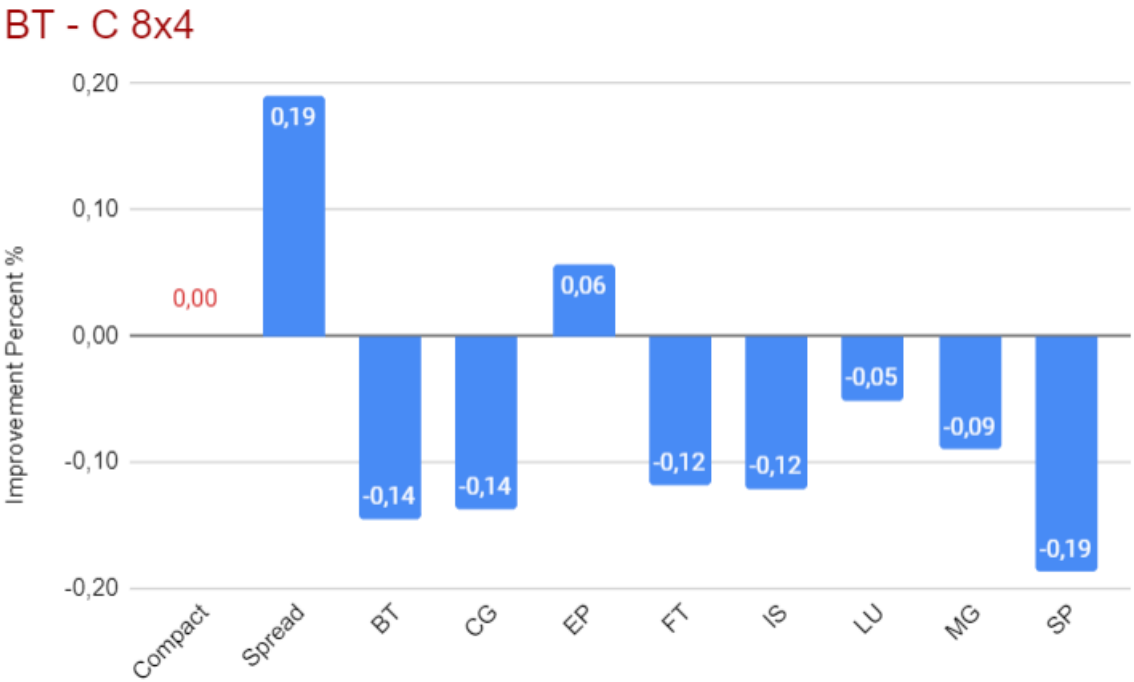


Γραφικές 72: SP - C Efficiency

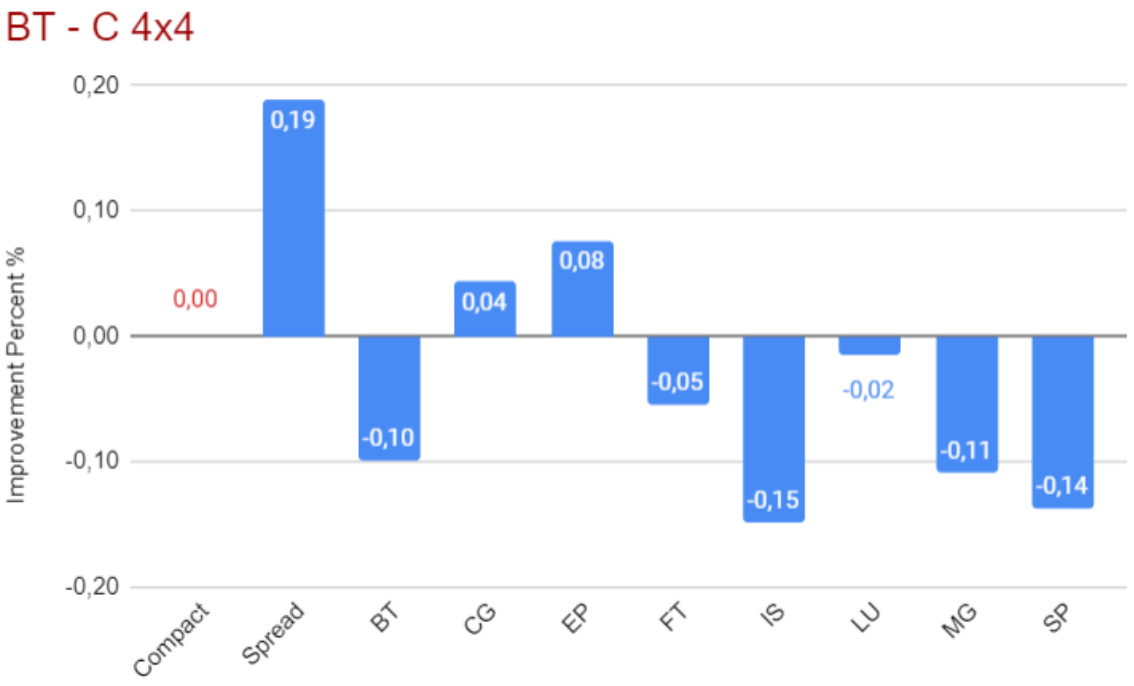
5.3.4 Mixed C



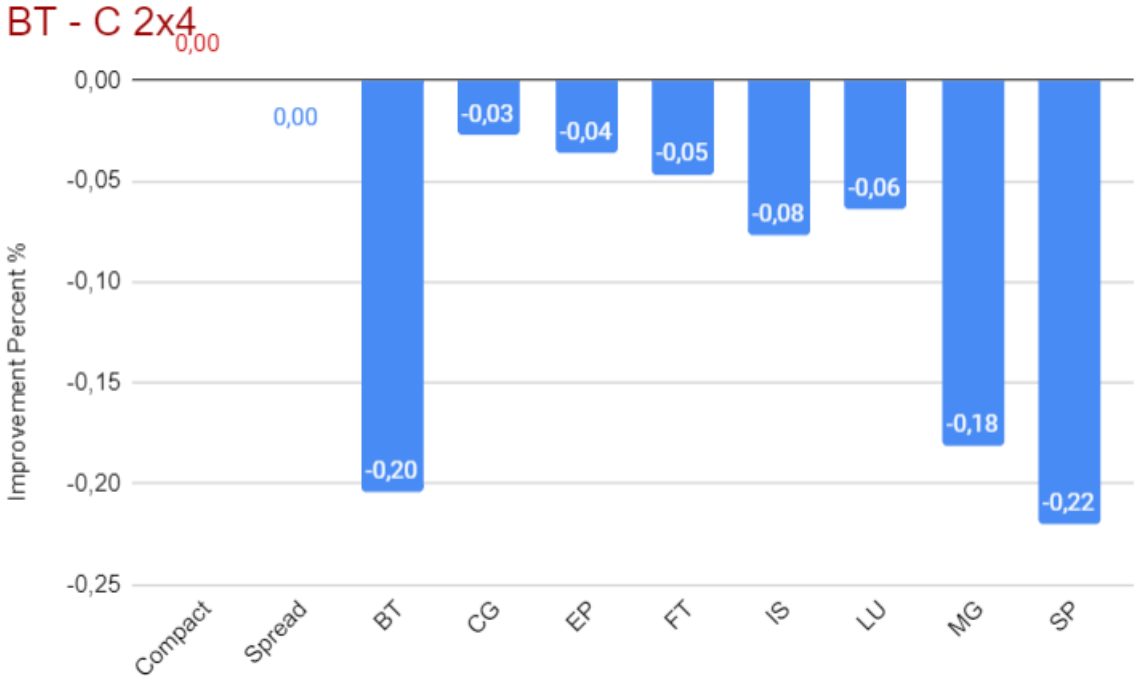
Γραφικές 73: BT - C 16x4



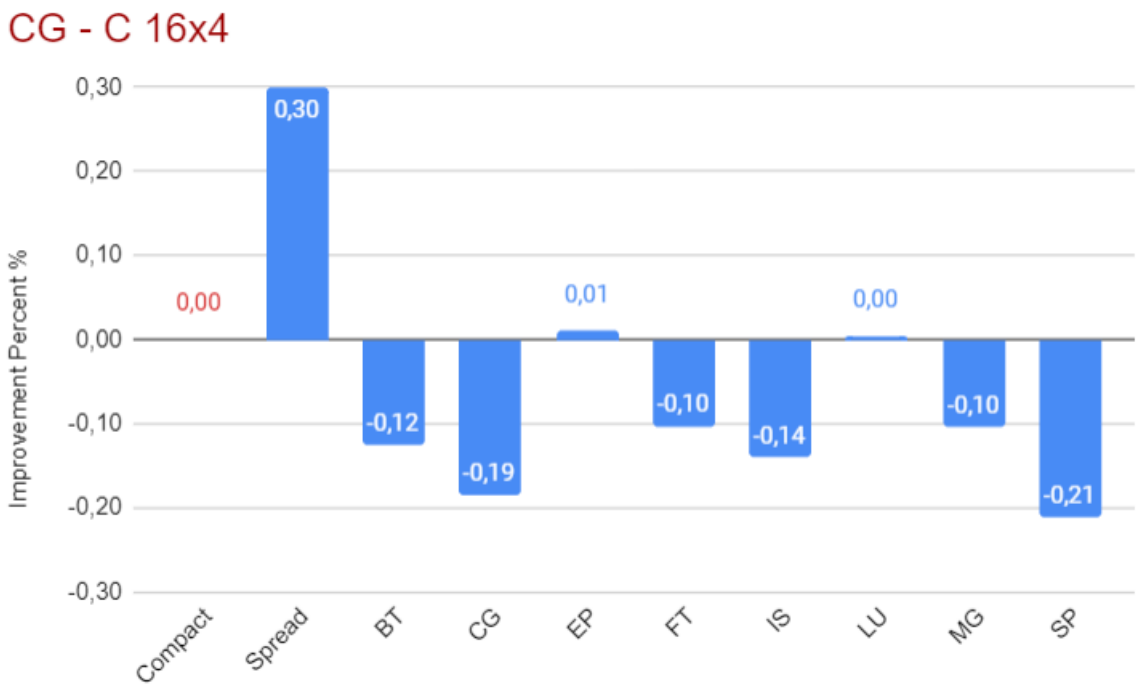
Γραφικές 74: BT - C 8x4



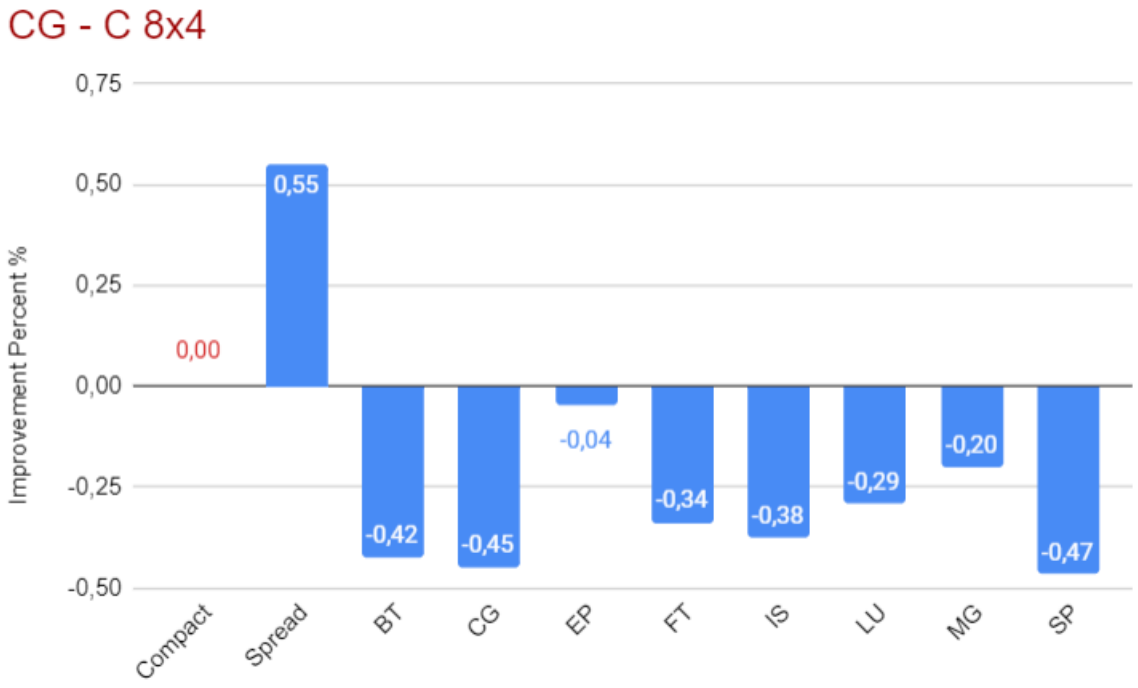
Γραφικές 75: BT - C 4x4



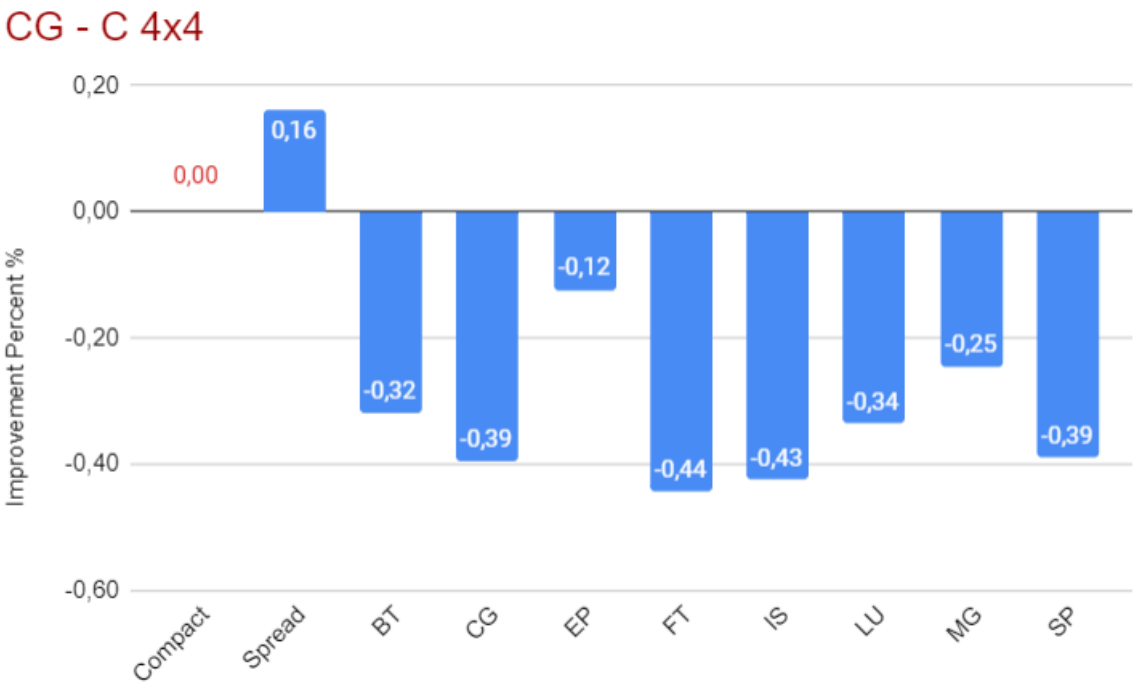
Γραφικές 76: BT - C 2x4



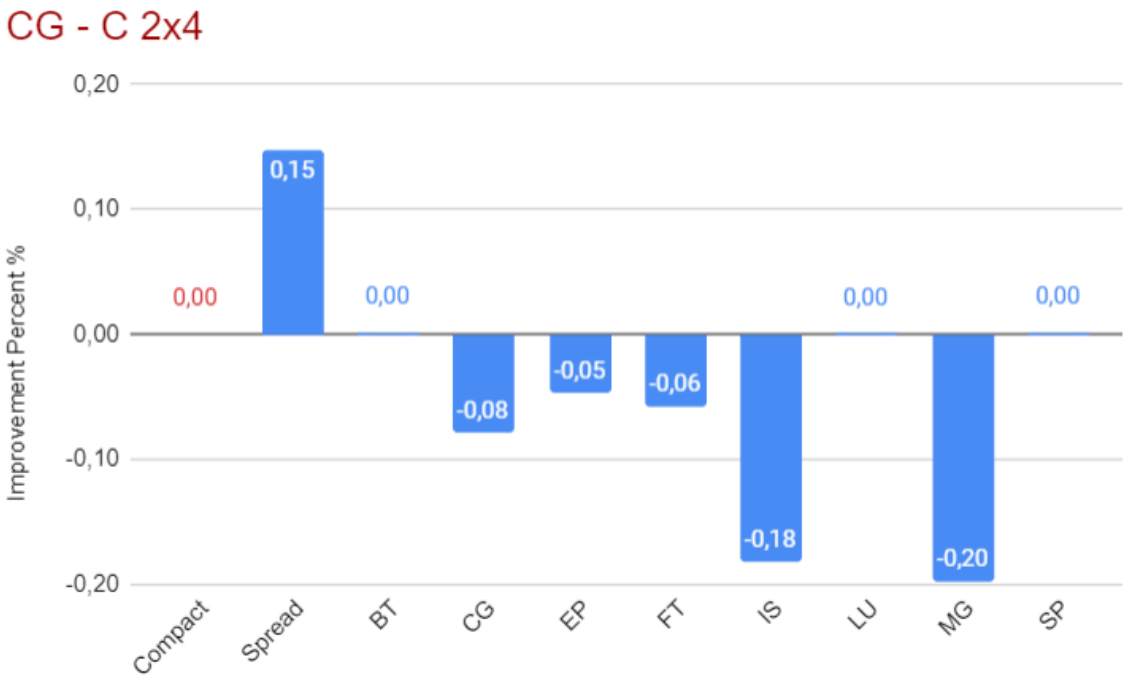
Γραφικές 77: CG - C 16x4



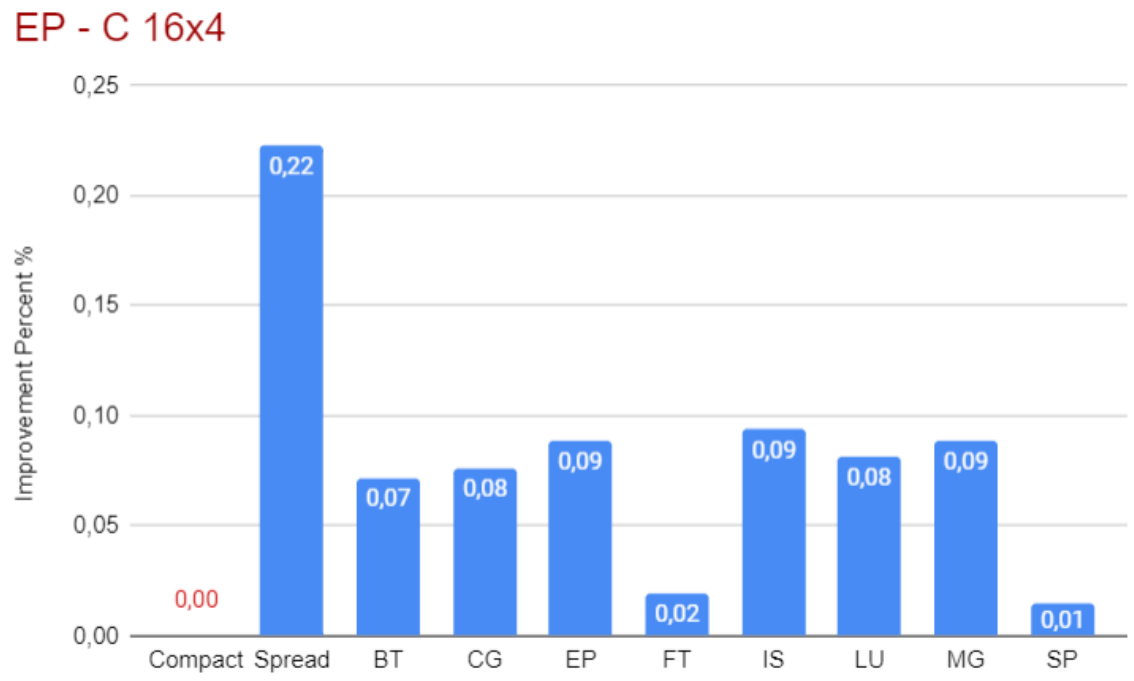
Γραφικές 78: CG - C 8x4



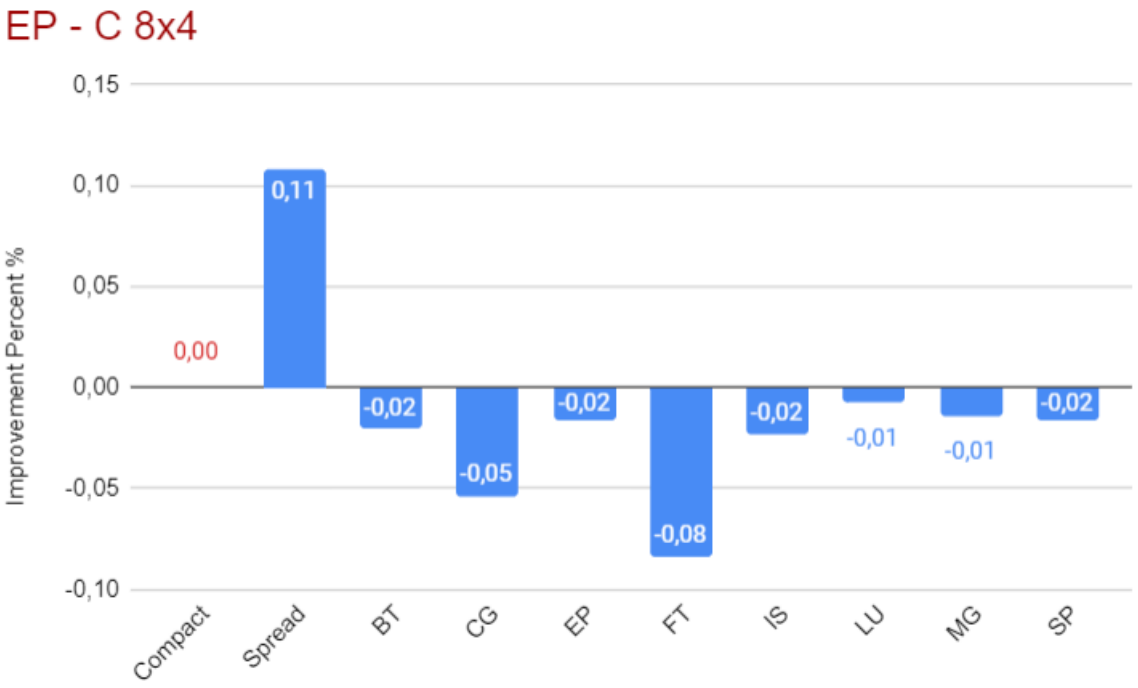
Γραφικές 79: CG - C 4x4



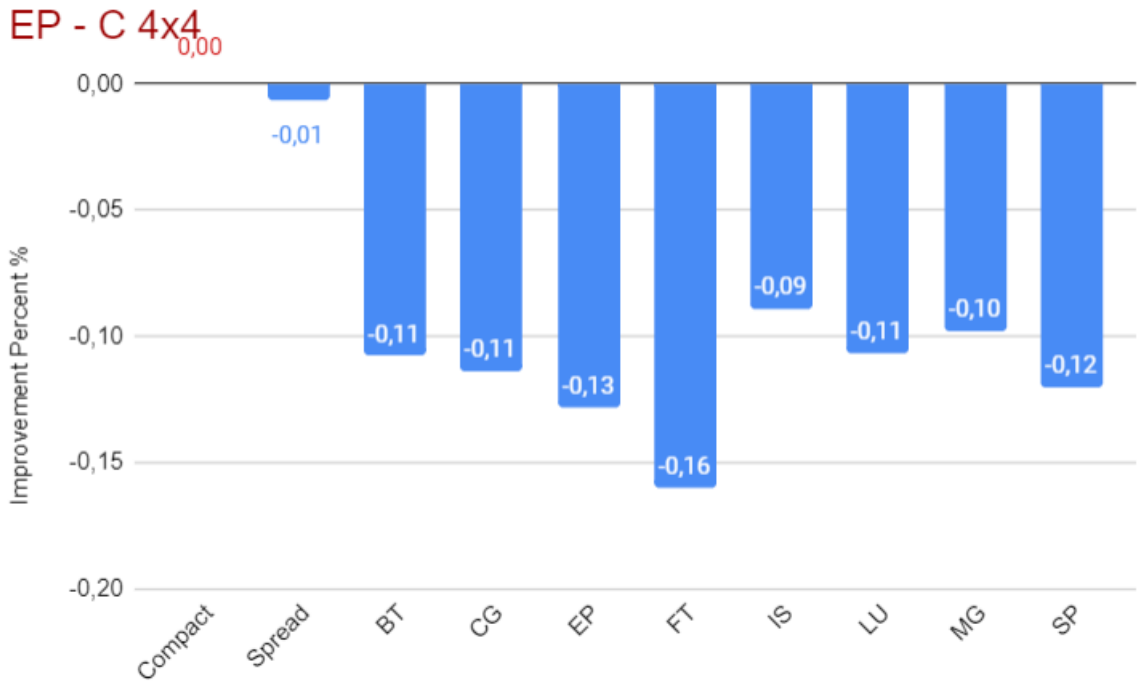
Γραφικές 80: CG - C 2x4



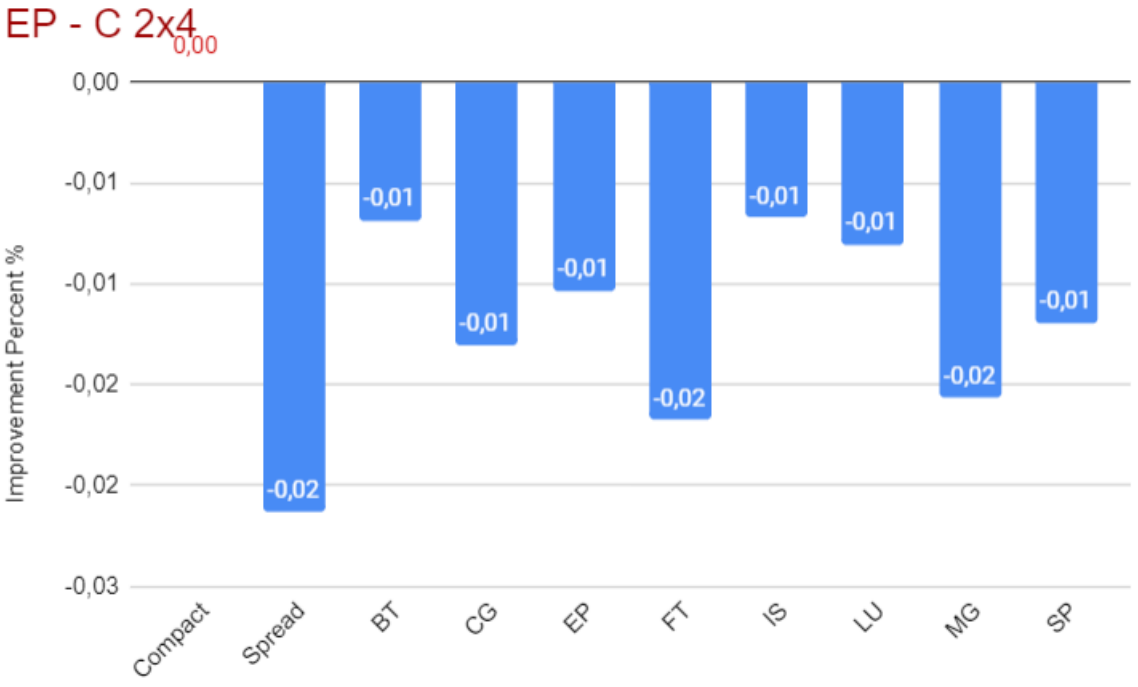
Γραφικές 81: EP - C 16x4



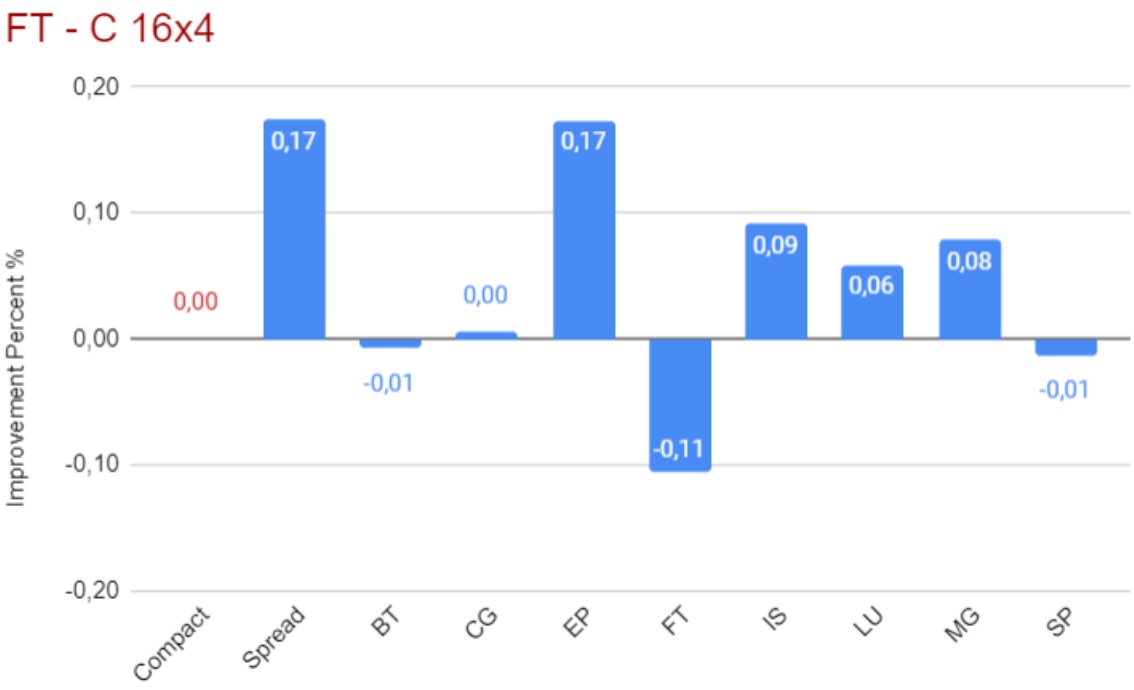
Γραφικές 82: EP - C 8x4



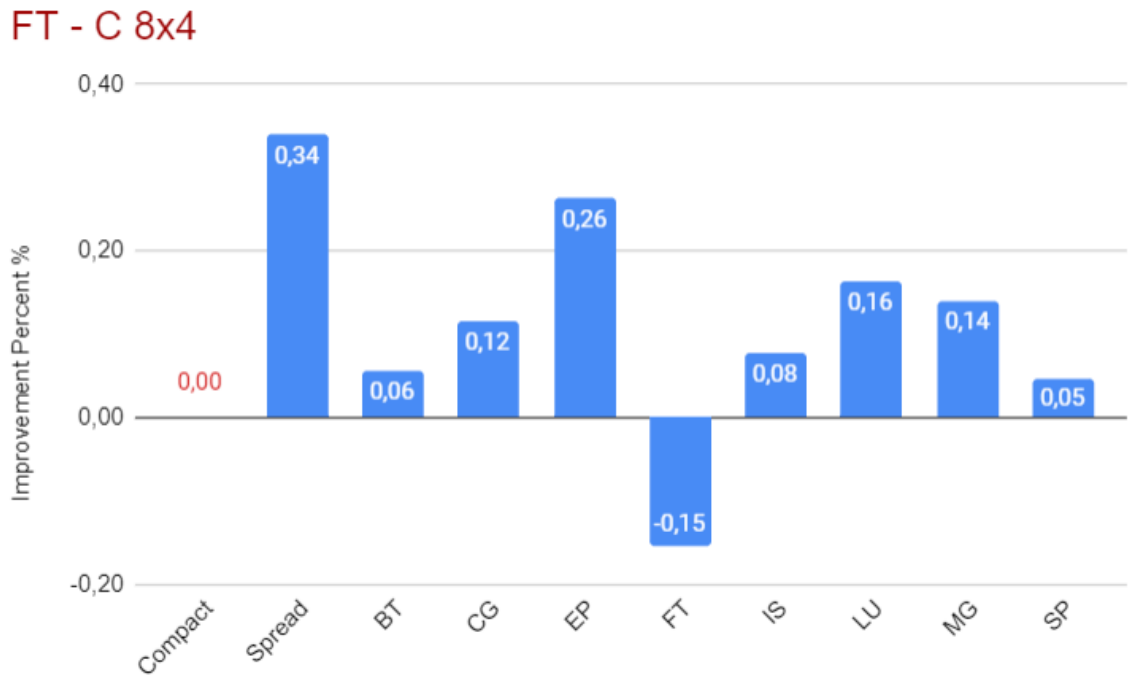
Γραφικές 83: EP - C 4x4



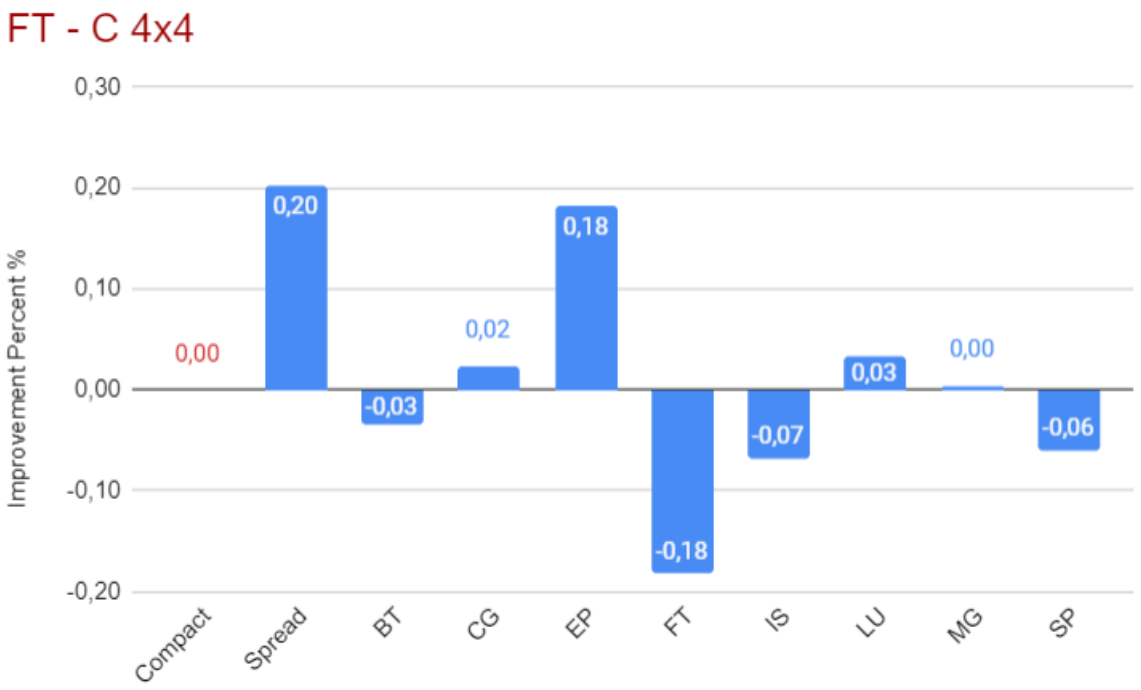
Γραφικές 84: EP - C 2x4



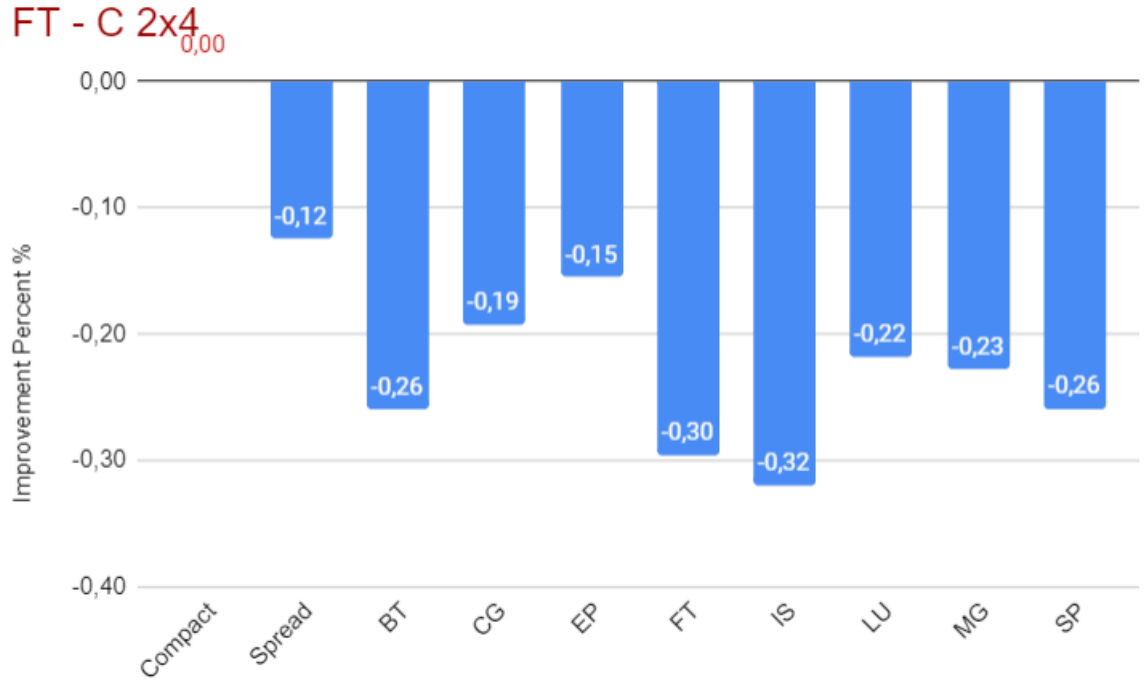
Γραφικές 85: FT - C 16x4



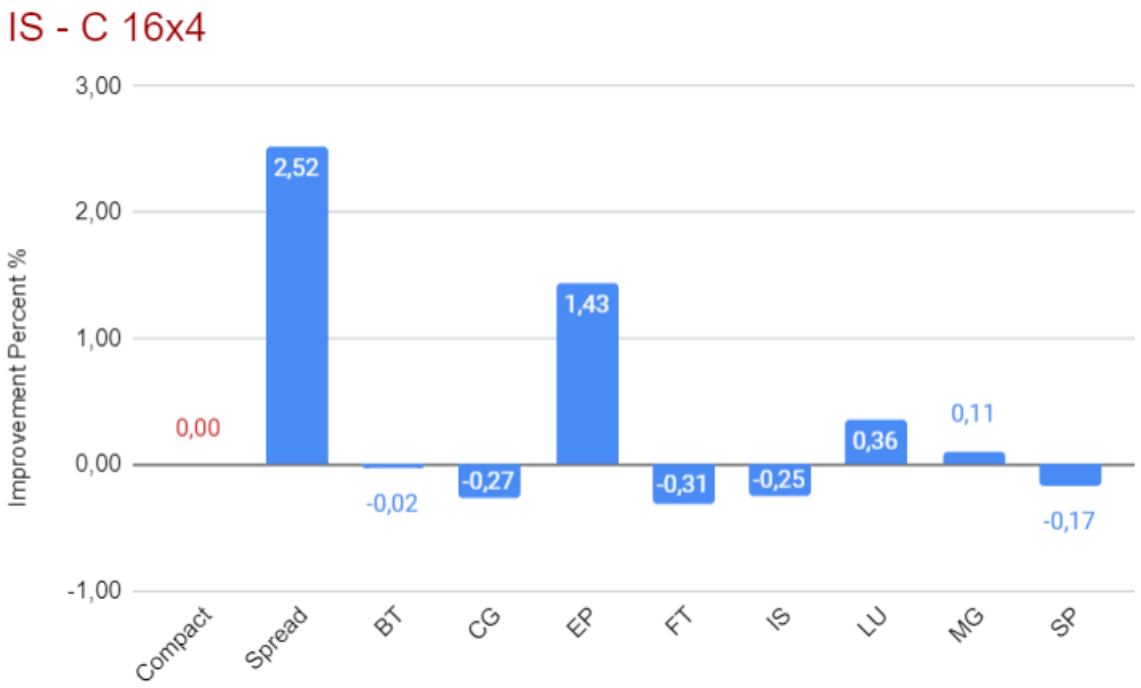
Γραφικές 86: FT - C 8x4



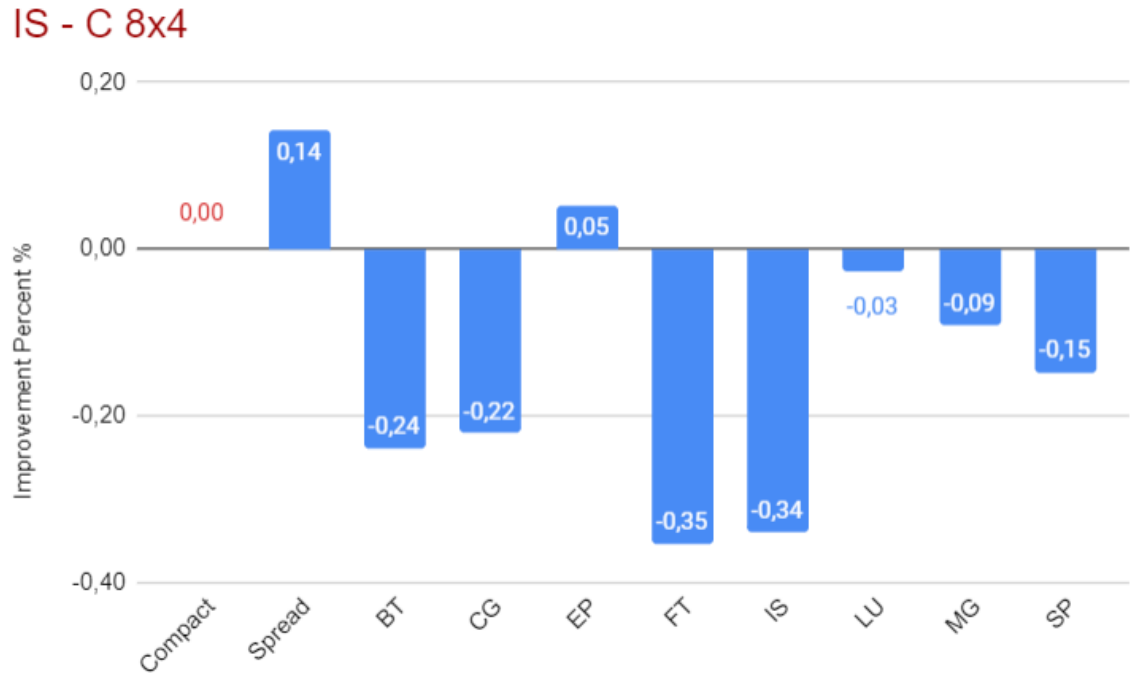
Γραφικές 87: FT - C 4x4



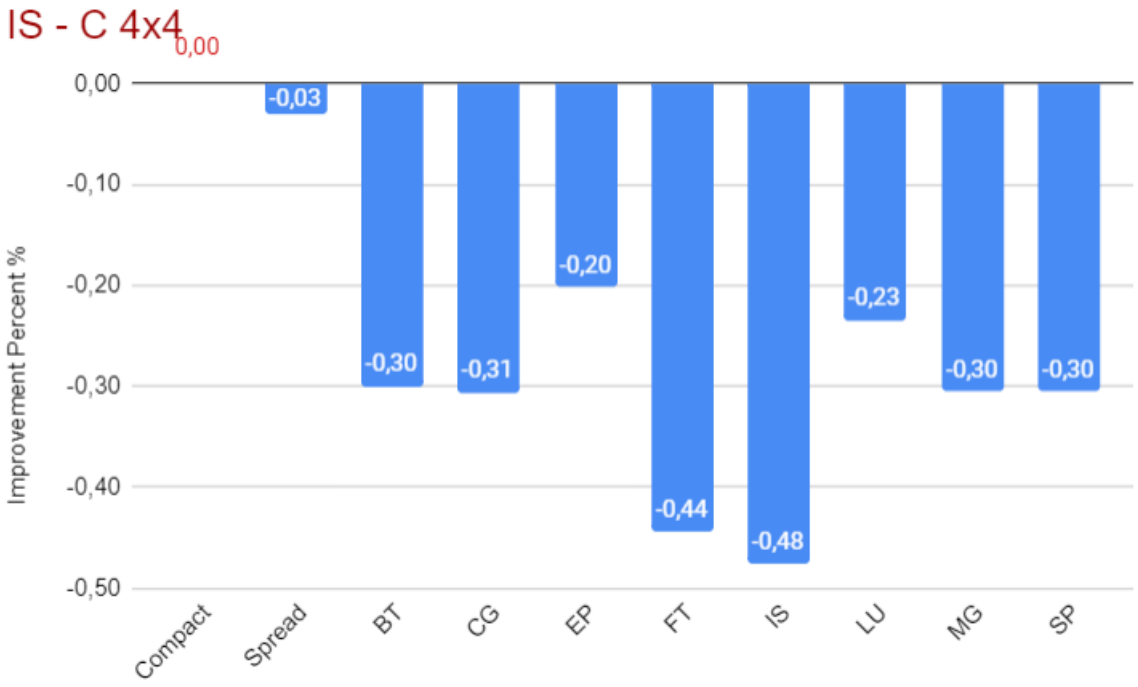
Γραφικές 88: FT - C 2x4



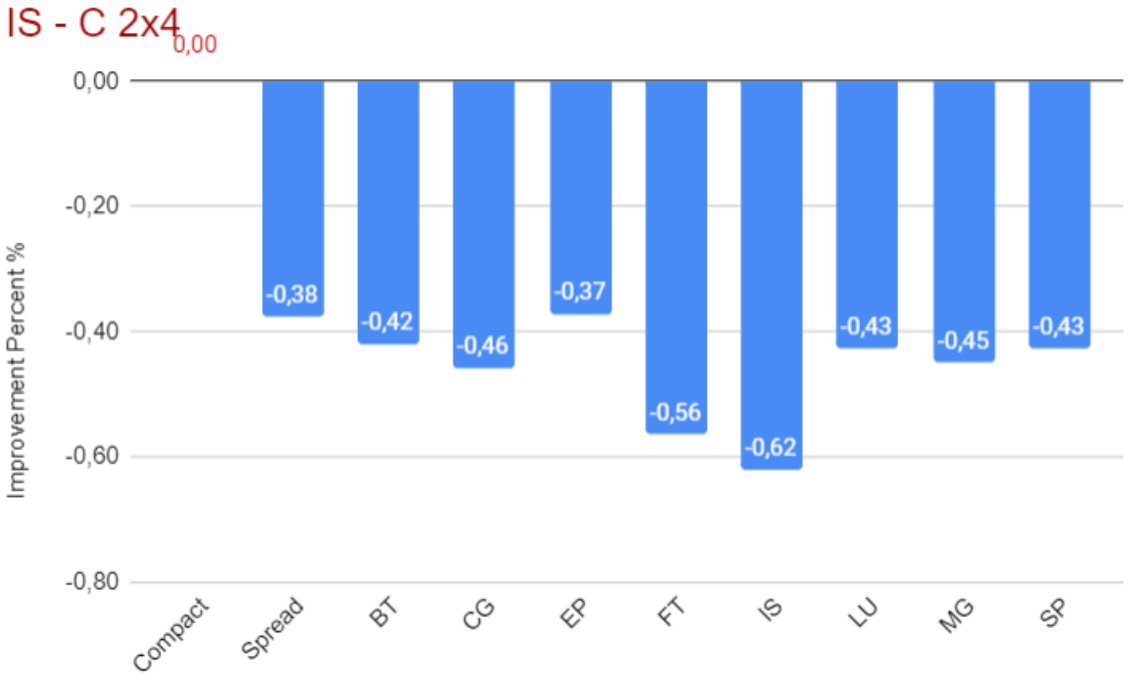
Γραφικές 89: IS - C 16x4



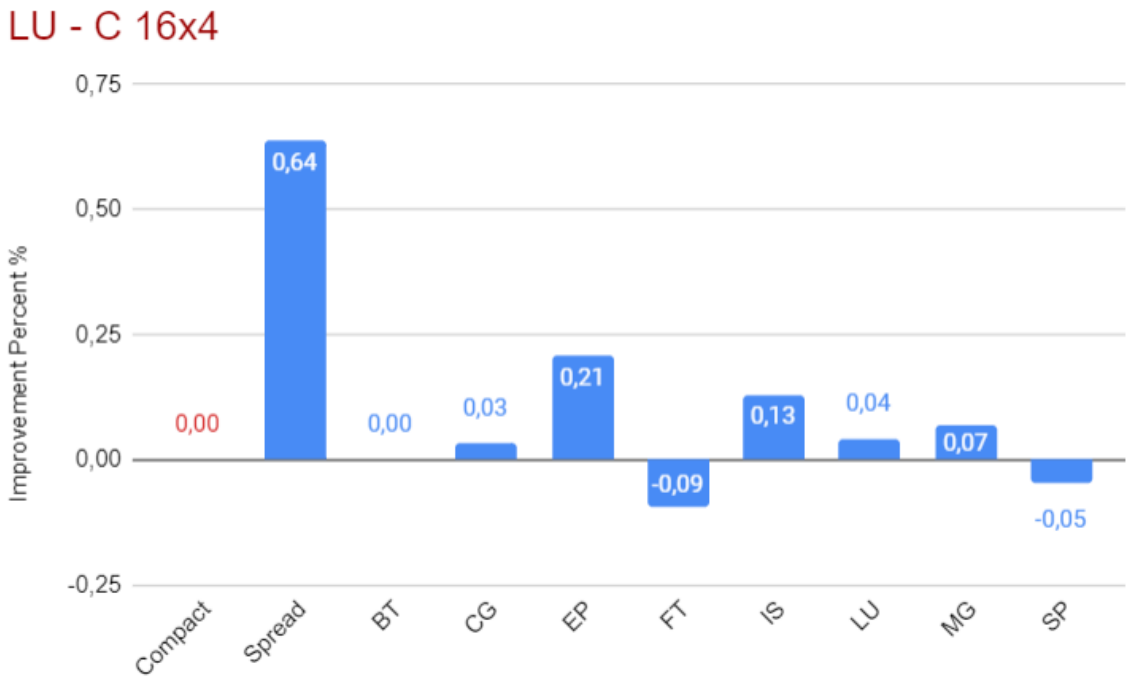
Γραφικές 90: IS - C 8x4



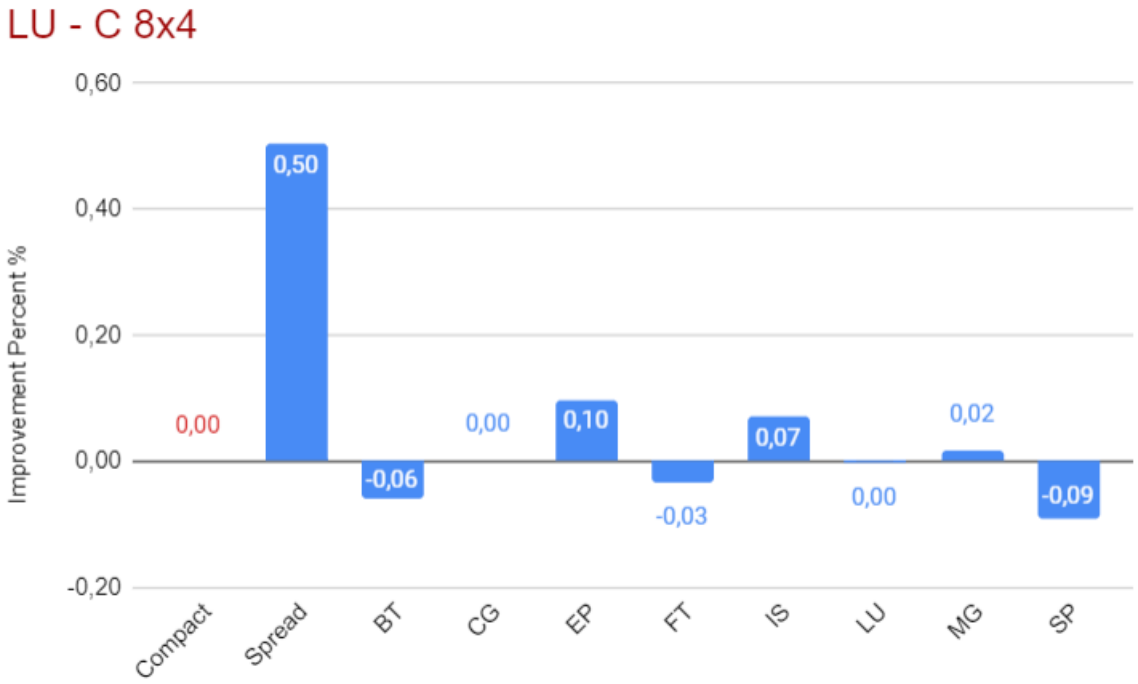
Γραφικές 91: IS - C 4x4



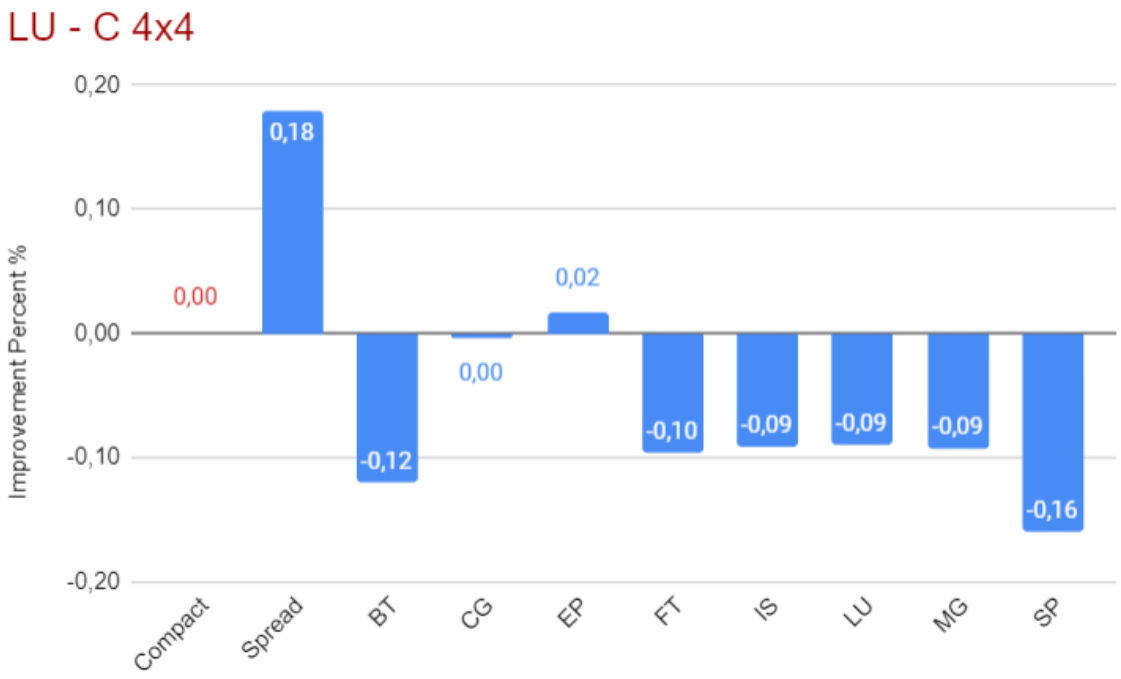
Γραφικές 92: IS - C 2x4



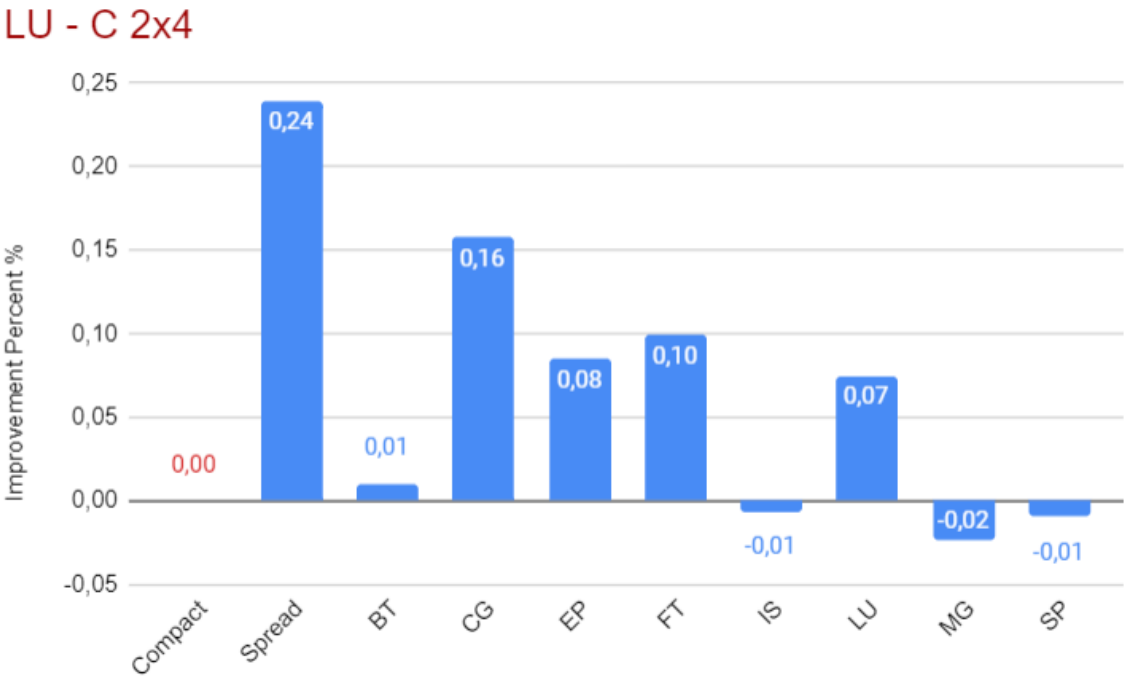
Γραφικές 93: LU - C 16x4



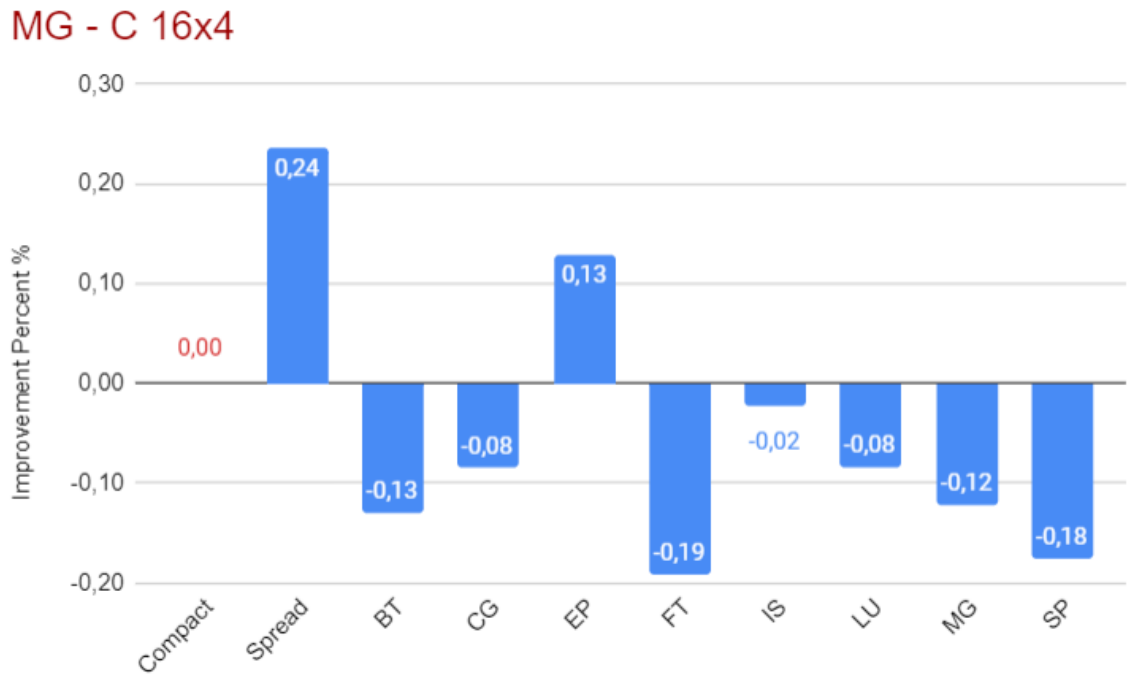
Γραφικές 94: LU - C 8x4



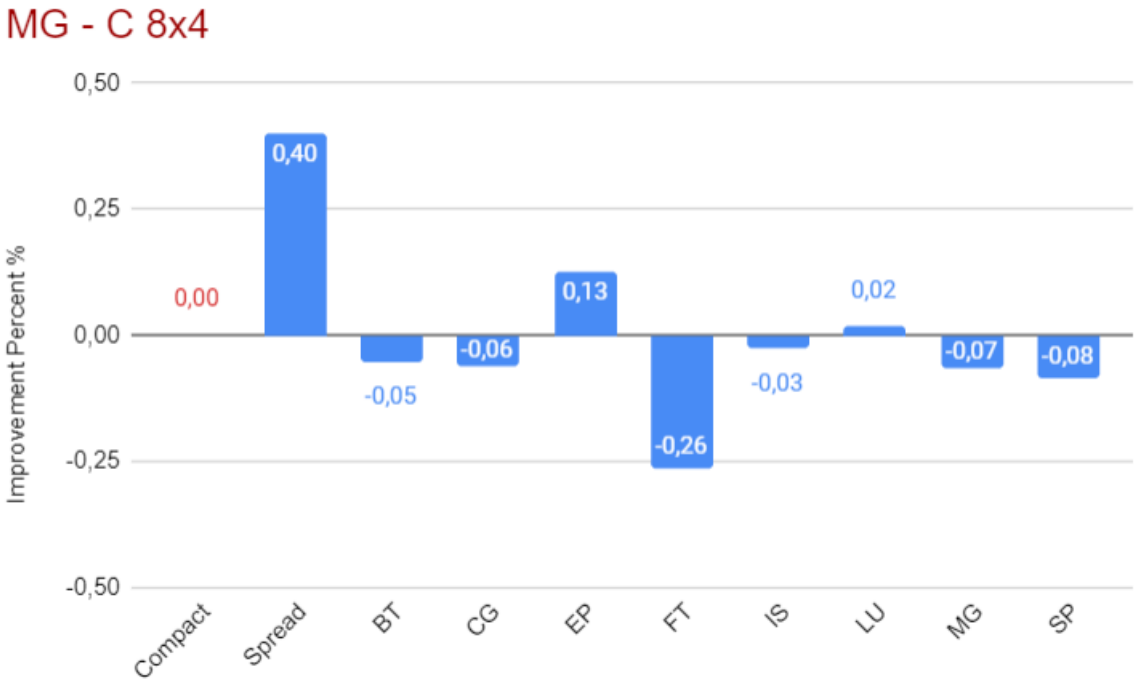
Γραφικές 95: LU - C 4x4



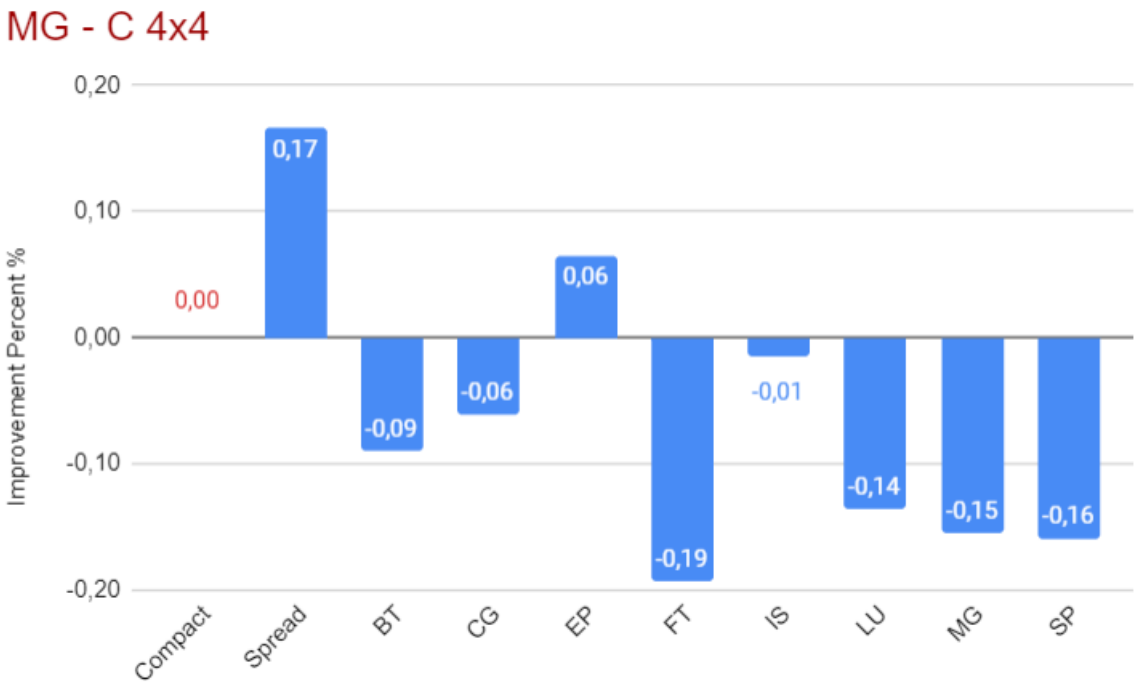
Γραφικές 96: LU - C 2x4



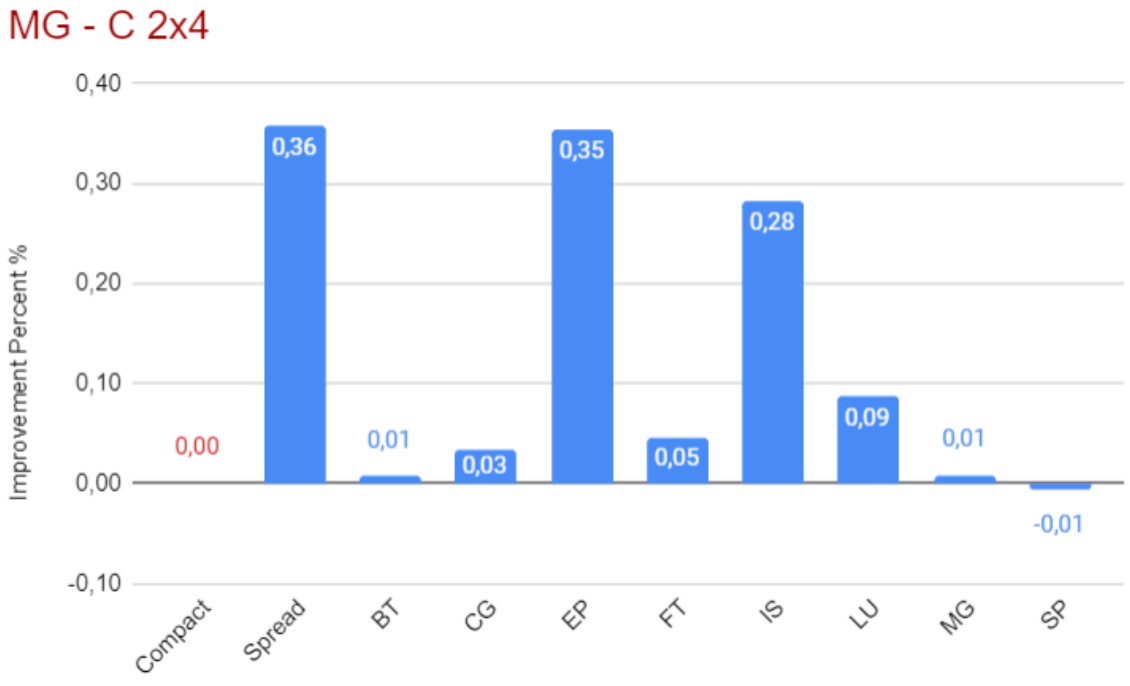
Γραφικές 97: MG - C 16x4



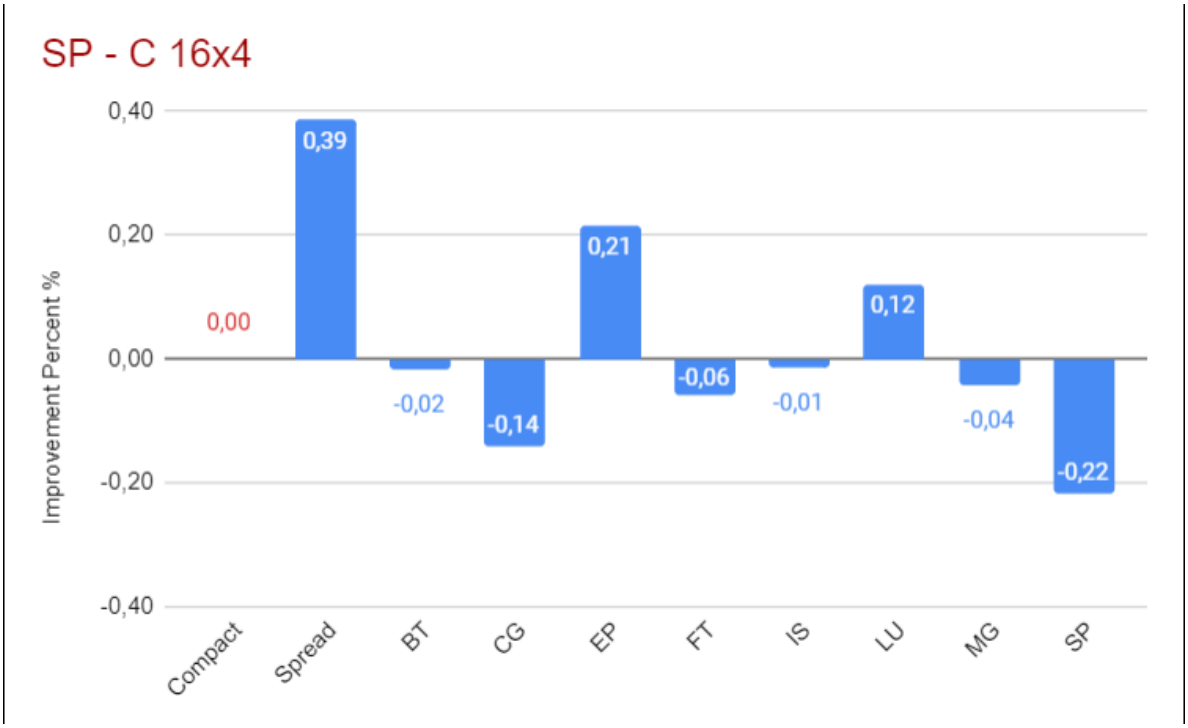
Γραφικές 98: MG - C 8x4



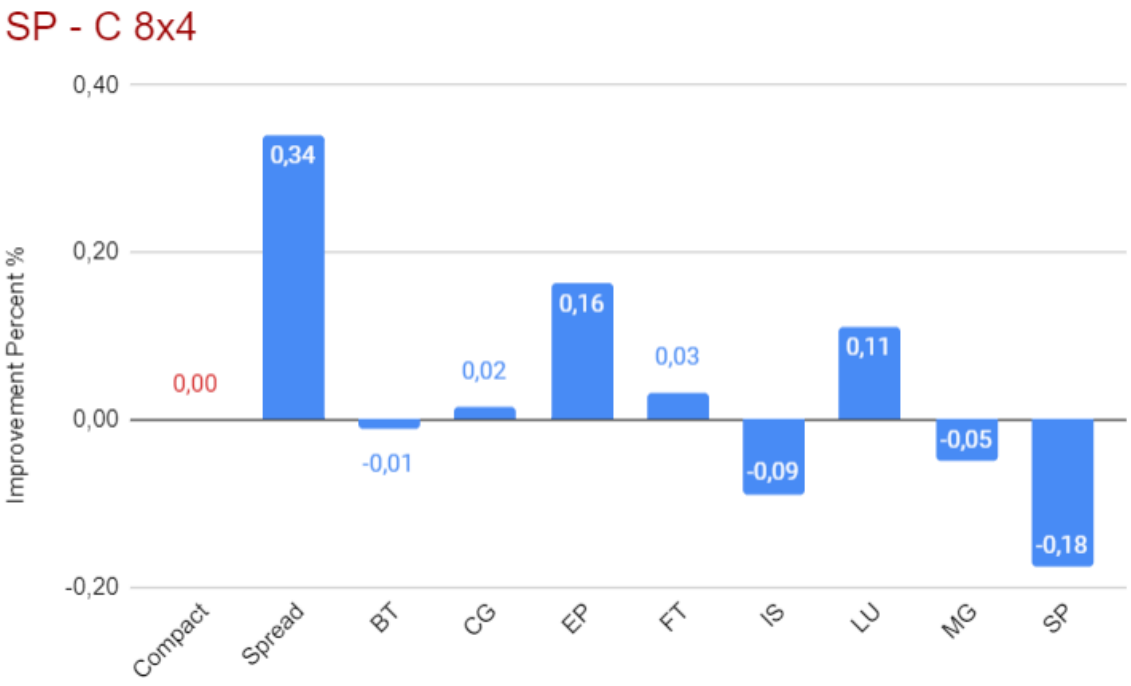
Γραφικές 99: MG - C 4x4



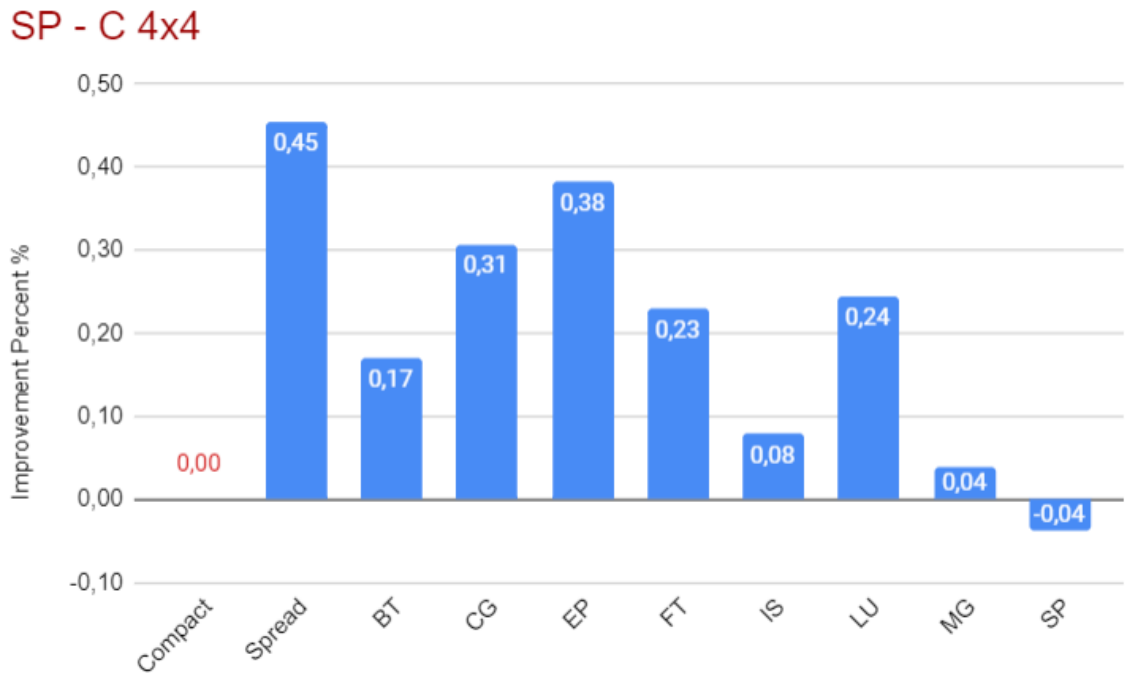
Γραφικές 100: MG - C 2x4



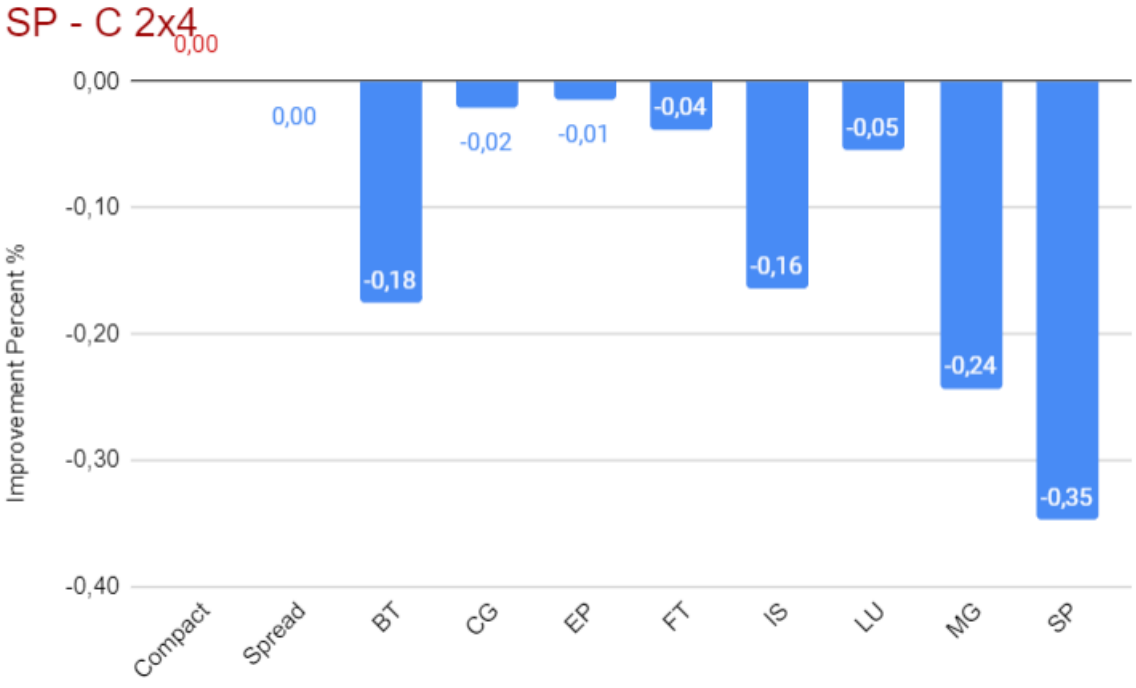
Γραφικές 101: SP - C 16x4



Γραφικές 102: SP - C 8x4



Γραφικές 103: SP - C 4x4



Γραφικές 104: SP - C 2x4

5.4 Σχολιασμός Αποτελεσμάτων

Στην προηγούμενη ενότητα παρουσιάστηκαν εκτενώς τα αποτελέσματα από τα NPB προβλήματα για τις κλάσεις A, B, C. Στις τρεις αυτές κλάσεις φαίνεται πως τα προβλήματα δεν έχουν τη συμπεριφορά που αναμένουμε.

Το πρόβλημα EP είναι το μόνο που βελτιώνεται σωστά και παρουσιάζει γραμμική επιτάχυνση. Αυτό συμβαίνει καθώς το πρόβλημα αυτό είναι αποκλειστικά compute intensive. Συνεπώς, όσο αυξάνουν οι πυρήνες τόσο μειώνεται ο χρόνος εκτέλεσης του προγράμματος. Αυτό επιτυγχάνεται από το EP όσο αυξάνουν οι πυρήνες, μέχρι και για 64 πυρήνες που είναι το μέγιστο που χρησιμοποιείται στην παρούσα εργασία.

Τα υπόλοιπα προβλήματα δεν παρουσιάζουν γραμμική επιτάχυνση με την αύξηση του αριθμού των πυρήνων, που είναι ο στόχος σε κάθε παράλληλο πρόγραμμα. Αυτό συμβαίνει γιατί τα προβλήματα αυτά, το καθένα σε διαφορετικό βαθμό, είναι memory και communication intensive. Το memory intensive δεν παρεμποδίζει την απόδοση τους. Το communication intensive όμως, όσο τα προβλήματα εξαπλώνονται σε περισσότερα μηχανήματα και περισσότερους πυρήνες, παρεμποδίζει το συνολικό χρόνο εκτέλεσης.

Χαρακτηριστικό παράδειγμα αποτελεί το IS το οποίο είναι ένα πρόβλημα integer sort. Απαιτεί αρκετή επικοινωνία μεταξύ των πυρήνων στους οποίους κατανέμεται. Αυτό έχει ως αποτέλεσμα, όσο αυξάνουν οι πυρήνες το πρόγραμμα να γίνεται πιο αργό. Μπορεί η συνολική εργασία να μοιράζεται σε περισσότερους πυρήνες και να γίνεται γρηγορότερα αλλά αν η επικοινωνία μεταξύ τους υπερτερεί σε σχέση με τον χρόνο επεξεργασίας όπως γίνεται στο IS πρόβλημα, τότε η αύξηση των πυρήνων οδηγεί σε πιο αργή συνολικά εκτέλεση. Αυτό έχει σαν αποτέλεσμα, στην κλάση A, η οποία έχει μικρά μεγέθη να έχει το μεγαλύτερο πρόβλημα. Όσο η κλάση μεγαλώνει, το πρόβλημα φαίνεται μικρότερο αφού αυξάνοντας τους πυρήνες, μπορεί να αυξάνεται το communication αλλά οι περισσότερες πράξεις της μεγαλύτερης κλάσης κατανέμονται καλύτερα στους διαθέσιμους πυρήνες.

Παρόμοια συμπεριφορά εμφανίζουν όλα τα υπόλοιπα προβλήματα. Στην κλάση A δηλαδή παρουσιάζουν την μικρότερη βελτίωση - κάποια ίσως και χειροτέρευση - στην κλάση B λίγο καλύτερη βελτίωση – μικρότερη χειροτέρευση – και στην κλάση C ακόμα καλύτερα.

Χαρακτηριστικό παράδειγμα αποτελεί το πρόβλημα BT. Στην κλάση A παρατηρούμε όπως φαίνεται και στο σχήμα 1 πως αυξάνοντας τους πυρήνες από 16 σε 32, ο συνολικός χρόνος εκτέλεσης δεν μειώνεται αλλά αντιθέτως αυξάνεται. Με αύξηση από 32 σε 64 πάλι δεν παρατηρείται βελτίωση. Στην κλάση B του ίδιου προβλήματος (σχήμα 25) αυξάνοντας τους πυρήνες από 8 σε 16 παρατηρείται βελτίωση, από 16 σε 32 παρατηρείται περίπου ίδιος χρόνος και από 32 σε 64 μικρή βελτίωση. Μεγαλώνοντας και άλλο την κλάση του προβλήματος σε C, φαίνεται στο σχήμα 49 πως πλέον όσο αυξάνονται οι πυρήνες παρατηρείται συνολική βελτίωση και μείωση του συνολικού χρόνου. Η μείωση αυτή δεν είναι ανάλογη της αύξησης των πυρήνων ωστόσο το πρόβλημα BT φαίνεται πως έχει αρχίσει να λειτουργεί καλύτερα στο σύστημά μας στην κλάση C.

Η συμπεριφορά αυτή είναι κοινή για τα περισσότερα προβλήματα. Αυτό οφείλεται στο πως η κλάση A δεν χρειάζεται τόσους πυρήνες για την διεκπεραίωση της όσους δίνονται στην παρούσα εργασία. Το ίδιο και η κλάση B. Η κλάση C φαίνεται να ταιριάζει καλύτερα σε μια μελέτη κάποιων δεκάδων πυρήνων όπως αυτή. Τα προβλήματα βελτιώνονται όλα με εξαίρεση το IS και το CG, τα οποία φαίνεται πως χρειάζονται ακόμα μεγαλύτερη κλάση για να φανεί βελτίωση σε μεγάλο αριθμό πυρήνων. Ωστόσο η βελτίωση που παρουσιάζουν τα προβλήματα για 32 και 64 πυρήνες δεν είναι και πάλι αρκετά ικανοποιητική.

Στην ενότητα 5.3.4 γίνεται μελέτη στον συνδυασμό των προβλημάτων της κλάσης C. Επιλέχθηκε αυτή η κλάση καθώς παρουσιάζει την καλύτερη συμπεριφορά σε σχέση με τις άλλες δύο. Στη μελέτη αυτή, τα προβλήματα τρέχουν σε compact, spread και stripe πολιτική. Στην stripe πολιτική γίνεται συνδυασμός κάθε ενός προβλήματος με κάθε άλλο.

Όπως φαίνεται από τις γραφικές, η spread πολιτική είναι η καλύτερη, αφού τα προβλήματα μοιράζονται σε περισσότερα μηχανήματα και χρησιμοποιούν την μισή επεξεργαστική ισχύ αλλά όλη τη memory. Συνεπώς διαθέτουν περισσότερα μηχανήματα για να τα χρησιμοποιήσουν όπως χρειάζονται και είναι μόνα τους σε αυτά. Η πολιτική αυτή όμως αποτελεί απλώς αντικείμενο μελέτης και δεν εφαρμόζεται σε πραγματικά συστήματα, καθώς αφήνει ανεκμετάλλευτους φυσικούς πόρους του συστήματος.

Στην πολιτική stripe, που είναι αυτή που μπορεί να έχει αντίκτυπο στον πραγματικό κόσμο, τα προβλήματα μοιράζονται σε περισσότερα συστήματα, στα οποία τρέχει παράλληλα κάποιο άλλο πρόβλημα. Μπορεί να τρέχουν και παραπάνω από δύο προβλήματα ταυτόχρονα, ωστόσο αυτή η περίπτωση δε θα απασχολήσει την παρούσα εργασία.

Τα προβλήματα έχουν διαφορετικούς χρόνους εκτέλεσης το ένα με το άλλο, που μπορεί να απέχουν έως και 1 με 2 τάξεις μεγέθους. Για να συνδυαστούν μεταξύ τους, έγινε κανονικοποίηση των μικρών σε διάρκεια προβλημάτων. Αυτό σημαίνει πως εάν ένα πρόβλημα Α χρειάζεται Χ δευτερόλεπτα για την εκτελεσή του και ένα πρόβλημα Β χρειάζεται Χ/5 δευτερόλεπτα για την εκτέλεσή του, το πρόβλημα Β θα εκτελεστεί 5 φορές συνεχόμενα για να έχει συνολική διάρκεια ίση με το Α. Αν δεν γινόταν κανονικοποίηση το μικρό πρόβλημα θα τελείωνε γρήγορα και το μεγάλο θα έτρεχε σε πολιτική spread για το υπόλοιπό του, με αποτέλεσμα να μην παράγονται σωστά οι χρόνοι για stripe πολιτική. Συνεπώς οι γραφικές παραστάσεις της ενότητας 5.3.4 είναι normalized και αντιπροσωπεύουν ορθώς όλους τους συνδυασμούς των προβλημάτων. Παράδειγμα κώδικα normalization φαίνεται στον κώδικα 6.

```
apps1=('bt.C.x' 'cg.C.x' 'ft.C.x' 'lu.C.x' 'sp.C.x')
apps2=('ep.C.x' 'is.C.x' 'mg.C.x')

times1=(1 1 1 1 1)
times2=(11 10 6 5 7 4 7 7 4 8 5 4 23 20 12)

#times=(2 7 1 17 1 6)

offset=0
i=0
j=0

for i in "${!apps1[@]}"; do

  for j in "${!apps2[@]}"; do

    echo "($i) ($j)"
    # first apps
    {
    file="$appdir/${apps1[$i]}"
    out="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/${apps1[i]}.${apps2[j]}.out"
    err="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/${apps1[i]}.${apps2[j]}.err"
    echo $file >> $out
    for (( t=1; t<=${times1[i]}; t++ )); do
      mpirun -np $procs -v --report-bindings --timestamp-output --rankfile $RANKFILEa --mca btl self,tcp $file >> $out 2>> $err
    done
    } &

    # second apps
    {
    file="$appdir/${apps2[$j]}"
    out="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/${apps2[j]}.${apps1[i]}.out"
    err="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/${apps2[j]}.${apps1[i]}.err"
    echo $file >> $out
    for (( t=1; t<=${times2[j + 3*i]}; t++ )); do
      mpirun -np $procs -v --report-bindings --timestamp-output --rankfile $RANKFILEb --mca btl self,tcp $file >> $out 2>> $err
    done
    } &

    wait

  done
```

Κώδικας 6: Normalizing example

5.5 Class D

Η κλάση D είναι μία κλάση που έχει προβλήματα πολύ μεγαλύτερα από τις προηγούμενες. Στο cluster της σχολής τα NPBs προβλήματα δεν ήταν εφικτό να τρέξουν όλα στα μηχανήματα. Παρατίθεται εικόνα με τα αποτελέσματα:

D										
Policy	Processes	Topology	BT	CG	EP	FT	IS	LU	MG	SP
Compact	8	1x8			1.092,42					
Spread	8	2x4			1.111,78			6.157,82		
Compact	16	2x8			552,50			3.593,07		
Spread	16	4x4	5.015,18	5.008,26	547,58		211,82	2.652,71	728,36	9.373,79
Compact	32	4x8		2.337,49	304,84		214,91	2.401,40	565,63	9.252,09
Spread	32	8x4	3.290,75	1.848,78	278,85		141,68	1.779,49	394,23	6.195,61
Compact	64	8x8	1.744,36	1.601,70	160,05		165,39	1.289,95	260,34	3.800,22
Spread	64	16x4			143,39					

Σχήμα 16: CLASS D

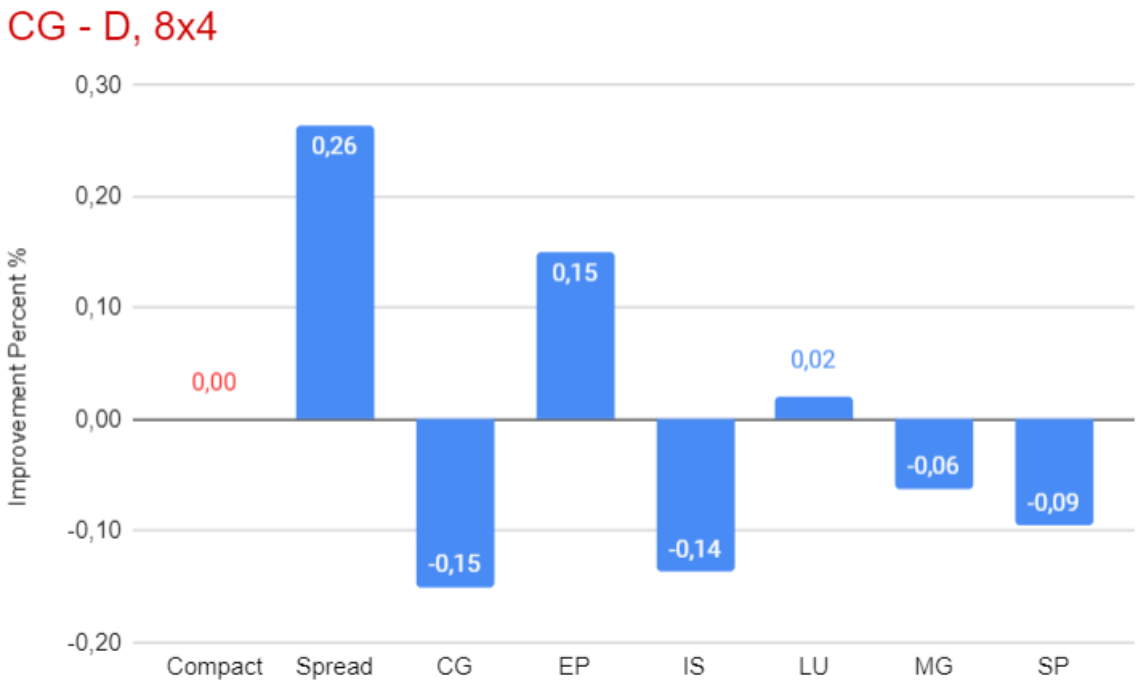
Όπως φαίνεται στον παραπάνω πίνακα, μόνο το πρόβλημα EP έτρεξε για όλους τους συνδυασμούς πυρήνων – κόμβων. Το πρόβλημα FT δεν έτρεξε καθόλου, ενώ τα υπόλοιπα έτρεξαν σε συγκεκριμένες περιπτώσεις.

Οι χρόνοι της κλάσης D ωστόσο, επειδή τα προβλήματα πλέον είναι μεγάλα, παρουσιάζουν μείωση όσο αυξάνονται οι πυρήνες, όπως αναμέναμε. Συνεπώς, στην κλάση αυτή έχουμε ένα καλό speedup σε σχέση με τις άλλες κλάσεις.

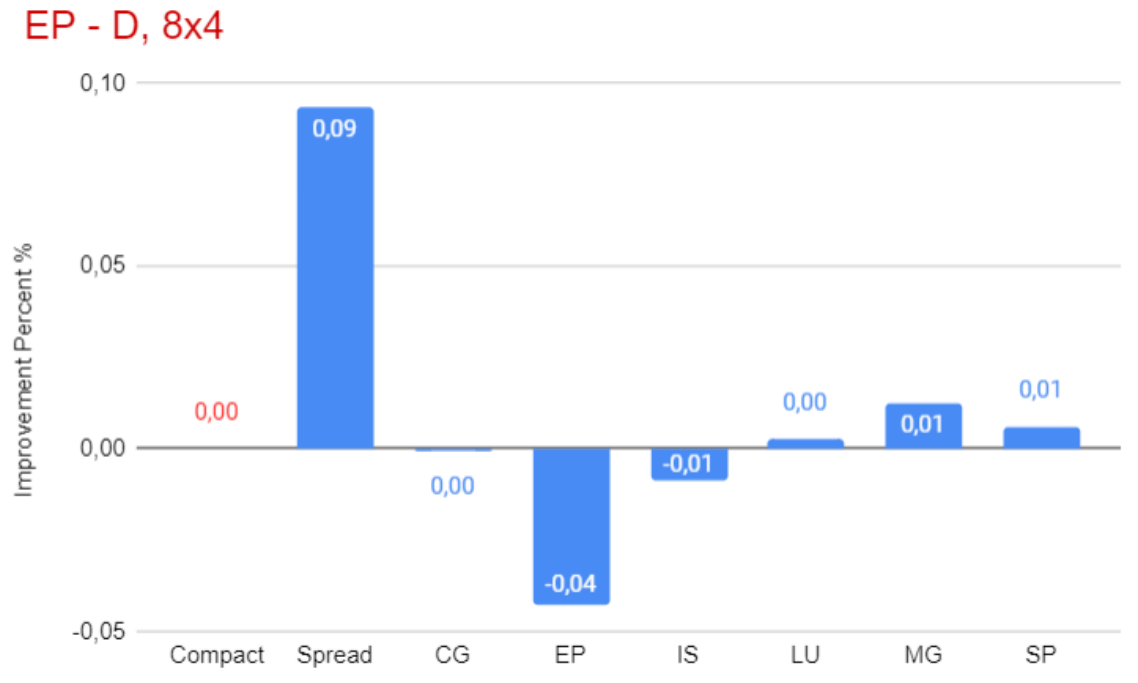
Ομοίως με την μελέτη για την κλάση C, στην κλάση D, έγιναν normalization τα μικρά προβλήματα στα μεγαλύτερα, ώστε να γίνουν stripe όλοι οι συνδυασμοί. Εξετάστηκε συγκεκριμένα η περίπτωση των 32 πυρήνων, καθώς μόνο σε αυτή τα προβλήματα είχαν αποτέλεσμα και σε πολιτική compact και σε πολιτική spread.

Συνεπώς, εξετάστηκαν τα εξής προβλήματα: CG, EP, IS, LU, MG, SP.

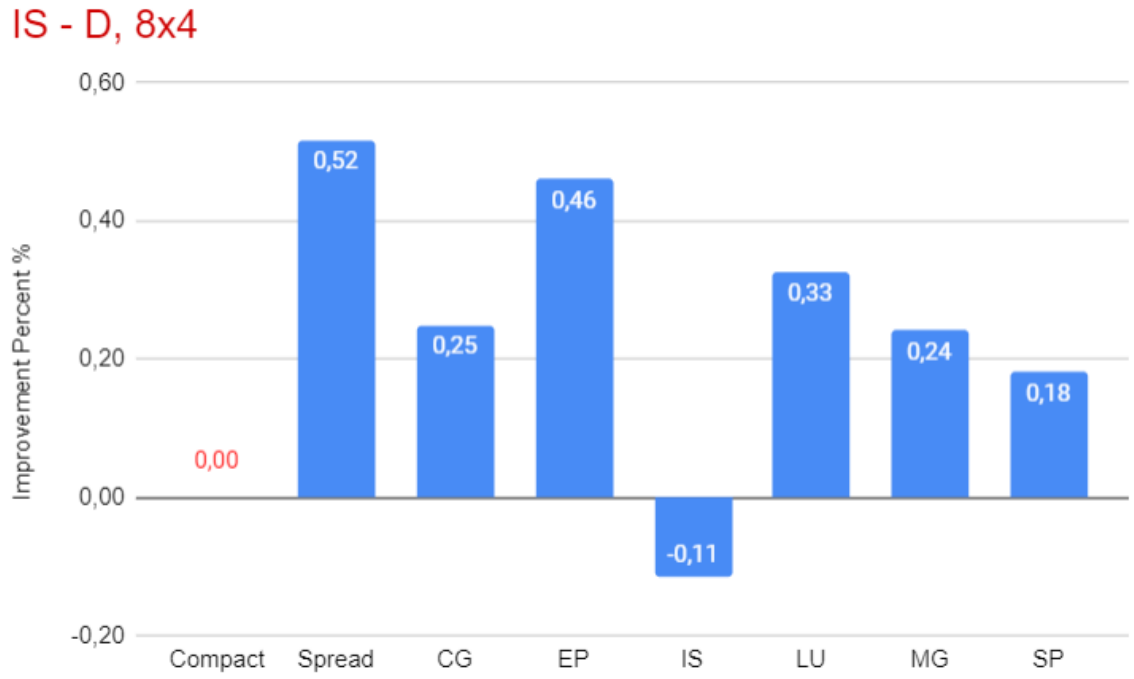
Παρουσιάζονται οι γραφικές παραστάσεις με τα αποτελέσματα για τα έξι παραπάνω προβλήματα.



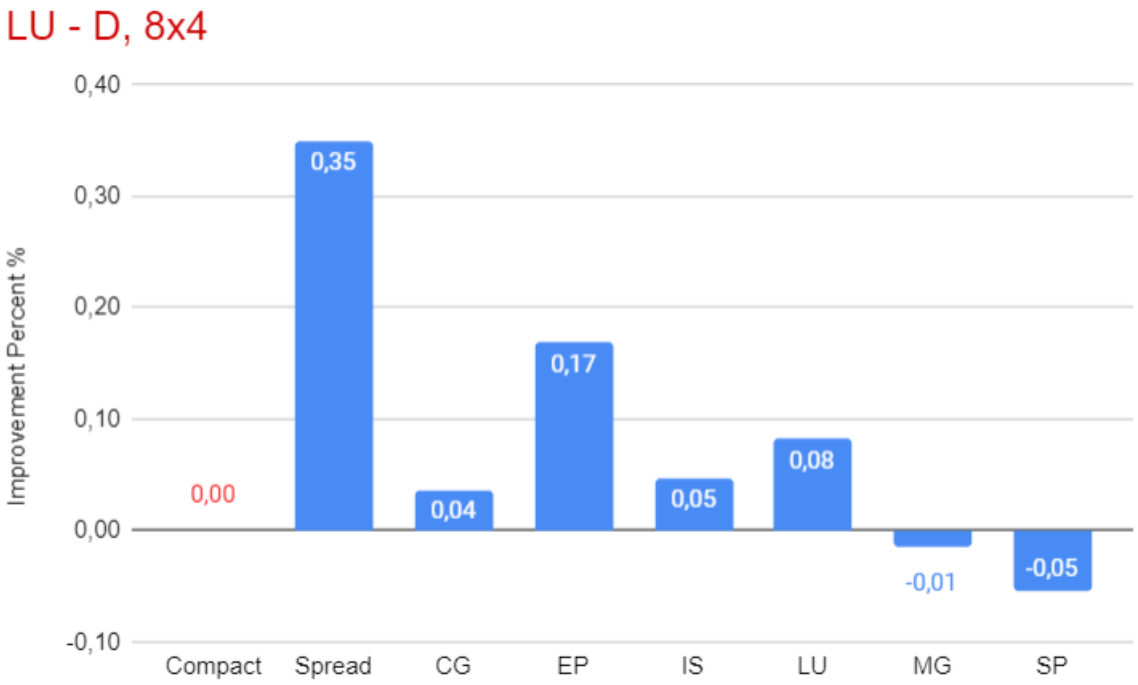
Γραφικές 105: CG - D 8x4



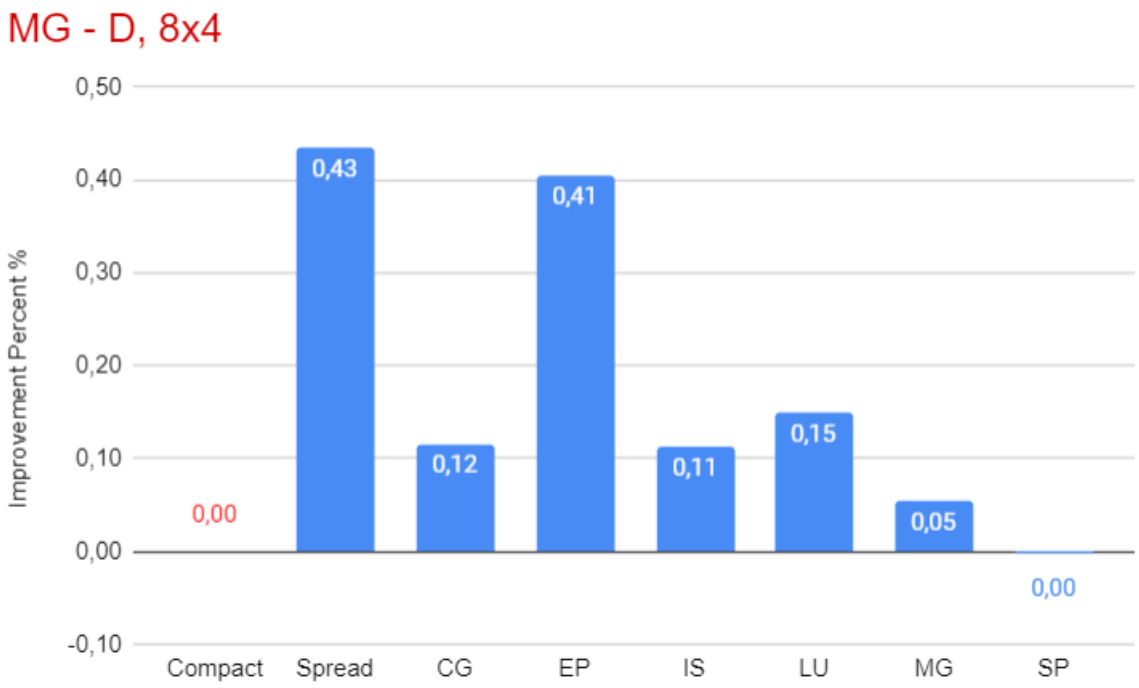
Γραφικές 106: EP - D 8x4



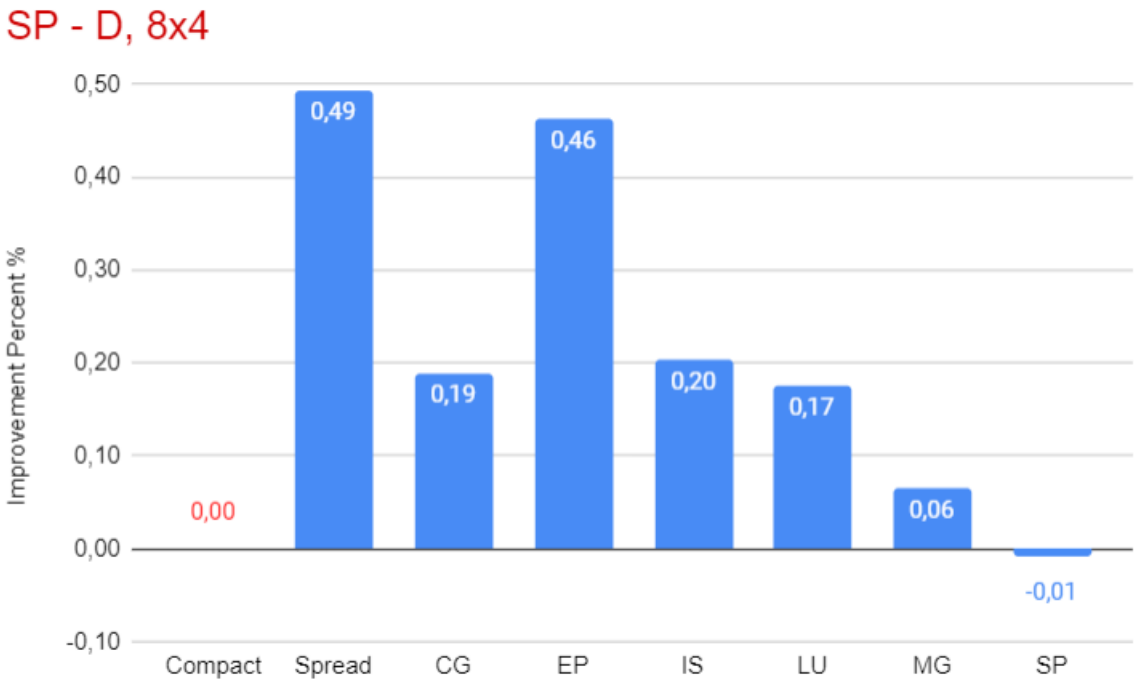
Γραφικές 107: IS - D 8x4



Γραφικές 108: LU - D 8x4



Γραφικές 109: MG - D 8x4



Γραφικές 110: SP - D 8x4

Συγκρίνοντας τις γραφικές παραστάσεις της κλάσης D με αυτές της κλάσης C στο κεφάλαιο 5.3.4, γίνεται φανερό πως η κλάση D έχει πολύ καλύτερο scale. Στην κλάση C, στα περισσότερα προβλήματα φαίνεται πως δεν αξίζει να γίνει stripe το πρόβλημα με κάποιο άλλο, καθώς ο χρόνος ο συνολικός των δύο προβλημάτων που γίνονται stripe αντί για compact αυξάνεται. Στην κλάση D, από την άλλη ο χρόνος αυτός μικραίνει.

Συγκεκριμένα, τα προβλήματα IS, LU, MG, SP φαίνεται πως συμφέρει πολύ να συνδυαστούν με όλα σχεδόν τα προβλήματα, αφού έχουν σημαντικό κέρδος. Το πρόβλημα EP, φαίνεται να έχει ανεβοκατεβάσματα στη γραφική του παράσταση ωστόσο στην πραγματικότητα είναι ουδέτερο. Παρατηρώντας τα νούμερα της γραφικής του παράστασης 106, φαίνεται πως το πόσο κερδίζει ή χάνει σε συνδυασμό με τα άλλα προβλήματα είναι πολύ λίγο. Συνεπώς μένει σταθερό, αφού αυτή η μικρή μεταβολή μπορεί να οφείλεται στο συγκεκριμένο τρέξιμο. Το πρόβλημα CG, γραφική παράσταση 105, φαίνεται να μην συνεργάζεται καλά με τα περισσότερα προβλήματα εκτός από το EP και το LU. Στον αλγόριθμο του επόμενου κεφαλαίου η ανάλυση αυτή της κλάσης D έχει ληφθεί υπόψιν.

6. Αλγόριθμος co-scheduling

Εξετάζοντας τα αποτελέσματα του προηγούμενου κεφαλαίου, φαίνεται πως οι κλάσεις A,B δεν αξίζει να μελετηθούν σε πολλούς πυρήνες, με το οποίο ασχολείται η παρούσα εργασία. Στην κλάση C, φαίνεται πως τα προβλήματα έχουν καλύτερη συμπεριφορά αυξάνοντας των αριθμών των πυρήνων σε 32 και 64. Ωστόσο στο κεφάλαιο 5.3.4 που εξετάστηκε ο συνδυασμός τους σε πολιτική stripe, δεν φάνηκε να υπάρχει περιθώριο βελτίωσης.

Η κλάση D, παρουσιάζει μια καλή συμπεριφορά. Τα αποτελέσματα των Nas Parallel Benchmark προβλημάτων όπως φαίνονται στον πίνακα 16, δείχνουν πως οι χρόνοι μειώνονται σε όλα τα προβλήματα, από αυτά που μπορούν να τρέξουν στο cluster της σχολής, όσο αυξάνονται οι πυρήνες. Στην μελέτη που έγινε στη συνέχεια με την πολιτική stripe, όπως φαίνεται από τα διαγράμματα 105,106,107,108,109,110, υπάρχει μεγάλο περιθώριο βελτίωσης του χρόνου εκτέλεσης των προβλημάτων αν αυτά τρέχουν σε stripe πολιτική. Για τον λόγο αυτό, ο αλγόριθμος αναπτύχθηκε για την κλάση D, ώστε να υπάρξει το καλύτερο δυνατό αποτέλεσμα.

Σκοπός της προσπάθειας, είναι η σύγκριση της πολιτικής compact με την πολιτική stripe. Στην κλάση D, όπως φαίνεται στον πίνακα 16, μόνο για την κατηγορία των 32 πυρήνων υπάρχουν αποτελέσματα για αρκετά benchmarks, συγκεκριμένα για τα CG, EP, IS, LU, MG, SP. Το MG πρόβλημα ωστόσο, παρόλο που έχει αποτέλεσμα για 32 πυρήνες σε πολιτική spread, δηλαδή 4 πυρήνες από 8 μηχανήματα, αδυνατεί να τρέξει σωστά στο σύστημα της σχολής. Αυτό συμβαίνει γιατί τρέχει αποτελεσματικά στο πρώτο socket των μηχανημάτων, αλλά αν χρειαστεί να τρέξει στο δεύτερο socket τότε το πρόγραμμα τερματίζει. Συνεπώς θα μείνει έξω από την συγκεκριμένη μελέτη, αφού ο αλγόριθμος διαλέγει για κάθε πρόγραμμα στην stripe πολιτική αν θα τρέξει στο πρώτο ή στο δεύτερο socket των οκτώ μηχανημάτων και είναι απαραίτητο τα προγράμματα που θα χρησιμοποιηθούν να έχουν την ελευθερία και την δυνατότητα να τρέξουν και στα δύο. Για την οικονομία του χρόνου, καθώς τα προβλήματα της κλάσης D είναι πολύ χρονοβόρα και η μελέτη που πραγματοποιείται απαιτεί οκτώ κόμβους του συστήματος της σχολής να τρέχουν συνεχόμενα, το πρόβλημα SP δεν θα συμπεριληφθεί ούτε αυτό στον αλγόριθμο, καθώς απαιτεί υπερβολικά πολύ χρόνο από μόνο του και ο αλγόριθμος θα χρησιμοποιήσει μια μεγάλη ουρά εργασιών που αν περιλαμβάνει το SP, θα απαιτεί

πολλές ώρες ή και μέρες για να ολοκληρωθεί, πράγμα μη εφικτό στο συγκεκριμένο σύστημα.

6.1 Ο Αλγόριθμος

Στην ενότητα αυτή παρουσιάζεται με τη μορφή ψευδοκώδικα ο αλγόριθμός μου. Σε python ο αλγόριθμος υπάρχει στο παράρτημα.

```
CG_COUNT = 0
EP_COUNT = 0
IS_COUNT = 0
LU_COUNT = 0
MG_COUNT = 0
SP_COUNT = 0

for task in taskQueue
        if task == PROBLEM_CG
                increase CG_COUNT by 1
        else if task == PROBLEM_EP
                increase EP_COUNT by 1
        else if task == PROBLEM_IS
                increase IS_COUNT by 1
        else if task == PROBLEM_LU
                increase LU_COUNT by 1
        else if task == PROBLEM_MG
                increase MG_COUNT by 1
        else if task == PROBLEM_SP
                increase SP_COUNT by 1
        else
                print error



result_queue = empty_list
compact_list = empty_list
stripe_list = empty_list
```

Κώδικας 7: Pseudocode 1

Σύμφωνα με τα προηγούμενα, στον αλγόριθμο θα χρησιμοποιηθούν τελικά τα προβλήματα CG, EP, IS, LU, για 32 πυρήνες. Για την μελέτη αυτή θα χρησιμοποιηθούν οκτώ συγκεκριμένα clones του συστήματος που έχουν ίδια χαρακτηριστικά.

Ο αυτόματος αλγόριθμος του χρονοδρομολογητή Torque που χρησιμοποιεί το σύστημα της σχολής, χρησιμοποιεί την πολιτική compact, δηλαδή οι 32 πυρήνες κατανέμονται σε 4 κόμβους των 8 πυρήνων ο καθένας. Ο δικός μου αλγόριθμος έχει επιλογή να κάνει τα προβλήματα stripe, δηλαδή οι 32 πυρήνες να κατανέμονται σε 8 μηχανήματα από τα οποία θα χρησιμοποιούνται οι 4 πυρήνες. Ταυτόχρονα τους

άλλους 4 πυρήνες θα χρησιμοποιεί κάποιο άλλο Nas Parallel Benchmark, που θα συνεργάζεται καλά με το πρώτο.

Στον κώδικα 7, φαίνεται το πρώτο μέρος του κώδικα όπου διαβάζεται η ουρά(queue) με τα Nas Parallel Benchmarks και κρατούνται μεταβλητές με το πλήθος του κάθε ενός.

```
if IS_COUNT > 0                                         // there are IS problems to be scheduled
        while EP_COUNT > 0 and IS_COUNT > 0             // there are EP and IS problems to be scheduled
                add to stripe_list the ISsa_problem
                add to stripe_list the EPsb_problem
                reduce IS_COUNT by 1
                reduce EP_COUNT by 1
        if IS_COUNT > 0         // there are IS problems to be scheduled
                while LU_COUNT > 0 and IS_COUNT > 0     // there are LU and IS problems to be scheduled
                        add to stripe_list the ISsa_problem
                        add to stripe_list the LUsb_problem
                        reduce IS_COUNT by 1
                        reduce LU_COUNT by 1


if IS_COUNT > 0                                         // there are IS problems to be scheduled
        while IS_COUNT > 0                              // there are IS problems to be scheduled
                add to compact_list the ISc_problem
                reduce IS_COUNT by 1
        while CG_COUNT > 0
                add to compact_list the CGc_problem
                reduce CG_COUNT by 1


if CG_COUNT > 0                                         // there are CG problems to be scheduled
        while EP_COUNT > 0 and CG_COUNT > 0             // there are EP and CG problems to be scheduled
                add to stripe_list the EPsa_problem
                add to stripe_list the CGsb_problem
                reduce EP_COUNT by 1
                reduce CG_COUNT by 1
        if CG_COUNT > 0         // there are CG problems to be scheduled
                while LU_COUNT > 0 and CG_COUNT > 0     // there are LU and CG problems to be scheduled
                        add to stripe_list the LUsa_problem
                        add to stripe_list the CGsb_problem
                        reduce LU_COUNT by 1
                        reduce CG_COUNT by 1
```

Κώδικας 8: Pseudocode 2

```
if CG_COUNT > 0                                         // there are CG problems to be scheduled
        while CG_COUNT > 0                              // there are CG problems to be scheduled
                add to compact_list the CGc_problem
                reduce CG_COUNT by 1


if LU_COUNT > 0                                         // there are LU problems to be scheduled
        while LU_COUNT > 0 and EP_COUNT > 0             // there are LU and EP problems to be scheduled
                add to stripe_list the LUsa_problem
                add to stripe_list the EPsb_problem
                reduce LU_COUNT by 1
                reduce EP_COUNT by 1

        if LU_COUNT > 0                                 // there are LU problems to be scheduled
                TEMPORARY_FLAG = 0
                while LU_COUNT > 0                      // there are LU problems to be scheduled
                        if TEMPORARY_FLAG == 0
                                add to stripe_list the LUsa_problem
                                reduce LU_COUNT by 1
                                TEMPORARY_FLAG = 1
                        else if TEMPORARY_FLAG == 1
                                add to stripe_list the LUsb_problem
                                reduce LU_COUNT by 1
                                TEMPORARY_FLAG = 0
```

Κώδικας 9: Pseudocode 3

```
if EP_COUNT > 0                                     // there are EP problems to be scheduled
        if EP_COUNT%2 == 0                          // there is an even number of EP problems to be scheduled
                while EP_COUNT > 0                  // there are EP problems to be scheduled
                        add to compact_list the EPc_problem
                        reduce EP_COUNT by 1
        else
                TEMPORARY_FLAG = 1
                while EP_COUNT > 0                  // there are EP problems to be scheduled
                        if TEMPORARY_FLAG == 0
                                add to stripe_list the EPsa_problem
                                reduce EP_COUNT by 1
                                TEMPORARY_FLAG = 1
                        else if TEMPORARY_FLAG == 1
                                add to stripe_list the EPsb_problem
                                reduce EP_COUNT by 1
                                TEMPORARY_FLAG = 0
```

Κώδικας 10: Pseudocode 4

Στους παραπάνω κώδικες 8,9,10, φαίνεται το κύριο μέρος του αλγορίθμου. Η πολιτική που ακολουθείται είναι πως ο αλγόριθμος φτιάχνει ζευγάρια προβλημάτων που συνεργάζονται καλά μεταξύ τους. Φτιάχνει δύο λίστες, μία compact που σηματοδοτεί πως τα προβλήματα που είναι σε αυτή τη λίστα θα τρέξουν compact,

δηλαδή με την αυτόματη πολιτική του Torque και μία stripe στην οποία τα προβλήματα θα τρέξουν σε ζευγάρια στους κόμβους, το ένα από τα δύο στο ένα socket των κόμβων και το άλλο στο άλλο socket.

Ο αλγόριθμος τρέχει συνεχόμενα με ένα while έως ότου όλα τα προβλήματα να μπουν σε μια από τις δύο λίστες. Αρχικά γεμίζει την stripe λίστα και όταν δεν μπορεί να φτιάξει άλλα ζευγάρια stripe, τοποθετεί τα εναπομείναντα προβλήματα στην ουρά compact. Για την ουρά stripe, ο τρόπος που φτιάχνει τα ζευγάρια έχει οριστεί από τη μελέτη των προβλημάτων της κλάσης D. Το πρόβλημα αυτό είναι NP πλήρες και συνεπώς δεν μπορεί να λυθεί μονοσήμαντα και σωστά σε κάθε περίπτωση. Λόγω του μικρού πλήθους προβλημάτων και της γνώσης της συμπεριφοράς τους σε συνάρτηση με τα υπόλοιπα από τη μελέτη της κλάσης D, το πρόβλημα εδώ αντιμετωπίζεται για τα συγκεκριμένα τέσσερα προβλήματα με την εξής λογική:

Πρώτα ταιριάζει το πρόγραμμα IS. Το πρόγραμμα IS σύμφωνα με τη γραφική 107 έχει τη δυνατότητα της μεγαλύτερης βελτίωσης του χρόνου του. Στην αρχή γίνεται stripe με προβλήματα EP και αν αυτά τελειώσουν με προβλήματα LU. Αν μένουν και άλλα IS τότε αυτά και τα πιθανόν υπάρχοντα CG γίνονται compact καθώς δεν συνεργάζονται καλά μεταξύ τους και κανένα από τα δύο δεν μπορεί να γίνει stripe με τον εαυτό του. Στο πρώτα βήμα του αλγορίθμου δηλαδή, γίνεται προσπάθεια για συνδυασμό του προβλήματος IS που είναι integer sort και συνεπώς communication intensive με το EP που είναι embarrassingly parallel και συνεπώς compute intensive και με το LU. Όμοια λογική, σύμφωνα και με τα διαγράμματα των προγραμμάτων και τους χρόνους τους ακολουθείται και στα επόμενα στάδια.

Έπειτα ταιριάζουν τα προγράμματα CG με τα EP, καθώς τα προγράμματα CG, έχουν τους μεγαλύτερους περιορισμούς σύμφωνα με τη γραφική 105. Στη συνέχεια, αν τα EP τελειώσουν, τα CG συνεχίζουν με το αμέσως επόμενο πρόβλημα που συνεργάζονται καλά που είναι το LU. Αν τελειώσουν και τα LU, τότε το CG μπαίνει compact με τον εαυτό του, καθώς έχει εξασφαλιστεί πως δεν υπάρχουν προβλήματα IS από το προηγούμενο βήμα και το CG δεν συμφέρει να μπει stripe με τον εαυτό του.

Στη συνέχεια γίνονται stripe τα LU προβλήματα με το μόνο πρόβλημα που έχει μείνει, δηλαδή το EP. Αν τα EP τελειώσουν, τα LU μπαίνουν στην ουρά stripe καθώς το LU συνεργάζεται καλά σε πολιτική stripe με τον εαυτό του, όπως φαίνεται στο σχήμα 108.

Τέλος, στο σημείο αυτό ο αλγόριθμος είτε θα έχει τελειώσει, είτε θα συνεχίζει με μόνο EP εναπομείναντα προβλήματα. Τα EP, ακολουθώντας την ροή του έως τώρα αλγορίθμου και εφόσον είτε τρέξουν σε compact πολιτική είτε σε stripe έχουν το ίδιο αποτέλεσμα (σχήμα 106) , επιλέγεται με μια πιο ιδιαίτερη λογική αν θα μπουν στην compact ή στην stripe ουρά, όπως φαίνεται στον κώδικα 10.

```
result_queue = concatenate(stripe_list, compact_list)
```

Κώδικας 11: Pseudocode 5

Τέλος όπως φαίνεται στον παραπάνω κώδικα, ο αλγόριθμος ενώνει τις ουρές stripe και compact σε μια ολική και παράγει ένα νέο αρχείο queue που είναι το τελικό και αυτό θα υποβληθεί στον Torque. Σημειώνεται πως στην τελική ουρά υποβάλλονται πρώτα τα προγράμματα της ουράς stripe και μετά αν υπάρχουν τα προγράμματα της ουράς compact.

6.2 Σύγκριση αλγορίθμων

Στις περισσότερες περιπτώσεις, αναλόγως την ουρά υποβολής εργασιών, ο αλγόριθμος καταφέρνει να αποφασίσει και να κάνει τα προβλήματα να τρέξουν σε πολιτική stripe. Μόνο αν κάποιο από τα προγράμματα που δεν συνεργάζεται καλά με τον εαυτό του υπάρχει περισσότερες φορές από τα άλλα στην αρχική ουρά, ο αλγόριθμος θα χρησιμοποιήσει και την πολιτική compact.

Για την σύγκριση του αυτόματου – default αλγορίθμου χρονοδρομολόγησης του Torque με τον αλγόριθμο της ενότητας 6.1, αναπτύχθηκαν 2 σενάρια υποβολής εργασιών. Το πρώτο σενάριο περιλαμβάνει μια ουρά 10 εργασιών και το δεύτερο σενάριο μια ουρά 30 εργασιών. Οι ουρές των προβλημάτων είναι τυχαίες, καθώς έχουν παραχθεί από πρόγραμμα, με αποτέλεσμα τα προβλήματα CG, EP, IS, LU να είναι ισοπίθανο να εμφανιστούν.

Παρουσιάζονται τα σενάρια εργασιών που δοκιμάστηκαν.

Queue 10

```
lu.D.32
cg.D.32
lu.D.32
ep.D.32
lu.D.32
ep.D.32
is.D.32
ep.D.32
ep.D.32
lu.D.32
```

Σχήμα 17: Queue 10

Queue 30

```
cg.D.32
cg.D.32
cg.D.32
ep.D.32
is.D.32
ep.D.32
ep.D.32
is.D.32
is.D.32
is.D.32
ep.D.32
is.D.32
lu.D.32
ep.D.32
ep.D.32
is.D.32
ep.D.32
cg.D.32
ep.D.32
cg.D.32
cg.D.32
lu.D.32
ep.D.32
lu.D.32
lu.D.32
ep.D.32
is.D.32
ep.D.32
is.D.32
lu.D.32
```

Σχήμα 18: Queue 30

Στις παραπάνω εικόνες βλέπουμε μια τυχαία ουρά εργασιών κλάσης D 10 εργασιών και μια 30 εργασιών.

Στην παρακάτω εικόνα φαίνεται πως τρέχουν τα προγράμματα στο cluster της σχολής, στους 8 πυρήνες που έχουν επιλεχθεί για τα πειράματα.

```
-------------------------------------------------------------------------------------------------------------------------
  NodeName | Taken |  Load|Mem(MB)| FreeMem | State |                  Attributes                  | Users(Slots)
-------------------------------------------------------------------------------------------------------------------------
    clone1 |  8/8  |  8.00|   7983|  98.28% | Full  |clones,mpi,disk,clover,highmem,parlab,sniper  | ppeppas(8)
    clone2 |  8/8  |  1.58|   7983|  98.65% | Full  |clones,mpi,disk,clover,highmem,parlab,sniper  |
    clone3 |  8/8  |  8.07|   7983|  98.35% | Full  |clones,mpi,disk,clover,highmem,parlab,sniper  | ppeppas(8)
    clone4 |  8/8  |  8.00|   7983|  98.37% | Full  |clones,mpi,disk,clover,highmem,parlab,sniper  | ppeppas(8)
    clone5 |  8/8  |  8.00|   7983|  98.40% | Full  |clones,mpi,disk,clover,highmem,parlab         | ppeppas(8)
    clone6 |  0/8  |  0.27|   3951|  97.67% | Empty |clones,mpi,mx10g,disk,clover,lowmem,kvm,parlab |
    clone7 |  0/8  |  0.14|   3951|  97.66% | Empty |clones,mpi,mx10g,clover,lowmem,collab,parlab   |
    clone8 |  0/8  |  0.13|   2000|  95.71% | Empty |clones,mpi,disk,clover,lowmem,collab,parlab    |
    clone9 |  0/8  |  0.05|   3008|  97.07% | Empty |clones,collab,mpi,disk,clover,lowmem,parlab    |
   clone10 |  0/8  |  0.11|   2000|  95.89% | Empty |clones,collab,mpi,disk,clover,lowmem,parlab    |
   clone11 |  0/8  |  0.22|   5967|  98.34% | Empty |clones,mx10g,collab,mpi,disk,clover,lowmem,parlab |
   clone12 |  0/8  |  0.11|   3951|  97.64% | Empty |clones,mpi,disk,clover,lowmem,parlab           |
   clone13 |  8/8  |  7.94|   7983|  21.26% | Full  |clones,mpi,disk,clover,highmem,parlab,sniper  | ppeppas(8)
   clone14 |  8/8  |  7.87|   7983|  21.31% | Full  |clones,mpi,disk,clover,highmem,sniper,parlab  | ppeppas(8)
   clone15 |  8/8  |  8.01|   7983|  21.13% | Full  |clones,mpi,disk,clover,highmem,sniper,parlab  | ppeppas(8)
   clone16 |  8/8  |  7.82|   7983|  21.56% | Full  |clones,mpi,disk,clover,highmem,sniper,parlab  | ppeppas(8)
   clone19 |  0/8  |  0.10|   7983|  98.77% | Empty |clones,mpi,mx10g,disk,harper,highmem,sniper,parlab|
   clone20 |  0/8  |  0.12|   7983|  98.77% | Empty |clones,mpi,mx10g,disk,harper,highmem,sniper,parlab|
```

Σχήμα 19: Jobs running in clones

Τα clones αυτά έχουν παρόμοια χαρακτηριστικά, και με δοκιμές σε προγράμματα μικρότερων κλάσεων, τα προβλήματα έκαναν παρόμοιους χρόνους. Συνεπώς όλη η μελέτη του κεφαλαίου αυτού έγινε σε αυτά τα nodes.

```
#!/bin/bash

input="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/myAlgo/queue"

while IFS= read -r line
do
        if [ "$line" == "CGc" ]
        then
                qsub -q parlab -l nodes=4:ppn=8:highmem:clover /home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/mpirun/mpirun.cg.c.D
        elif [ "$line" == "CGsa" ]
        then
                qsub -q parlab -l nodes=8:ppn=4:highmem:clover /home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/mpirun/mpirun.cg.sa.D
        elif [ "$line" == "CGsb" ]
        then
                qsub -q parlab -l nodes=8:ppn=4:highmem:clover /home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/mpirun/mpirun.cg.sb.D
        elif [ "$line" == "EPc" ]
        then
                qsub -q parlab -l nodes=4:ppn=8:highmem:clover /home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/mpirun/mpirun.ep.c.D
        elif [ "$line" == "EPsa" ]
        then
                qsub -q parlab -l nodes=8:ppn=4:highmem:clover /home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/mpirun/mpirun.ep.sa.D
        elif [ "$line" == "EPsb" ]
        then
                qsub -q parlab -l nodes=8:ppn=4:highmem:clover /home/users/ppeppas/NPB3.4/NPB3.4-MPI/final/mpirun/mpirun.ep.sb.D
```

Σχήμα 20: Finalrun.sh file

Το παραπάνω αρχείο είναι αυτό που διαβάζει την ουρά που έχει παράξει ο αλγόριθμος μου και την υποβάλλει στο cluster για τρέξιμο. Τα mpirun αρχεία έχουν την ακόλουθη εικόνα:

```
#!/bin/bash

## Give the Job a descriptive name
#PBS -N ep.sa.D.8x4

## Output and error files
#PBS -o /home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/run.ep.sa.D.8x4.out
#PBS -e /home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/run.ep.sa.D.8x4.err

## Limit memory, runtime etc.
#PBS -l walltime=10:00:00

## How many nodes:processors_per_node should we get?
#PBS -l nodes=8:ppn=4

export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/home/users/ppeppas/mpi1/bin:$PATH
export LD_LIBRARY_PATH=/various/common_tools/gcc-4.5.2/lib64/

now=$(date +%d-%b-%H_%M)

out="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/ep.s.$now.out"
err="/home/users/ppeppas/NPB3.4/NPB3.4-MPI/logs/ep.s.$now.err"

echo $now >> $out

RANKFILEa=/home/users/ppeppas/NPB3.4/NPB3.4-MPI/mixedall/rankfiles/rankfile8x4a

for (( t=1; t<=7; t++ )); do
    mpirun -np 32 --rankfile $RANKFILEa -v --report-bindings --timestamp-output --mca btl self,tcp /home/users/ppeppas/NPB3.4/NPB3.4-MPI/bin/ep.D.x >> $out 2>> $err
done
```

Κώδικας 12: Final mpirun example

Στο παραπάνω αρχείο φαίνεται και η κανονικοποίηση που γίνεται στα προβλήματα όταν χρειάζεται για να έχουν συνολικά παρόμοιο χρόνο εκτέλεσης μεταξύ τους.

6.3 Αποτελέσματα

Παρουσιάζονται τα παραγόμενα αρχεία από τους δύο αλγορίθμους για την ουρά των 10 εργασιών.

```
cg.c.10-Aug-08_28.err  ep.c.10-Aug-11_01.err  lu.c.10-Aug-09_41.err  run.is.c.D.4x8.err
cg.c.10-Aug-08_28.out  ep.c.10-Aug-11_01.out  lu.c.10-Aug-09_41.out  run.is.c.D.4x8.out
ep.c.10-Aug-09_09.err  is.c.10-Aug-10_19.err  lu.c.10-Aug-11_10.err  run.lu.c.D.4x8.err
ep.c.10-Aug-09_09.out  is.c.10-Aug-10_19.out  lu.c.10-Aug-11_10.out  run.lu.c.D.4x8.out
ep.c.10-Aug-09_49.err  lu.c.10-Aug-08_30.err  run.cg.c.D.4x8.err     totaltime
ep.c.10-Aug-09_49.out  lu.c.10-Aug-08_30.out  run.cg.c.D.4x8.out
ep.c.10-Aug-10_25.err  lu.c.10-Aug-09_09.err  run.ep.c.D.4x8.err
ep.c.10-Aug-10_25.out  lu.c.10-Aug-09_09.out  run.ep.c.D.4x8.out
```

Σχήμα 21: Default results example

```
cg.s.10-Aug-13_16.err  ep.s.10-Aug-14_27.out  lu.s.10-Aug-15_03.err  run.is.sa.D.8x4.out
cg.s.10-Aug-13_16.out  is.s.10-Aug-12_39.err  lu.s.10-Aug-15_03.out  run.lu.sa.D.8x4.err
ep.s.10-Aug-12_39.err  is.s.10-Aug-12_39.out  run.cg.sb.D.8x4.err    run.lu.sa.D.8x4.out
ep.s.10-Aug-12_39.out  lu.s.10-Aug-13_51.err  run.cg.sb.D.8x4.out    run.lu.sb.D.8x4.err
ep.s.10-Aug-13_15.err  lu.s.10-Aug-13_51.out  run.ep.sa.D.8x4.err    run.lu.sb.D.8x4.out
ep.s.10-Aug-13_15.out  lu.s.10-Aug-14_22.err  run.ep.sa.D.8x4.out    totaltime
ep.s.10-Aug-13_52.err  lu.s.10-Aug-14_22.out  run.ep.sb.D.8x4.err
ep.s.10-Aug-13_52.out  lu.s.10-Aug-14_54.err  run.ep.sb.D.8x4.out
ep.s.10-Aug-14_27.err  lu.s.10-Aug-14_54.out  run.is.sa.D.8x4.err
```

Σχήμα 22: myAlgo results example

Η ουρά των 30 εργασιών παρουσιάζει αποτελέσματα που έχουν μορφή παρόμοια με την παραπάνω.

Αριθμητικά τα αποτελέσματα είναι τα εξής:

Queue 10

Default: 3 ώρες και 20 λεπτά

myAlgo: 2 ώρες και 58 λεπτά

Βελτίωση 12%

Queue 30

Default: 10 ώρες και 12 λεπτά

myAlgo: 9 ώρες και 22 λεπτά

Βελτίωση 9%

Σε συνδυασμό και με άλλα τρεξίματα σε ουρά 20 εργασιών και σε άλλη των 30, παρατηρήθηκε μέση βελτίωση 10%. Συνεπώς ο αλγόριθμος μου εξοικονομεί 10% χρόνο στο σύστημα και στον χρήστη, χρησιμοποιώντας τους ίδιους πόρους με τον Default του Torque αλλάζοντας την πολιτική από compact σε stripe και εκμεταλλεύοντας τις συνεργασίες των προβλημάτων.

Στην χειρότερη περίπτωση, ο αλγόριθμος μου μπορεί να βγάλει αποτέλεσμα ίσο με τον Default του Torque. Αυτό συμβαίνει καθώς αν η ουρά περιέχει μόνο προβλήματα ίδιου τύπου που δεν συμφέρει να γίνουν stripe μεταξύ τους, όπως για παράδειγμα μια ουρά που περιέχει 10 IS (γραφική 107), τότε ο αλγόριθμός μου θα τα βάλει στην ουρά compact και συνεπώς θα γίνει ισοδύναμος με τον αυτόματο.

Στην καλύτερη περίπτωση ο αλγόριθμος πετυχαίνει ποσοστό βελτίωσης ίσο με την μέγιστη δυνατή συνεργασία, που είναι μεγαλύτερο από 10%. Αν το cluster της σχολής μας επέτρεπε να χρησιμοποιήσουμε ουρές που περιέχουν και προβλήματα SP και MG τα οποία συνεργάζονται πολύ καλά με τα υπόλοιπα όπως φαίνεται και στις γραφικές 109 110, το βέλτιστο αποτέλεσμα θα μπορούσε να είναι αρκετά υψηλότερο.

6.4 Slurm

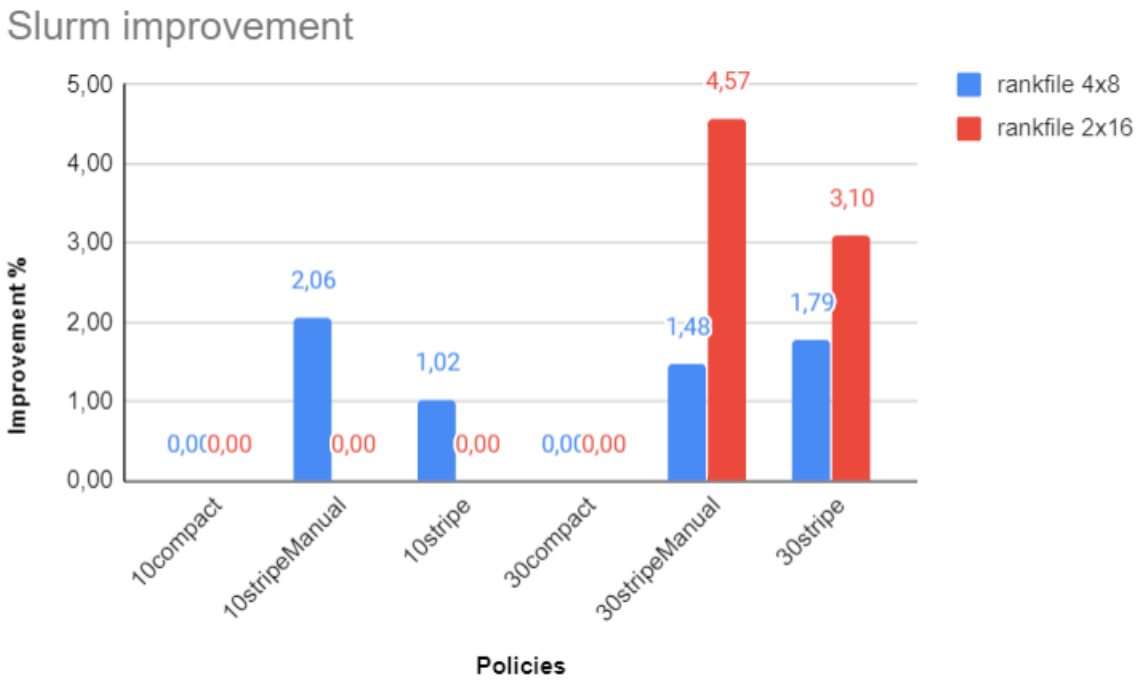
Στην ενότητα αυτή, έγινε προσπάθεια να τρέξουν τα προβλήματα σε ένα άλλο σύστημα για να συγκριθούν τα αποτελέσματα με το σύστημα της σχολής. Τα προβλήματα έτρεξαν στο ARIS, τον υπερυπολογιστή της Ελλάδας που χρησιμοποιεί το Slurm.

Το Slurm Workload Manager, Simple Linux Utility for Resource Management (SLURM), είναι ένας δωρεάν και ανοιχτού κώδικα χρονοδρομολογητής εργασιών που χρησιμοποιείται σε Linux πυρήνες και χρησιμοποιείται από πολλούς υπερυπολογιστές και πολλά cluster υπολογιστών. Το Slurm χρησιμοποιείται σε περίπου 60% των υπερυπολογιστών από τη λίστα TOP500 [1]. Χρησιμοποιεί έναν αλγόριθμο καλύτερου ταιριάσματος που βασίζεται στη χρονοδρομολόγηση της καμπύλης Hilbert ή στην τοπολογία ενός fat tree δικτύου, προκειμένου να βελτιστοποιηθεί η ανάθεση εργασιών σε παράλληλους υπολογιστές. Περισσότερα για το Slurm θα βρείτε στο [19].

Στο σύστημα του ARIS, έτρεξαν οι δύο ουρές της ενότητας 6.2, δηλαδή η ουρά των 10 εργασιών και η ουρά των 30 εργασιών, που είχαν τα προβλήματα της κλάσης D για 32 πυρήνες. Το ίδιο σενάριο δηλαδή που εξετάστηκε και στις προηγούμενες παραγράφους. Επειδή το hardware του ARIS ήταν διαφορετικό, και τα μηχανήματά του είχαν 20 πυρήνες αντί για 8 που είχε το cluster της σχολής, δοκιμάστηκαν δύο διαφορετικά rankfiles. Το rankfile 4x8 είναι αυτό που χρησιμοποιούνταν και στο τοπικό cluster του εργαστηρίου και το rankfile 2x16 είναι το προσαρμοσμένο για το σύστημα του ARIS ώστε να γεμίζουν πιο σωστά οι πυρήνες των μηχανημάτων.

Παρουσιάζονται σε σχήμα οι βελτιώσεις στους χρόνους που έγιναν από τα τρεξίματα στο slurm. Χρησιμοποιούνται τρεις πολιτικές για κάθε ένα από τα δύο παραπάνω rankfiles. Η πρώτη πολιτική είναι η compact, η δεύτερη είναι η manual stripe όπου δηλαδή γίνονται stripe τα προβλήματα με τη σειρά που είναι στην ουρά υποβολής με τις εργασίες, και η τρίτη είναι η stripe που είναι η ουρά που παράγει ο

αλγόριθμος μου και περιέχει ζευγάρια stripe σύμφωνα με τις συνεργασίες των προβλημάτων από την μελέτη που έχει γίνει στο κεφάλαιο 5, στο σύστημα της σχολής. Η βελτίωση αναφέρεται ως προς την πολιτική compact, συνεπώς η compact ως προς τον εαυτό της έχει 0% βελτίωση.



Γραφικές 111: Slurm improvement

Στο slurm, παρατηρούμε στα δύο διαφορετικά rankfile πως οι αλγόριθμοι stripe δεν έχουν ιδιαίτερη βελτίωση σε σχέση με τον compact. Το μεγαλύτερο αποτέλεσμα είναι το 4% στην ουρά των 30 εργασιών. Αυτό γίνεται γιατί τα προβλήματα δε συμπεριφέρονται όπως στο Torque. Πρέπει να γίνει μελέτη των προβλημάτων από την αρχή και των πιθανών τους συνεργασίων σε αυτή και ίσως και σε μεγαλύτερη κλάση, ώστε να φτάσει ο υπερυπολογιστής στα όρια του και τα προβλήματα να αποδίδουν αισθητά καλύτερα σε stripe πολιτική όπως έγινε και στο τοπικό cluster. Αυτό γίνεται γιατί τα προβλήματα στο Torque συμπεριφέρονταν αλλιώς λόγο διαφορετικού υλικού σε σχέση με το Slurm. Μάλιστα στο slurm, το manual stripe φαίνεται να είναι ισοδύναμο και ελαφρώς καλύτερο από τον αλγόριθμό μου καθώς οι συνεργασίες και οι περιορισμοί των προβλημάτων που είχαν μελετηθεί στο Torque δεν ισχύουν εδώ και συνεπώς ο αλγόριθμός μου γίνεται ισοδύναμος με το manual stripe.

7. Επίλογος

7.1 Αξιολόγηση

Στην εργασία αυτή μελετήθηκαν σε σημαντικό βαθμό τα Nas Parallel Benchmarks. Τα προβλήματα αυτά αναλύθηκαν στις κλάσεις A,B,C και D για 8,16,32 και 64 πυρήνες. Χρησιμοποιήθηκαν οι πολιτικές χρονοδρομολόγησης compact, spread και stripe.

Τα προβλήματα έτρεξαν στο cluster της σχολής με τη χρήση του διαχειριστή κατανεμημένων πόρων Torque που είναι ένας από τους πιο διαδεδομένους χρονοδρομολογητές για μεγάλα υπολογιστικά συστήματα. Φάνηκε πως οι κλάσεις A και B χρήζουν μελέτης με λιγότερους πυρήνες, όπως για παράδειγμα 1,2 έως 8 ή έως 16. Η κλάση C φαίνεται να ταιριάζει στο σύστημα της σχολής ωστόσο και αυτή χρήζει μελέτης με λιγότερους επεξεργαστές και πιο ισχυρούς. Η κλάση D, για την οποία δημιουργήθηκε και ο απλός αλγόριθμος που διαβάζει μια ουρά από εργασίες προς εκτέλεση και αποφασίζει και παράγει μια νέα ουρά με τις εργασίες αυτές σε stripe και compact πολιτική, είναι αυτή που έχει την πιο σωστή επιτάχυνση αυξάνοντας τους πυρήνες στο σύστημα της σχολής. Ωστόσο, η κλάση αυτή είναι ιδιαίτερα απαιτητική και τα NPB προβλήματα δεν έτρεχαν στο σύστημα της σχολής για τους περισσότερους συνδυασμούς πυρήνων.

Η μελέτη έγινε για τα προβλήματα της κλάσης D με 32 πυρήνες, ωστόσο αντίστοιχη λογική μπορεί να αναπτυχθεί και για περισσότερους συνδυασμούς πυρήνων αν υπήρχαν τα δεδομένα και τα προβλήματα ήταν εφικτό να τρέξουν στο σύστημα της σχολής.

Ταυτόχρονα, στην παρούσα εργασία αποκτήθηκε εξοικείωση με τη γλώσα bash και τη δημιουργία διαφόρων αρχείων – scripts για υποβολή εργασιών σε κάποιον χρονοδρομολογητή.

Το αποτέλεσμα της διπλωματικής και το συνολικό συμπέρασμα, δείχνει πως μπορεί να αναπτυχθεί αλγόριθμος χρονοδρομολόγησης καλύτερος από τον αυτόματο που έχουν οι μεγάλοι και γνωστοί χρονοδρομολογητές, ο οποίος συνήθως είναι map by core. Στην παρούσα εργασία παρατηρήθηκε βελτίωση του συνολικού χρόνου κατά μέσο όρο 10%, που σε πραγματικά συστήματα που προβλήματα μπορεί να τρέχουν για μέρες ή και μήνες είναι πολύ σημαντική. Η μελέτη αν το σύστημα της

σχολής είχε τη δυνατότητα να ανταπεξέλθει στις απαιτήσεις των προβλημάτων της κλάσης D πλήρως, ενδέχεται να είχε ακόμα καλύτερα αποτελέσματα. Ωστόσο, δεδομένου ότι αποτελεί ένα ελεύθερο υπολογιστικό σύστημα για τους ερευνητές της σχολής το οποίο μοιράζεται σε αυτούς, το αποτέλεσμα είναι ικανοποιητικό.

7.2 Μελλοντικές επεκτάσεις – Δυνατότητες

Ο αλγόριθμος που αναπτύχθηκε αφορούσε μόνο τέσσερα από τα οκτώ προβλήματα της κλάσης D των Nas Parallel Benchmark. Για τον λόγο αυτό, ο αλγόριθμος δεν ήταν γενικός και αναπτύχθηκε πάνω στα αποτελέσματα από τα τρεξίματα της κλάσης D.

Σαν επέκταση στον τομέα αυτό, θα μπορούσε να αναπτυχθεί ένας γενικός αλγόριθμος co-scheduling, που θα διαβάζει από έναν πίνακα τους χρόνους των προβλημάτων και θα αποφασίζει αυτός τον τρόπο και την πολιτική που πρέπει να εκτελεστούν. Φυσικά αυτό στον πραγματικό κόσμο είναι δύσκολο καθώς δεν γνωρίζει κανείς τους χρόνους αυτούς παρά μόνο αν έχει ήδη τρέξει τα προβλήματα.

Συνεπώς, για να μπορέσει να επωφεληθεί κανείς από πιθανούς καλούς συνδυασμούς προβλημάτων, οι τρόποι είναι δύο. Ο πρώτος είναι ο ερευνητής που υποβάλει τα προβλήματα του σε έναν υπερυπολογιστή, να γνωρίζει τι είδους είναι το κάθε πρόβλημα, όπως για παράδειγμα compute intensive, memory intensive, communication intensive κ.α ώστε να αποφασίσει ο ίδιος πως αξίζει για παράδειγμα ένα πρόβλημα compute intensive να συνδυαστεί με ένα communication intensive για να τρέξουν καλύτερα μαζί από το να έτρεχαν το καθένα compact, όπως έγινε σε αυτή τη διπλωματική για παράδειγμα με το EP και το IS. Ωστόσο αυτό είναι δύσκολο να το γνωρίζει ο ίδιος ο ερευνητής, καθώς δύσκολα ένα πρόγραμμα είναι αμιγώς μιας κατηγορίας. Συνήθως ανήκει σε περισσότερες κατηγορίες, απλώς σε μία ενδέχεται να εστιάζουν οι απαιτήσεις του. Στην διπλωματική αυτή, το EP για παράδειγμα ήταν ένα καθαρά computing intensive πρόγραμμα, αλλά ήταν το μόνο. Τα υπόλοιπα ήταν ένας συνδυασμός computing, memory και communication intensive.

Ο δεύτερος τρόπος για να καταφέρει κάποιος να εφαρμόσει έναν καλύτερο αλγόριθμο χρονοδρομολόγησης είναι να καταφέρει να μάθει ο ίδιος ο αλγόριθμος τα χαρακτηριστικά των προβλημάτων που τρέχουν στον υπερυπολογιστή. Για να γίνει αυτό, θα πρέπει τα προβλήματα να επαναλαμβάνονται παραπάνω από μία φορές. Με

αυτό τον τρόπο, ο αλγόριθμος αρχίκα θα τρέξει τα προβλήματα με την default πολιτική του και αναλόγως τα αποτελέσματα και τις απαιτήσεις των προβλημάτων θα μπορέσει να τα κατατάξει σε κατηγορίες και να φτιάξει υποθετικούς συνδυασμούς τους. Κάτι τέτοιο φυσικά είναι αρκετά δύσκολο, καθώς ο αλγόριθμος ή καλύτερα ο χρονοδρομολογητής πρέπει να έχει γνώση και συνεργασία με την αρχιτεκτονική του συστήματος.

Βιβλιογραφία

[1] "Top500. The List," [Online]. Available: https://www.top500.org/lists/top500/2020/06/

[2] Eadline, D., PhD. (n.d.). *High Performance Computing For Dummies* (Sun and AMD Special Edition ed.). 111 River Street: Wiley Publishing.

[3] Wikipedia, "Flynn's Taxonomy." [Online]. Available: https://en.wikipedia.org/wiki/Flynn's_taxonomy

[4] Duncan, R. (1990). *A Survey of Parallel Computer Architectures*. Survey & Tutorial Series.

[5] Wikipedia, "Amdahl's Law." [Online]. Available: https://en.wikipedia.org/wiki/Amdahl%27s_law

[6] S. Iqbal, R. Gupta, and Y. Fang, "Planning Considerations for Job Scheduling in HPC Clusters," Dell Power Solutions, Tech. Rep., pp. 133–136, 2005.

[7] Wikipedia, "Scheduling (Computing)." [Online]. Available: https://en.wikipedia.org/wiki/Scheduling_(computing)#Scheduling_disciplines

[8] Wikipedia, "Job Scheduler." [Online]. Available: https://en.wikipedia.org/wiki/Job_scheduler

[9] Aida, K. (n.d.). *Effect of Job Size Characteristics on Job Scheduling Performance* (Vol. Department of Computational Intelligence and Systems Science). JAPAN.

[10] Wikipedia, "Job Shop Scheduling." [Online]. Available: https://en.wikipedia.org/wiki/Job_shop_scheduling

[11] Breslow, A. D., Porter, L., Tiwari, A., Laurenzano, M., Carrington, L., Tullsen, D. M., & Snavely, A. E. (2013). *The case for colocation of high performance computing workloads* (Vol. SPECIAL ISSUE PAPER). Wiley Online Library. doi:10.1002/cpe.3187

[12] Wikipedia, "TORQUE." [Online]. Available: https://en.wikipedia.org/wiki/TORQUE

[13] HPC wiki, "TORQUE." [Online]. Available: https://hpc-wiki.info/hpc/Torque

[14] qsub man [Online]. Available: http://docs.adaptivecomputing.com/torque/4-0-2/Content/topics/commands/qsub.htm

[15] mpirun man [Online]. Available: https://www.open-mpi.org/doc/v4.0/man1/mpirun.1.php

[16] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dangum, L., Fatoohi, R., Fineberg, S., Frederichson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnam, V., Weeratunga, S. (March 1994). *THE NAS PARALLEL BENCHMARKS*. RNR Techical Report. [Online]. Available: https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf

[17] NASA, "NASA Advanced Supercomputing Division." [Online]. Available: https://www.nas.nasa.gov/publications/npb.html

[18] Subhlok, J., Venkataramaiah, S., & Singh, A. (2002). *Characterizing NAS Benchmark Performance on Shared Heterogeneous Networks* (Vol. Department of Computer Science University of Houston).

[19] Wikipedia, "Slurm Workload Manager." [Online]. Available: https://en.wikipedia.org/wiki/Slurm_Workload_Manager

ΠΑΡΑΡΤΗΜΑ

Η τοπολογία

Τα πειράματα έγιναν στο σύστημα της σχολής του εργαστηρίου Υπολογιστικών Συστημάτων, κάτω από τον @scirouter.cslab.ece.ntua.gr.

Με την εντολή ***uname -a*** παίρνουμε πληροφορίες για το όνομα του συστήματος, το hostname του, το version και το release του πυρήνα και το όνομα της αρχιτεκτονικής του μηχανήματος. Τα αποτελέσματα είναι τα εξής :

Linux scirouter 2.6.26-2-amd64 #1 SMP Sun Mar 4 21:48:06 UTC 2012 x86_64 GNU/Linux

```
=============================================
                         system     Computer
/0                       bus        Motherboard
/0/0                     memory     2047MiB System memory
/0/2                     processor  AMD Opteron(tm) Processor 248
/0/3                     processor  AMD Opteron(tm) Processor 248
/0/1                     bridge     BCM5785 [HT1000] PCI/PCI-X Bridge
/0/1/d                   bridge     BCM5785 [HT1000] PCI/PCI-X Bridge
/0/1/d/6                 bus        Helios LightPulse Fibre Channel Host Adapter
/0/1/e                   storage    BCM5785 [HT1000] SATA (PATA/IDE Mode)
/0/1/e.1                 storage    BCM5785 [HT1000] SATA (PATA/IDE Mode)
/0/100                   bridge     BCM5785 [HT1000] Legacy South Bridge
/0/100/2.1               storage    BCM5785 [HT1000] IDE
/0/100/2.1/0     ide0    bus        IDE Channel 0
/0/100/2.1/0/1   /dev/hdb disk      MATSHITADVD-ROM SR-8178
/0/100/2.2               bridge     BCM5785 [HT1000] LPC
/0/100/3                 bus        BCM5785 [HT1000] USB
/0/100/3/1       usb1    bus        OHCI Host Controller
/0/100/3.1               bus        BCM5785 [HT1000] USB
/0/100/3.1/1     usb2    bus        OHCI Host Controller
/0/100/3.2               bus        BCM5785 [HT1000] USB
/0/100/3.2/1     usb3    bus        EHCI Host Controller
/0/100/5                 display    Rage XL
/0/100/6         eth2    network    82541GI Gigabit Ethernet Controller
/0/100/8                 bridge     BCM5780 [HT2000] PCI-X bridge
/0/100/9                 bridge     BCM5780 [HT2000] PCI-X bridge
/0/100/9/4       eth0    network    NetXtreme BCM5780 Gigabit Ethernet
/0/100/9/4.1     eth1    network    NetXtreme BCM5780 Gigabit Ethernet
/0/100/a                 bridge     BCM5780 [HT2000] PCI-Express Bridge
/0/100/b                 bridge     BCM5780 [HT2000] PCI-Express Bridge
/0/100/c                 bridge     BCM5780 [HT2000] PCI-Express Bridge
/0/100/c/0               storage    Areca Technology Corp.
/0/100/d                 bridge     BCM5780 [HT2000] PCI-Express Bridge
/0/101                   bridge     K8 [Athlon64/Opteron] HyperTransport Technology Configuration
/0/102                   bridge     K8 [Athlon64/Opteron] Address Map
/0/103                   bridge     K8 [Athlon64/Opteron] DRAM Controller
/0/104                   bridge     K8 [Athlon64/Opteron] Miscellaneous Control
/0/105                   bridge     K8 [Athlon64/Opteron] HyperTransport Technology Configuration
/0/106                   bridge     K8 [Athlon64/Opteron] Address Map
/0/107                   bridge     K8 [Athlon64/Opteron] DRAM Controller
/0/108                   bridge     K8 [Athlon64/Opteron] Miscellaneous Control

=============================================
```

Σχήμα 23: Scirouter hardware

Με την εντολή ***lshw -short*** παίρνουμε πληροφορίες για το hardware του scirouter που φαίνονται στην εικόνα 23.

Για τα πειράματα χρησιμοποιήθηκε η ουρά των clones κάτω από των scirouter, η οποία φαίνεται στο ακόλουθο σχήμα.

NodeName	Taken	Load	Mem(MB)	FreeMem	State	Attributes	Users(Slots)
clone1	0/8	0.09	7983	98.69%	Empty	clones,mpi,disk,clover,highmem,parlab,sniper	
clone2	0/8	0.18	7983	98.74%	Empty	clones,mpi,disk,clover,highmem,parlab,sniper	
clone3	0/8	0.05	7983	98.74%	Empty	clones,mpi,disk,clover,highmem,parlab,sniper	
clone4	0/8	0.26	7983	98.74%	Empty	clones,mpi,disk,clover,highmem,parlab,sniper	
clone5	0/8	0.13	7983	98.76%	Empty	clones,mpi,disk,clover,highmem,parlab	
clone6	0/8	0.10	3951	97.71%	Empty	clones,mpi,mx10g,disk,clover,lowmem,kvm,parlab	
clone7	0/8	0.17	3951	97.66%	Empty	clones,mpi,mx10g,clover,lowmem,collab,parlab	
clone8	0/8	0.14	2000	95.77%	Empty	clones,mpi,disk,clover,lowmem,collab,parlab	
clone9	0/8	0.26	3008	97.06%	Empty	clones,collab,mpi,disk,clover,lowmem,parlab	
clone10	0/8	0.16	2000	95.93%	Empty	clones,collab,mpi,disk,clover,lowmem,parlab	
clone11	0/8	0.03	5967	98.34%	Empty	clones,mx10g,collab,mpi,disk,clover,lowmem,parlab	
clone12	0/8	0.10	3951	97.71%	Empty	clones,mpi,disk,clover,lowmem,parlab	
clone13	0/8	0.15	7983	98.79%	Empty	clones,mpi,disk,clover,highmem,parlab,sniper	
clone14	0/8	0.20	7983	98.77%	Empty	clones,mpi,disk,clover,highmem,sniper,parlab	
clone15	0/8	0.21	7983	98.75%	Empty	clones,mpi,disk,clover,highmem,sniper,parlab	
clone16	0/8	0.06	7983	98.77%	Empty	clones,mpi,disk,clover,highmem,sniper,parlab	
clone19	0/8	0.11	7983	98.77%	Empty	clones,mpi,mx10g,disk,harper,highmem,sniper,parlab	
clone20	0/8	0.18	7983	98.78%	Empty	clones,mpi,mx10g,disk,harper,highmem,sniper,parlab	
clone25	0/8	0.09	7983	98.79%	Empty	clones,mpi,harper,disk,highmem,sniper	
clone26	0/8	0.17	7983	98.80%	Empty	clones,mpi,disk,harper,highmem,sniper	

Σχήμα 24: Scirouter Clones Queue

Η ουρά αποτελείται από 20 clones τα οποία είναι τα clone1-clone16, clone19, clone20, clone25, clone26. Το κάθε ένα από τα nodes αυτά, έχει 8 πυρήνες που μοιράζονται σε 2 sockets των 4 πυρήνων το κάθε ένα.

Εκτελώντας τις εντολές που κάναμε παραπάνω στον scirouter και για τα clones παίρνουμε αναλυτικά:

uname -a

Linux clone1 4.0.4-1-eangelou #1 SMP Thu May 21 10:37:35 EEST 2015 x86_64 GNU/Linux

lsblk

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 465.8G  0 disk
`-sda1   8:1    0 465.8G  0 part /local
```

Σχήμα 25: Block devices informations

Με την εντολή ***lscpu*** παίρνουμε αναλυτικά τα χαρακτηριστικά του επεξεργαστή των nodes και παρατηρείται πως χρησιμοποιούν NUMA αρχιτεκτονική προσπέλαση μνήμης.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             2
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 15
Model name:            Intel(R) Xeon(R) CPU           E5335  @ 2.00GHz
Stepping:              7
CPU MHz:               2000.106
BogoMIPS:              4000.47
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
NUMA node0 CPU(s):     0-7
```

Σχήμα 26: Clones CPU

NPB

Για την εκτέλεση των προγραμμάτων χρησιμοποιήθηκε η έκδοση openmpi1.8.3. Κάνοντάς το compile παράχθηκε ο φάκελος mpi1 τον οποίο χρησιμοποιούσαν τελικά τα εκτελέσιμα. Για να συμβεί αυτό γινόταν, μέσα στον κώδικα των αρχείων, export τα κατάλληλα path για την εκτέλεσή τους, όπως φαίνεται στην παρακάτω εικόνα.

```
export PATH=/various/common_tools/gcc-4.5.2/bin:$PATH
export PATH=/home/users/ppeppas/mpi1/bin:$PATH
export LD_LIBRARY_PATH=/various/common_tools/gcc-4.5.2/lib64/
```

Σχήμα 27: Mpi1 export

Η έκδοση των Nas Parallel Benchmark που χρησιμοποιήθηκε στην παρούσα εργασία είναι η πιο πρόσφατη έως τώρα και είναι η NPB.3.4. Κατέβηκε από την επίσημη ιστοσελίδα της NASA [17]. Όταν το αρχείο αποσυμπιέστηκε, παρήχθησαν δύο φάκελοι, οι NPB3.4-MPI και NPB3.4-OMP και αρκετά readme file αρχεία που βοηθούσαν για το compile των προβλημάτων και περιείχαν πληροφορίες σχετικά με τις αλλαγές τους από προηγούμενες εκδόσεις.

Μέσα στον φάκελο NPB3.4-MPI υπάρχουν τα ακόλουθα αρχεία:

```
bin  BT  CG  common  config  DT  EP  FT  IS  LU  Makefile  MG  MPI_dummy SP  sys  test_scripts
```

Σχήμα 28: NPB3.4-MPI files

Στα αρχεία αυτά υπάρχουν τα προγράμματα των NPB, ένα Makefile για να γίνουν compile και άλλα αρχεία με συναρτήσεις που χρησιμοποιούνται. Για την μεταγλώττιση των προγραμμάτων, το Makefile είναι το αρχείο που καλείται και δεν χρειάζεται αλλαγή. Στον φάκελο common υπάρχουν διάφορες συναρτήσεις χρήσιμες για τα προγράμματα, όπως συναρτήσεις για χρόνους, για εκτύπωση των αποτελεσμάτων σε αναγνώσιμη μορφή, συναρτήσεις παραγωγής random αριθμών κ.α. Στον φάκελο config, υπάρχουν τα αρχεία make.def.template και suite.def.template, τα οποία αντιγράφουμε σε make.def και suite.def και τα οποία καλούνται από το Makefile. Το make.def πρέπει να ρυθμιστεί με τις σωστές εκδόσεις των compilers που θα κάνουν compile τα προγράμματα αν αυτό είναι απαραίτητο. Το suite.def περιλαμβάνει τα προβλήματα και τις κλάσεις για τα οποία επιθυμούμε να παραχθούν εκτελέσιμα και συνεπώς θέλει αλλαγή.

Παράδειγμα του suite.def για την κλάση A είναι το ακόλουθο:

```
# config/suite.def
# This file is used to build several benchmarks with a single command.
# Typing "make suite" in the main directory will build all the benchmarks
# specified in this file.
# Each line of this file contains a benchmark name, class, and number
# of nodes. The name is one of "cg", "is", "ep", mg", "ft", "sp", "bt",
# "lu", and "dt".
# The class is one of "S", "W", "A", "B", "C", "D", and "E"
# (except that no classes C, D and E for DT, and no class E for IS).
# The number of nodes must be a legal number for a particular
# benchmark. The utility which parses this file is primitive, so
# formatting is inflexible. Separate name/class by tabs.
# Comments start with "#" as the first character on a line.
# No blank lines.
# The following example builds sample sizes of all benchmarks.
ft      A
mg      A
sp      A
lu      A
bt      A
is      A
ep      A
cg      A
```

Σχήμα 29: suite.def file

Το συνολικό αρχείο suite.def απλώς περιέχει τα 8 προβλήματα άλλες 3 φορές για τις κλάσεις B,C,D.

Έχοντας φτιάξει αυτά τα αρχεία και αφού γίνουν τα export της εικόνας 27 σε κάποιο clone του scirouter, τότε με την εντολή ***make suite*** παράγονται τα εκτελέσιμα αρχεία στον φάκελο bin.

Ακριβώς η ίδια διαδικασία ακολουθείται και για τον NPB3.4-OMP φάκελο, με τα openmp εκτελέσιμα να πηγαίνουν στον bin φάκελο ομοίως.

Η εικόνα του φακέλου bin με τα τελικά εκτελέσιμα που χρησιμοποιούνται στη μελέτης της εργασίας και καλούνται από τα mpirun αρχεία είναι η ακόλουθη:

```
bt.A.x  bt.D.x  cg.C.x  dt.B.x  ep.A.x  ep.D.x  ft.C.x  is.B.x  lu.A.x  lu.D.x  mg.C.x  sp.B.x
bt.B.x  cg.A.x  cg.D.x  dt.C.x  ep.B.x  ft.A.x  ft.D.x  is.C.x  lu.B.x  mg.A.x  mg.D.x  sp.C.x
bt.C.x  cg.B.x  dt.A.x  dt.D.x  ep.C.x  ft.B.x  is.A.x  is.D.x  lu.C.x  mg.B.x  sp.A.x  sp.D.x
```

Σχήμα 30: Bin Εκτελέσιμα

Κώδικας

Παρουσιάζεται ο κώδικας του αλγορίθμου σε python.

```python
import sys

# read file as argument
if (len(sys.argv) != 2):
    print("Give one file as argument! ")
    sys.exit(0)

# open file to read it line by line
file1 = open(sys.argv[1],"r")
Lines = file1.readlines()

# counters for the problems
CG = 0
EP = 0
IS = 0
LU = 0
MG = 0
SP = 0

for line in Lines:
    if (line.strip() == "cg.D.32"):
        CG+=1
    elif (line.strip() == "ep.D.32"):
        EP+=1
    elif (line.strip() == "is.D.32"):
        IS+=1
    elif (line.strip() == "lu.D.32"):
        LU+=1
    elif (line.strip() == "mg.D.32"):
        MG+=1
    elif (line.strip() == "sp.D.32"):
        SP+=1
    else:
        print ("The << " + str(line.strip()) + " >> is not a problem we can manage")
        sys.exit(0)

result = []
compact = []
stripe = []
```

Κώδικας 13: myAlgo 1

```
if (IS > 0):
    while (EP > 0 and IS > 0):
        stripe.append("ISsa")
        stripe.append("EPsb")
        IS-=1
        EP-=1
    if (IS > 0):
        while (LU > 0 and IS > 0):
            stripe.append("ISsa")
            stripe.append("LUsb")
            IS-=1
            LU-=1

# if there are no more EP and LU, make compact all IS and CG remaining cause IS with CG is not a good match
if (IS > 0):
    while (IS > 0):
        compact.append("ISc")
        IS-=1
    while (CG > 0):
        compact.append("CGc")
        CG-=1

# stripe CG applications with EP and after with LU
if (CG > 0):
    while (EP > 0 and CG > 0):
        stripe.append("EPsa")
        stripe.append("CGsb")
        EP-=1
        CG-=1
    if (CG > 0):
        while(LU > 0 and CG > 0):
            stripe.append("LUsa")
            stripe.append("CGsb")
            LU-=1
            CG-=1
```

Κώδικας 14: myAlgo 2

```
if (CG > 0):
    while (CG > 0):
        compact.append("CGc")
        CG-=1

# stripe LU with EP
if (LU > 0):
    while(LU > 0 and EP > 0):
        stripe.append("LUsa")
        stripe.append("EPsb")
        LU-=1
        EP-=1
    # stripe LU with remaining LU cause LU-LU is a good match
    if (LU > 0):
        temp = 0
        while(LU > 0):
            if (temp==0):
                stripe.append("LUsa")
                LU-=1
                temp=1
            elif (temp==1):
                stripe.append("LUsb")
                LU-=1
                temp=0
```

Κώδικας 15: myAlgo 3

```
124  # Here we have remaining only EP applications
125  # EP make time a bit better if it will be compact, so if EP%2==0, better compact
126  # if EP%2==0, better compact
127  # if EP%2==1, better stripe, so first EP application that will be assinged will no wait for previous spare application to end
128 if (EP > 0):
129     temp=1
130     if (EP%2 == 0):
131         while (EP > 0):
132             compact.append("EPc")
133             EP-=1
134     else:
135         while (EP > 0):
136             if (temp==0):
137                 stripe.append("EPsa")
138                 EP-=1
139                 temp=1
140             elif (temp==1):
141                 stripe.append("EPsb")
142                 EP-=1
143                 temp=0
144
```

Κώδικας 16: myAlgo 4

```
148  # concat two list in one
149 for st in stripe:
150     result.append(st)
151
152 for cm in compact:
153     result.append(cm)
154
155  print("result: ", result)
156
157  # write the result to a new file named queue
158  f=open('queue','w')
159 for element in result:
160     f.write(element+'\n')
161
162  f.close()
```

Κώδικας 17: myAlgo 5