HPC - Principal Component Analysis

Anti-gravitational Forces Computation

Η Εργασία εκπονήθηκε από τους:

- Σφήκας Θεόδωρος, 1072550
- Τσούλος Βασίλειος, 1072605

❖ Εισαγωγή:

System Specifications:

Το πρόγραμμα για την παράδοση των τελικών αποτελεσμάτων εκτελέστηκε σε επεξεργαστή *Ryzen 5600x* και σε σύστημα με *16GB RAM 3200Mhz*. Ακολουθούν περαιτέρω διευκρινίσεις πάνω στα χαρακτηριστικά του επεξεργαστή:

» AMD Ryzen 5 Desktop Processor:

# of CPU Cores → 6
# of Threads → 12
Max. Boost Clock→ Up to 4.6GHz *(ενεργό)*
Base Clock→ 3.7GHz
L1 Cache → 64 KB *(per core)*
L2 Cache → 512 KB *(per core)*
L3 Cache→ 32 MB *(shared)*

Οδηγίες εκτέλεσης:

Στον φάκελο που παραδώθηκε, εμπεριέχονται δύο υπό-φάκελοι για τα parts της εργαστηριακής άσκησης. Σε κάθε ένα από αυτά εμπεριέχεται ο source code, makefile for compilation και ένα run.sh bash script για την αυτοματοποιημένη εκτέλεση των προγραμμάτων.

Παραδοχές:

1) Για τον πειραματισμό με τα BLAS και την LAPACKE χρησιμοποιήθηκαν τα openblas, cblas, lapack και lapacke πακέτα από Linux repositories.
2) Κατά υλοποίηση του πρώτου part, η principal component analysis λειτουργεί ορθά για Column major μητρώα εισόδου. Θεωρώντας δηλαδή την αρχή αποθήκευσης των features της εικόνας ως Row major δημιουργεί πρόβλημα στο κάλεσμα της συνάρτησης LAPACKE_dsyev το οποίο δεν προλάβαμε να λύσουμε λόγω χρονικής πίεσης. Επεξηγηματικά καθώς διαβάζουμε την εικόνα στην μνήμη ως ένα packed μητρώο δύο διαστάσεων είναι στο χέρι μας να επιλέξουμε εάν στο μητρώο τα features είναι τα rows ή τα columns και κατά συνέπεια να επιλέξουμε αντίστοιχη μορφή αποθήκευσης. Εάν θεωρήσουμε πως τα features είναι τα rows τότε θεωρούμε πως τα observations του μητρώου είναι τα columns και αντίστροφα.
3) Παρά την παραπάνω σημείωση, έγινε προσπάθεια δημιουργίας δυναμικών των συναρτήσεων και για τις δύο περιπτώσεις εισόδου. Επιπροσθέτως έχουν δημιουργηθεί και συναρτήσεις τόσο για την manual εκτέλεση SIMD intrinsics όσο και για την χρήση παραλληλίας και SIMD αυτοματοποιημένα μέσω OPENMP directives. Ακόμα έχουν γίνει πειραματισμός για την αντικατάσταση καλεσμάτων και συναρτήσεων της LAPACK με δικές μας όπου θεωρήσαμε εφικτό. Ωστόσο λόγω χρόνου δεν έχει επιτευχθεί κατάλληλο optimization και κατά συνέπεια χρήση ορισμένων.

❖ Part A

Αρχικά κρίνουμε σημαντικό να εξηγήσουμε συνοπτικά την αλγοριθμική διαδικασία του principal component analysis. Σαν Input διαβάζουμε μία ασπρόμαυρη εικόνα της οποίας τα pixel αντιστοιχίζουν σε ένα MxN μητρώο. Όπως αναφέραμε και προηγουμένως θεωρούμε τα features του μητρώου ως τα columns. Με στόχο λοιπόν να ολοκληρώσουμε την διαδικασία στην συνέχεια cache efficiently κάνουμε ένα αρχικό transpose του μητρώου μας αποθηκεύοντας το ενεργά σε column major order. Κάνουμε normalize τα features διαιρώντας με την τυπική απόκλιση και αφαιρώντας την μέση τιμή. Ως εκτούτου κάνουμε centralize and gather τα δεδομένα μας κοντά στην αρχή νοητών αξόνων. Έπειτα βρίσκουμε το covariance matrix (μητρώο ίδιο-συσχέτισης) και ύστερα ψάχνουμε τα X eigenvectors τα οποία αντιστοιχούν στις X μεγαλύτερες eigenvalues. Ουσιαστικά ψάχνουμε ένα πολυδιάστατο σύστημα αξόνων το οποίο διαμορφώνεται από τα X κάθετα μεταξύ τους eigenvectors. Με στόχο να βρούμε τα principal components προβάλουμε την normalized εικόνα (μητρώο) στο προαναφερθέντα σύστημα αξόνων. Έχουμε πετύχει την lossy συμπίεση της αρχικής εικόνας και με στόχο την ανακατασκευή της πρέπει να ακολουθήσουμε την αντίστροφη διαδικασία. Να ξανά προβάλουμε δηλαδή το μητρώο των principal components στο σύστημα αξόνων και να αντιστρέψουμε την διαδικασία του normalization.
Για την παραλληλοποίηση της διαδικασίας ελπίζουμε να είμαστε συνοπτικοί καθώς αρχικά οι συναρτήσεις από τις συναρτήσεις των OpenBlas έχουν innate παραλληλοποιημένο κώδικα εξαιρώντας την συνάρτηση LAPACKE_dsyev, η οποία μάλιστα κάνοντας profiling της εφαρμογής μας παρατηρούμε πως καταλαμβάνει πάνω από τον μισό χρόνο εκτέλεσης της διαδικασίας. Η συνάρτηση δεν είναι καν thread safe από ότι αναζητήσαμε. Τα χρήση SIMD κώδικα ωστόσο έγινε όπως αναφέραμε προηγουμένως τόσο manually όσο και με την χρήση OPENMP directives. Μάλιστα μπορούμε να παρατηρήσουμε και καλύτερα αποτελέσματα στις δικές μας υλοποιήσεις και παραλληλισμούς ωστόσο τάξη μεγέθους αυτής της βελτίωσης είναι φυσικά όπως αναμένεται μικρή. Καθώς δεν είναι και αυτός ο στόχος της εργαστηριακής άσκησης δεν θα επεκταθούμε περαιτέρω με την χρήση intrinsics.
Τέλος στις συναρτήσεις που δημιουργήσαμε εμείς για τις υλοποιήσεις των ζητουμένων χρησιμοποιήσαμε OPENMP για την παραλληλοποίηση του normalization/denormalization και την εύρεση του covariance matrix, ως προς τα features. Αυτό μάλιστα γίνεται δυνατό (efficient) λόγω του αρχικού transpose που υλοποιήσαμε, αποθηκεύοντας κάθε feature στην μνήμη σειριακά πετυχαίνουμε πολύ λιγότερα cache misses.

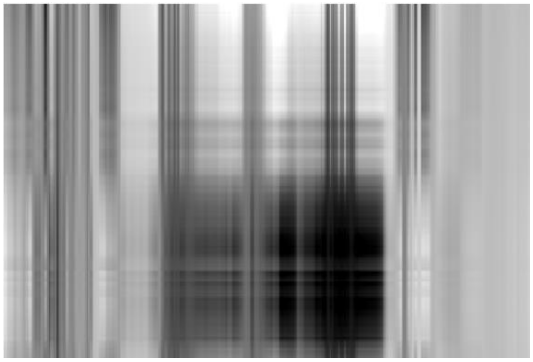
❖ Part A - Παρουσίαση αποτελεσμάτων:

Με σκοπό να υπολογίσουμε τον βαθμό συμπίεσης που πετυχαίνουμε αφού βρίσκουμε τα principal components δημιουργούμε αρχεία στα οποία αποθηκεύουμε τα eigenvectors, principal components , mean Value of each feature και τέλος το std Value of each feature και τα αποθηκεύουμε όλα σε ένα tarball συμπιεσμένο με την χρήση του gnu zip. Συγκρίνουμε το μέγεθος σε bytes της αρχικής zipped εικόνας με αυτή των αποτελεσμάτων. Η διαδικασίας συμπίεσης έχει αυτοματοποιηθεί και αρκεί να καλέσουμε την εντολή "make debug" για να παράξουμε τα ζητούμενα αποτελέσματα. Αξίζει ωστόσο να σημειωθεί πως συγκρίνουμε το μέγεθος zipped αρχείων τα οποία ευνοούν αραιά μητρώα αντί πυκνών. Παρόλα αυτά θεωρήσαμε πως είναι ο καλύτερος τρόπος να ελέγξουμε την πρακτική χρήση του κώδικα μας παρά θεωρητικά αποτελέσματα με βάση την μαθηματική συμπίεση που πετυχαίνουμε μικραίνοντας ουσιαστικά την διάσταση των μητρώων μας.

Elvis PCA	**1**	**30**	**50**	**100**	**Original**
Size in KB	8	92	148	276	416
Time (S)	0.18	0.21	0.22	0.25	

Τυπική παρουσίαση αντίστοιχων εικόνων:

Principal Components = 1



Principal Components = 30

Principal Components = 50

Principal Components = 100

❖ Συμπεράσματα και Παρατηρήσεις:

Από τα παραπάνω εύκολα συνεπάγεται η επιτυχής λειτουργία της PCA διαδικασίας. Καθώς μεγαλώνουμε τον αριθμό των eigenvectors (χρησιμοποιώντας τα eigenvectors που αντιστοιχίζονται στις μεγαλύτερες eigenvalues, ώστε να πετύχουμε τον διανυσματικό υπόχωρο με την πιο valuable πληροφορία) πετυχαίνουμε καλύτερη ανακατασκευή της εικόνας. Όπως περιμένουμε επίσης μεγαλώνει αντιστοίχως και το των principal components παρουσιάζοντας το trade off της διαδικασίας. Τέλος θεωρούμε σημαντικό να αναφέρουμε πως η διαδικασία της PCA παράγει ένα lossy compression. Όσα principal components και να χρησιμοποιήσουμε δεν θα ανακατασκευάσουμε πλήρως το αρχικό μητρώο, παρουσιάζεται ένα diminishing return καθώς αυξάνουμε aggressively τον αριθμό των principal components.

Καθώς η χρονική διάρκεια της εκτέλεσης της PCA για την εικόνα του "Elvis" είναι τόσο μικρή κρίνουμε αδύνατη την ουσιαστική μέτρηση της παραλληλοποίησης που επιτύχαμε. Για αυτό τον σκοπό θα αναφέρουμε τις μετρήσεις μας για τις δύο άλλες εισόδους, χωρίς να αποθηκεύουμε τα παραγόμενα principal components και τα υπόλοιπα απαραίτητα στοιχεία που αναφέραμε με στόχο την ανακατασκευή της εικόνας.

Εφόσον χρησιμοποιούμε την Openblas βιβλιοθήκη για ορισμένες εκ των συναρτήσεων που χρησιμοποιούμε, οφείλουμε να ορίσουμε environmental variables για να δηλώσουμε τον αριθμό των threads που χρησιμοποιούμε. Υπάρχει η δυνατότητα ορισμού με την χρήση της openblas_set_num_threads(), ωστόσο καθώς χρησιμοποιούμε ταυτόχρονα και openmp το documentation και οι πηγές οι αναζητήσαμε διαφέρουν. Ο αριθμός των threads που χρησιμοποιεί η βιβλιοθήκη των openblas ορίζεται σε μία global variable η οποία διαβάζεται στην αρχή της εκτέλεσης του προγράμματος μας εφόσον δεν έχει οριστεί exported variable για τα threads του openmp, αν έχει γίνει ορθά compiled η βιβλιοθήκη με USE_OPENMP argument. Θεωρήσαμε πως ο τρόπος που ακολουθούμε είναι ο πιο ασφαλής για την εξασφάλιση ορθών μετρήσεων.

		Components - npc	**Time (Seconds) For Threads**			
			1	**2**	**4**	**6**
Cyclone 4096x4096	Overall Time:	**1**	82.1684	58.7663	47.2383	42.9213
		30	83.5421	58.6766	46.3105	42.0739
		50	82.6683	58.6018	45.6156	42.8198
		100	82.4998	57.3617	46.0178	42.3505
	DSYEV Time:	**1**	36.7753	35.8556	35.7620	35.2785
		30	37.9779	35.7488	34.7318	34.4173
		50	37.1523	35.4884	34.1846	35.1469
		100	36.8623	34.4970	34.5298	34.6789
Earth 9500x9500	Overall Time:	**1**	1018.16	725.663	586.721	538.002
		30	1015.91	723.181	590.229	542.349
		50	1021.72	717.328	590.967	536.177
		100	1019.89	710.155	592.623	537.041
	DSYEV Time:	**1**	449.017	444.73	439.328	442.016
		30	445.221	441.73	447.878	449.449
		50	450.724	438.73	446.218	441.272
		100	448.061	422.73	448.318	445.350
[Cyclone] Efficiency (P)		**Strong Scaling**		0.706	0.448	0.324
[Earth] Efficiency (P)				0.718	0.430	0.316

Σχολιάζοντας τα αποτελέσματα μας, αρχικά πρέπει να επισημάνουμε την αδυναμία παραλληλοποίησης της συνάρτησης LAPACKE_dsyev, όπως είχαμε αναφέρει και προηγουμένως. Ως εκτούτου σε κάθε κάλεσμα της διαδικασίας εξαρτάται από τον χρόνο εύρεσης των ιδιοτιμών και ιδιοδιανυσμάτων. Με στόχο την δημιουργία μίας αποδοτικής υλοποίησης της PCA πρέπει να οδηγηθούμε σε εκτιμήσεις των ιδιοτιμών και την προσεγγίση μόνο του αριθμού των ιδιαδιανυσμάτων που θέλουμε να χρησιμοποιήσουμε. Ο υπολογισμός όλων των ιδιοδιανυσμάτων του covariance matrix γίνεται σε κάθε εκτέλεση του κώδικα μας ανεξάρτητα από τον αριθμό των principal components που εντέλει χρησιμοποιούμε. Συνάγεται το συμπέρασμα πως εξαρτόμαστε πλήρως από τον χρόνο της dsyev ενώ δεν παρατηρούμε ιδιαίτερη βελτίωση της απόδοσης μειώνοντας τον αριθμό των principal components. Περίπου λοιπόν το 42.5% του προγράμματος μας είναι σειριακό από τα προαναφερθέντα, ενώ ακόμα δεν έχουμε παραλληλοποιήσει και την συνάρτηση για το Matrix Transposing καθώς δεν θεωρείται πως είναι και ιδιαίτερα εφικτό. Εξετάζοντας το Efficiency για Strong Scaling ενισχύονται και οι υποθέσεις που είχαμε προηγουμένως. Καθώς αυξάνουμε τα threads παρουσιάζεται σημαντική πτώση του efficiency καθώς περίπου ο μισός κώδικας ουσιαστικά βελτιώνεται λόγω του multithreading.

❖ Part B

Στην υλοποίηση του δεύτερου μέρους της εργαστηριακής άσκησης μεταβάλουμε αποκλειστικά το κάλεσμα της *computeGravitationalForces()* παρουσιάζοντας 3 αρχεία, τον σειριακό κώδικα, αποκλειστικά χρήση SIMD intrinsics και τέλος παραλληλοποίηση του προαναφερθέντος με την χρήση OPENMP.
Με στόχο τον σχολιασμό της υλοποίησης μας οφείλουμε να αναφέρουμε πως το πιο αξιοσημείωτο 'trick' μας δόθηκε. Υπολογίζοντας τις αντιβαρυτικές δυνάμεις, πρέπει να προβλέψουμε να μην υπολογίσουμε την δύναμη που ασκεί το ίδιο το σώμα στον εαυτό του, καθώς αυτό θα αποτελούσε και λάθος στην φυσική σημασία του προβλήματος. Δημιουργούμε μία μάσκα κάνοντας compare των αριθμών των επαναλήψεων με το τρέχον particle, στην περίπτωση που υπάρχει ταύτιση των particles από τα οποία υπολογίζουμε τις αντιβαρυτικές δυνάμεις με το particle για το οποίο τις υπολογίζουμε, πολλαπλασιάζουμε το αποτέλεσμα με την μάσκα, μηδενίζοντας το ουσιαστικά. Ενώ φαίνεται πως αυτό οδηγεί σε περισσότερους υπολογισμούς καθώς χρησιμοποιούμε SIMD κώδικα υπολογίζουμε ταυτόχρονα τις δυνάμεις των 3 άλλων particles, το οποίο εντέλει οδηγεί σε ένα "net positive" αποτέλεσμα.
Επιπροσθέτως σημαντική βελτίωση στην απόδοση οδήγησε η χρήση SIMD instructions για τον υπολογισμό της τετραγωνικής ρίζας για την ύψωση σε δυνάμεις και η προσπάθεια εκμετάλλευσης κάθε δυνατής μορφής FMA instructions σε συνδιασμό με SIMD. Η "αναδόμηση" περιοχών του κώδικα με τα παραπάνω παρουσιάζει και καλύτερα αποτελέσματα από το αυτοματοποιημένο SIMD directive που μπορεί να επιτευχθεί με την χρήση του OPENMP.
Τα μεγαλύτερα ερωτήματα δημιουργήθηκαν εξετάζοντας το pipelining και τον assembly κώδικα που δημιουργείται σε εντολές όπως " _mm256_fmadd_pd(xr,xr, _mm256_fmadd_pd(yr,yr, _mm256_mul_pd(zr,zr))); " στον Compiler explorer του

Godbolt καθώς αναζητήσαμε τις διαφοροποιήσεις των εννοιών του instruction latency και throughput σε συνδιασμό με operational chains. Ωστόσο δεν κατέχουμε πλήρη κατανόηση φυσικά ούτε των βελτιστοποιήσεων που παράγονται από τον GCC, ούτε (κατά ομολογία) των εννοιών εξαρχής. Τέλος θέλουμε πάλι να επισημάνουμε πως τα αποτελέσματα μπορεί να είναι skewed καθώς οι mainstream AMD επεξεργαστές δεν υποστηρίζουν "θεωρητικά" το FMA 4 instruction set.

Particles Number	Serial Code	SIMD Code	Threaded Code			
			1	2	4	6
3	8e-06	7e-06	1e-05	1.6e-05	1e-05	1.3e-05
32	1.7e-05	2e-06	3e-06	3.5e-05	6e-05	2e-06
2017	0.05829	0.006928	0.00675	0.00312	0.00195	0.00183
11111	1.76323	0.190121	0.18740	0.09335	0.05441	0.05318
Efficiency (P) - Strong Scaling			2017 :	1.110	0.8882	0.6309
			11111 :	1.018	0.8735	0.5958

❖ Συμπεράσματα και Παρατηρήσεις:

Όπως διακρίνουμε ο κώδικας περνάει επιτυχώς όλα τα "tests" βάση του particles number, γεγονός μου μας οδηγεί στην βεβαίωση πως τα παραγόμενα αποτελέσματα είναι ορθά. Τα αποτελέσματα και το speedup που παρατηρούμε κατά την εκτέλεση ειδικότερα του SIMD κώδικα φαντάζουν αδύνατα, 9 φορές γρηγορότερα σε σχέση με τον σειριακό για να είμαστε ακριβείς. Το συγκεκριμένο speedup είναι φυσικά αδύνατο να αιτιολογηθεί αποκλειστικά σε χρήση SIMD κώδικα και υποθέτουμε πως η χρήση των FMA παίζει ένα καθοριστικό παράγοντα στα λαμβανόμενα αποτελέσματα. Επιπρόσθετα είναι σημαντικό να σημειωθεί πως ο χρόνος εκτέλεσης είναι πολύ μικρός για να θεωρήσουμε τα αποτελέσματα αξιόπιστα. Σε αυτό επίσης θα αιτιολογήσουμε και το θεωρητικά αδύνατο efficiency scaling καθώς χρησιμοποιούμε περισσότερα (2) threads. Επίσης είναι πιθανό πως στην πράξη η χρήση των δύο threads πετυχαίνει καλύτερο cache utilization, απαιτώντας λιγότερες μεταφορές μεταξύ των κρυφών μνημών, μειώνεται έτσι πρακτικά το primary storage I/O latency. Καθώς μεταβαίνουμε σε χρήση περισσότερων threads ο κώδικας φαίνεται να μην δέχεται περαιτέρω speedup από 4 σε 6 threads. Πιθανολογούμε πως ο χρόνος εκτέλεσης είναι τόσο μικρός που το latency που παράγεται από την δημιουργία περισσότερων threads μειώνει τόσο το workload του κάθε thread που "ζυγοσταθμίζεται" ο χρόνος εκτέλεσης. Η υπόθεση ωστόσο αυτή δεν έχει βάση σε πειραματικές μετρήσεις και αιτολογείται αποκλειστικά στο γεγονός πως η φύση του κώδικα φαίνεται να "προσφέρεται" για παραλληλισμό και δεν εμποδίζεται από τίποτα, παραμόνο το oversubscription στα threads και στο workload που διαμοιράζεται μεταξύ τους.