ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
"ΘΕΩΡΗΤΙΚΗ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΘΕΩΡΙΑ ΣΥΣΤΗΜΑΤΩΝ ΕΛΕΓΧΟΥ"

Υπολογισμός ορίζουσας πολυμεταβλητού πολυωνυμικού πίνακα με χρήση παρεμβολής και μετασχηματισμού Hartley

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Πολίτης

Επιβλέπων: Νικόλαος Καραμπετάκης
Αν. Καθηγητής Α.Π.Θ.

Θεσσαλονίκη, Δεκέμβριος 2011

ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ
"ΘΕΩΡΗΤΙΚΗ ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΘΕΩΡΙΑ ΣΥΣΤΗΜΑΤΩΝ ΕΛΕΓΧΟΥ"

Υπολογισμός ορίζουσας πολυμεταβλητού πολυωνυμικού πίνακα με χρήση παρεμβολής και μετασχηματισμού Hartley

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος Πολίτης

Επιβλέπων: Νικόλαος Καραμπετάκης
Αν. Καθηγητής Α.Π.Θ.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22 Δεκεμβρίου 2011.

.....	.....	.....
Ν. Καραμπετάκης	Α. Βαρδουλάκης	Μ. Γουσίδου
Αν. Καθηγητής Α.Π.Θ.	Καθηγητής Α.Π.Θ.	Αν. Καθηγητής Α.Π.Θ.

Θεσσαλονίκη, Δεκέμβριος 2011

.....

Δημήτριος Πολίτης
Πτυχιούχος Εφαρμοσμένης Πληροφορικής Πανεπιστημίου Μακεδονίας

ΠΕΡΙΛΗΨΗ

Ο μετασχηματισμός Hartley (HT) είναι ένας πραγματικός μετασχηματισμός συνεχούς χρόνου, με παρόμοιες ιδιότητες και εφαρμογές με το μιγαδικό μετασχηματισμό Fourier. Η διακριτή μορφή του έχει μελετηθεί σε διάφορες εργασίες και έχει αποδειχθεί γρηγορότερη και πιο εύκολη στην εφαρμογή από τον αντίστοιχο διακριτό μετασχηματισμό Fourier (FFT). Καθώς ο πυρήνας του μετασχηματισμού Hartley δεν είναι διαχωρίσιμος, όπως αυτός του Fourier, είναι αναγκαίο ένα επιπλέον βήμα για την μετατροπή μετασχηματισμών Hartley μικρότερων διαστάσεων στον πολυδιάστατο μετασχηματισμό. Στην βιβλιογραφία υπάρχει πληθώρα αλγορίθμων για τον μετασχηματισμό Hartley, σε δύο ή τρεις διαστάσεις, ενώ ο πολυδιάστατος μετασχηματισμός Hartley δεν έχει μελετηθεί εκτενώς. Σε αυτήν την εργασία παρουσιάζουμε διάφορους αλγόριθμους για τον υπολογισμό του γρήγορού διακριτού μετασχηματισμού Hartley. Τέλος, παρουσιάζουμε την εφαρμογή του μετασχηματισμού Hartley στον υπολογισμό του 2-D μετασχηματισμού Fourier, για την εύρεση της ορίζουσας ενός πολυωνυμικού πίνακα μέσω πολυωνυμικής παρεμβολής. Δείχνουμε ότι η παραλληλοποίηση του κώδικα κατά τον υπολογισμό του μετασχηματισμού FFT, που προσφέρει η χρήση του μετασχηματισμού Hartley, μπορεί να προσφέρει σημαντικά υπολογιστικά οφέλη. Ας σημειωθεί ότι οι όροι «πολυδιάστατος» και «πολυμεταβλητός» χρησιμοποιούνται εναλλακτικά, για να ορίσουν έναν μετασχηματισμό πολλών μεταβλητών.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Διακριτός Μετασχηματισμός Hartley (DHT, FHT), Διακριτός Μετασχηματισμός Fourier (DFT, FFT), Πολυωνυμικός Πίνακας, Πολυωνυμική Παρεμβολή.

ABSTRACT

The Discrete Hartley Transform (HT) is a real variable, continuous time transform that shows similar properties and has nearly the same applications, as the Fourier Transform. The discrete form of Hartley transform has been studied in many papers and is proven out to be faster and easier to implement, than the corresponding Fourier transform (FFT). Whilst the core function of the Hartley transform is not separable, as the Fourier core, an added step is required in order to compute the multidimensional transform. There exists a plethora of algorithms in the literature, concerning the Hartley transform in two or three dimensions. On the contrary, the multidimensional Hartley transform has not been yet studied thoroughly. In this paper we present several algorithms for the computation of the 1-D fast Hartley transform (FHT) and introduce the use of Hartley transform in the computation of the 2-D FFT. We conclude by applying the aforementioned FFT via FHT algorithm, to compute the determinant of a 2-variable polynomial matrix via interpolation. We show that great performance gain can be achieved by the parallel code execution, which the use of Hartley transform offers, in multiple computers or cores. In this paper the terms "multivariable" and "multidimensional" are used alternatively, to denote a transform of more than one variables.

KEY WORDS

Discrete Hartley Transform (DHT, FHT), Discrete Fourier Transform (DFT, FFT), Polynomial Matrix, Polynomial Interpolation.

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία πραγματοποιήθηκε στα πλαίσια του προγράμματος μεταπτυχιακών σπουδών του Τμήματος Μαθηματικών, στην ειδίκευση Θεωρητικής Πληροφορικής και Θεωρίας Συστημάτων και Ελέγχου.

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον επιβλέπων της εργασίας μου, Αν. καθηγητή κ. Καραμπετάκη Νικόλαο για την υπόδειξη του θέματος, την επιστημονική καθοδήγηση και το διαρκές ενδιαφέρον καθόλη την διάρκεια της εκπόνησης της εργασίας.

Ευχαριστώ, τα μέλη της τριμελούς επιτροπής, καθηγητή κ. Βαρδουλάκη Αντώνιο και Αν. καθηγήτρια κα. Γουσίδου-Κουτίτα Μαρία για την προσεκτική μελέτη της εργασίας και τις ιδιαίτερα χρήσιμες παρατηρήσεις τους.

Τέλος, οφείλω να ευχαριστήσω τη σύζυγό μου, τους γονείς μου και τον αδερφό μου, για την αμέριστη συμπαράσταση και βοήθεια που μου προσέφεραν κατά τη διάρκεια της συγγραφής της εργασίας.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΕΦΑΛΑΙΟ 1

Ο ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΣ HARTLEY

1.1. Εισαγωγή

Ο μετασχηματισμός Hartley είναι ένας μετασχηματισμός ολοκληρώματος, ο οποίος αντιστοιχεί μια πραγματική συνάρτηση του χρόνου σε μια πραγματική συνάρτηση συχνοτήτων μέσω του πυρήνα $cas(\nu x) = cos(\nu x) + sin(\nu x)$. Αυτή η νέα συμμετρική μορφή του παραδοσιακού μετασχηματισμού Fourier, η οποία αποδίδεται στον Ralf Hartley, οδηγεί σε έναν συσχετισμό μεταξύ της συνάρτησης της αρχικής μεταβλητής και του μετασχηματισμού αυτής. Επιπλέον, ο μετασχηματισμός Hartley επιτρέπει τον διαχωρισμό μίας συνάρτησης σε δύο ανεξάρτητα σύνολα ημιτονοειδών συνιστωσών.

Κατά την ανάλυση και σύνθεση συστημάτων και σημάτων στο πεδίο των συχνοτήτων, ο μετασχηματισμός Hartley μπορεί να χρησιμοποιηθεί ως ένας ταχύς αλγόριθμος, εναλλακτικός του Fourier, ενώ μπορεί να μειώσει σημαντικά το υπολογιστικό βάρος, συγκρινόμενος με τον κλασικό μετασχηματισμό Fourier. Στη συνέχεια θα θεωρήσουμε ότι η προς μετασχηματισμό συνάρτηση είναι πραγματικής μεταβλητής. Στις περισσότερες εφαρμογές πρακτικού ενδιαφέροντος όντως αυτή η παραδοχή ισχύει.

1.2. Λίγα ιστορικά στοιχεία

Ο Ralph V. L. Hartley γεννήθηκε στο Spruce Mountain της Νεβάδα το 1888. Αποφοίτησε από το Πανεπιστήμιο της Γιούτα το 1809 και στην συνέχεια σπούδασε στην Οξφόρδη για τρία χρόνια. Με την ολοκλήρωση των σπουδών του ο Hartley επέστρεψε από την Αγγλία και ξεκίνησε την επαγγελματική του καριέρα στην Western Electric Company. Το 1925 ο Hartley και άλλοι συνεργάτες του συνέβαλαν ως ιδρυτικά μέλη στη δημιουργία των εργαστηρίων Bell (Bell Telephone Laboratories).

Το 1927 διατύπωσε τον νόμο που φέρει το όνομά του οποίος λέει ότι: Το συνολικό ποσό της πληροφορίας το οποίο μπορεί να μεταδοθεί μέσα από ένα

σύστημα είναι ανάλογο με το γινόμενο του εύρους των συχνοτήτων στο οποίο μεταδίδει, επί το χρόνο που διαρκεί η μετάδοση.

Ο Hartley ήταν επινοητής 72 ευρεσιτεχνιών, οι οποίες απεικονίζουν τις ερευνητικές του δραστηριότητες. Παραιτήθηκε το 1950 από τα εργαστήρια Bell και πέθανε σε ηλικία 81 ετών το Μάιο του 1970.

Ίσως η πιο σημαντική του συμβολή στη σύγχρονη επιστήμη είναι η ανάπτυξη του μετασχηματισμού Hartley, ο οποίος δημοσιεύτηκε για πρώτη φορά στο βιβλίο του **[19].**

1.3 Βασικά στοιχεία του μετασχηματισμού Hartley

Ο μετασχηματισμός Hartley μιας συνάρτησης $f(x)$ μπορεί να εκφραστεί είτε

$$H(\nu) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x)\, cas(\nu x) dx \tag{1.1}$$

είτε

$$H(f) = \int_{-\infty}^{\infty} f(x)\, cas(2\pi f x) dx \tag{1.2}$$

όπου η γωνιακή ταχύτητα ν σχετίζεται με την συχνότητα f με τον τύπο $\nu = 2\pi f$.
Για την f ισχύει:

$$H(f) = \sqrt{2\pi} H(2\pi f) = \sqrt{2\pi} H(\nu) \tag{1.3}$$

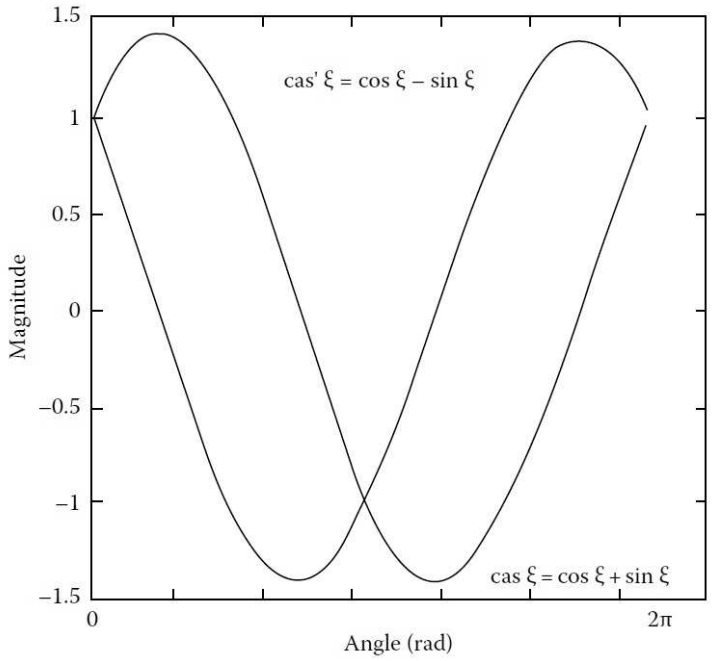
Ενώ για την συνάρτηση πυρήνα του μετασχηματισμού ισχύει ότι:

$$cas(\nu x) = cos(\nu x) + sin(\nu x) \tag{1.4}$$

$$cas(\nu x) = \sqrt{2} sin\left(\nu x + \frac{\pi}{4}\right) \tag{1.5}$$

$$cas(\nu x) = \sqrt{2} cos\left(\nu x - \frac{\pi}{4}\right) \tag{1.6}$$

Ο τύπος (1.5) μπορεί να χρησιμοποιηθεί στο MatLab για τον υπολογισμό της (1.4), καθώς υπερέχει αυτής σε πλήθος αριθμητικών πράξεων, ενώ το Matlab αποδείχθηκε μετά από σχετικά τεστ (κώδικας sine_tests.m), ότι υπολογίζει γρηγορότερα το ημίτονο παρά το συνημίτονο. Το παρακάτω σχεδιάγραμμα αναπαριστά τη συνάρτηση πυρήνα του μετασχηματισμού στο πεδίο ορισμού $[0, 2\pi]$.



Σχήμα 1: *Γραφική παράσταση του cas* ***[35].***

Άλλες χρήσιμες τις τριγωνομετρικές ιδιότητες του *cas* φαίνονται στον παρακάτω πίνακα:

Πίνακας 1: *Επιλεγμένες ιδιότητες της συνάρτησης – πυρήνα του μετασχηματισμού Hartley*

Ορισμός	$cas\xi = cos\xi + sin\xi$
Μιγαδική σχέση	$cas\xi = \frac{1}{2}\left[(1+j)exp(-j\xi)+(1-j)exp(j\xi)\right]$
Παράγωγος συνάρτηση	$cas'\xi = cas(-\xi) = cos\xi - sin\xi$
Σχέση με το ημίτονο	$cos\xi = \frac{1}{2}\left[cas\xi + cas(-\xi)\right]$
Σχέση με το συνημίτονο	$sin\xi = \frac{1}{2}\left[cas\xi - cas(-\xi)\right]$
Αμοιβαία σχέση	$cas\xi = \frac{csc\xi + sec\xi}{sec\xi + csc\xi}$
Διάσπαση σε άθροισμα	$cas\xi = cot\,\xi\, sin\,\xi + tan\,\xi\, cos\,\xi$
Σχέση γινομένων	$cas\,\tau\, cas\,\upsilon = cos(\tau - \upsilon) + sin(\tau + \upsilon)$
Σχέση πηλίκου	$cas\xi = \frac{cot\,\xi\, sec\,\xi + tan\,\xi\, csc\,\xi}{csc\,\xi\, sec\,\xi}$
Σχέση διπλάσιας γωνίας	$cas2\xi = cas^2\xi - cas^2(-\xi)$

Πίνακας 1: *Επιλεγμένες ιδιότητες της συνάρτησης – πυρήνα του μετασχηματισμού Hartley*

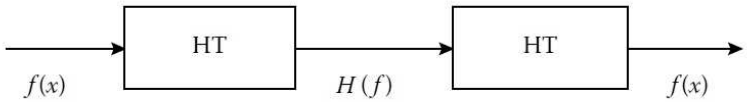
Αόριστο ολοκλήρωμα	$\int cas(\tau)d\tau = -cas(-\tau) = -cas'\tau$
Παραγώγιση	$\frac{d}{dt} cas\tau = cas(-\tau) = cas'\tau$
Σχέση αθροίσματος γωνιών	$cas(\tau + \upsilon) = cos\ \tau\ cas\ \upsilon + sin\ \tau\ cas'\upsilon$
Σχέση διαφοράς γωνιών	$cas(\tau - \upsilon) = cos\ \tau\ cas'\upsilon + sin\ \tau\ cas\ \upsilon$
Άθροισμα	$cas\ \tau + cas\ \upsilon = 2cas\frac{1}{2}(\tau + \upsilon)\ cos\frac{1}{2}(\tau - \upsilon)$
Διαφορά	$cas\ \tau - cas\ \upsilon = 2cas'\frac{1}{2}(\tau + \upsilon)\ sin\frac{1}{2}(\tau - \upsilon)$

Ο αντίστροφος μετασχηματισμός Hartley ορίζεται ως

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H(\nu) cas(\nu x) d\nu \quad \text{ή} \tag{1.7}$$

$$f(x) = \int_{-\infty}^{\infty} H(f) cas(2\pi f x) df \tag{1.8}$$

Η γωνιακή ταχύτητα ν αντιστοιχεί στην συχνότητα ω του μετασχηματισμού Fourier. Ο πολλαπλασιαστής $1/\sqrt{2\pi}$ χρησιμοποιείται για να διατηρηθεί η ιδιότητα της αυτο-αντιστροφής όπως φαίνεται στο σχήμα 2:



Σχήμα 2: *Ιδιότητα της αυτο-αντιστροφής*

Η ύπαρξη του μετασχηματισμού Hartley μιας συνάρτησης f ορίζεται από τις ίδιες συνθήκες που ορίζουν και την ύπαρξη του μετασχηματισμού Fourier αυτής, δηλαδή αν και μόνο αν το ολοκλήρωμα της σχέσης (1.1) συγκλίνει. Αυτό είναι δυνατό όταν ισχύουν οι παρακάτω συνθήκες:

1. $\int_{-\infty}^{\infty} |f(x)| dx < \infty$ δηλαδή η $f(x)$ είναι απολύτως ολοκληρώσιμη

2. Η $f(x)$ έχει πεπερασμένα σημεία ασυνέχειας σε ένα πεπερασμένο διάστημα.

3. Η $f(x)$ έχει ένα πεπερασμένο πλήθος τοπικών ελαχίστων και τοπικών μεγίστων σε ένα πεπερασμένο διάστημα.

Οι παραπάνω ικανές συνθήκες περιλαμβάνουν τα περισσότερα, πεπερασμένης ενέργειας, σήματα που συναντούμε συχνότερα. Πολλά όμως σήματα ιδιαίτερης σημασίας, όπως τα περιοδικά σήματα ή η βηματική συνάρτηση, δεν είναι απολύτως ολοκληρώσιμα. Αν επιτρέψουμε στον μετασχηματισμό Hartley να συμπεριλάβει και την συνάρτηση δ του Dirac, τότε ακόμα και αυτές οι συναρτήσεις μπορούν να μετασχηματίσουν, χρησιμοποιώντας παρόμοιες μεθόδους με αυτές που χρησιμοποιούμε στα σήματα πεπερασμένης ενέργειας **[35]**.

1.4 Σχέση μεταξύ μετασχηματισμού Fourier και Hartley

Όπως αναφέρθηκε παραπάνω, η ύπαρξη του μετασχηματισμού Hartley είναι ισοδύναμη με την ύπαρξη του μετασχηματισμού Fourier μιας συνάρτησης $f(x)$, ο οποίος δίνεται από τον παρακάτω τύπο.

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-j\omega x} dx \tag{1.9}$$

Η παραπάνω σχέση μπορεί ισοδύναμα να εκφραστεί από τις παρακάτω εξισώσεις **[35]**:

$$F(\omega) = \left[\frac{H(\nu) + H(-\nu)}{2} - j \frac{H(\nu) - H(-\nu)}{2} \right]_{\nu=\omega} \tag{1.10}$$

ή εναλλακτικά ως εξής

$$F(f) = \frac{1}{2} e^{-j\pi/4} H(f) + \frac{1}{2} e^{j\pi/4} H(-f) \tag{1.11}$$

Παράλληλα, οι συναρτήσεις f και $H(f)$ μπορούν να αναλυθούν στα άρτια και στα περιττά μέρη τους ως εξής:

$$\begin{aligned} f^{even} = \frac{1}{2}[f(x) + f(-x)] &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H^{even}(\nu) \cos(\nu x) dx \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H(\nu) \cos(\nu x) dx \end{aligned} \tag{1.12}$$

και

$$f^{odd} = \frac{1}{2}[f(x) - f(-x)] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H^{odd}(\nu) sin(\nu x) dx$$
$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H(\nu) sin(\nu x) dx \tag{1.13}$$

όπου $H^{even}(f)$ και H^{odd} είναι το άρτιο και το περιττό μέρος του μετασχηματισμού Hartley αντίστοιχα:

$$H^{even}(f) = \frac{H(f) + H(-f)}{2} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) cos(\nu x) dx \tag{1.14}$$

$$H^{odd}(f) = \frac{H(f) - H(-f)}{2} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) sin(\nu x) dx \tag{1.15}$$

Συνοψίζοντας, ο μετασχηματισμός Fourier είναι το άρτιο μέρος του μετασχηματισμού Hartley αθροιζόμενο με j φορές το περιττό.

Αντίστροφα, ο μετασχηματισμός Hartley μπορεί να εκφραστεί, χρησιμοποιώντας τον μετασχηματισμό Fourier, ως εξής:

$$H(\nu) = [\Re\{F(\omega)\} - \Im\{F(\omega)\}]_{\omega=\nu} \tag{1.16}$$

όπου

$$\Re\{F(\omega)\} = \Re(\omega) = H^{even}(\nu) = H^{even}(-\nu) \tag{1.17}$$

$$\Im\{F(\omega)\} = \Im(\omega) = -H^{odd}(\nu) = H^{odd}(-\nu) \tag{1.18}$$

Κατά παρόμοιο τρόπο, ο μετασχηματισμός Hartley είναι το πραγματικό μέρος αθροιζόμενο με το αντίθετο του φανταστικού μέρους του μετασχηματισμού Fourier.

$$H(f) = \frac{1 + j1}{2} F(f) + \frac{1 - j1}{2} F(-f) \qquad \text{ή} \tag{1.19}$$

$$H(f) = \frac{1}{2} e^{j\pi/4} F(f) + \frac{1}{2} e^{-j\pi/4} F^*(f) \tag{1.20}$$

Είναι εύκολο να καταλάβουμε, ότι ο μετασχηματισμός Hartley συνδέεται στενά με αυτόν του Fourier, εφόσον υπολογίζοντας τον έναν, μπορούμε με λίγες πράξεις να υπολογίσουμε και τον άλλο. Αυτή η ιδιότητα του μετασχηματισμού Hartley είναι ο λόγος για τον οποίο ο μετασχηματισμός αυτός χρησιμοποιείται ως εναλλακτικός και έχει παρόμοιες εφαρμογές, με τον μετασχηματισμό Fourier.

1.5 Στοιχειώδεις ιδιότητες του μετασχηματισμού Hartley

Παρακάτω θα αναπτύξουμε κάποια από τα θεωρήματα του μετασχηματισμού Hartley, τα οποία είναι χρήσιμα για την ανάλυση συστημάτων και σημάτων.

Ιδιότητα 1 – Γραμμικότητα

Εάν $f_1(x)$ και $f_2(x)$ έχουν μετασχηματισμό Hartley, αντίστοιχα $H_1(f)$ και $H_2(f)$, τότε το άθροισμα $af_1(x)+\beta f_2(x)$ έχει μετασχηματισμό Hartley $aH_1(f)+\beta H_2(f)$.

Απόδειξη

$$\int_{-\infty}^{\infty}[af_1(x)+\beta f_2(x)]cas(2\pi fx)dx$$

$$=a\int_{-\infty}^{\infty}f_1(x)cas(2\pi fx)dx+\beta\int_{-\infty}^{\infty}f_2(x)cas(2\pi fx)dx$$

$$=aH_1(f)+\beta H_1(f) \tag{1.21}$$

▲

Ιδιότητα 2 – Φάσμα ισχύος και φάση

Το φάσμα ισχύος ενός σήματος $f(x)$ μπορεί να εκφραστεί στο πεδίο συχνοτήτων ως $P(f)=|F(f)|^2=\Re\{F(f)\}^2+\Im\{F(f)\}^2$

Απόδειξη

Το φάσμα ισχύος μπορεί να υπολογιστεί απευθείας από τον μετασχηματισμό Hartley, χρησιμοποιώντας τις σχέσεις (1.14) και (1.15) όπως παρακάτω:

$$P(f)=|F(f)|^2=\Re\{F(f)\}^2+\Im\{F(f)\}^2$$

$$=[H^{even}(f)]^2+[-H^{odd}(f)]^2$$

$$=\frac{1}{4}[H(f)+H(-f)]^2+\frac{1}{4}[H(f)-H(-f)]^2$$

$$P(f)=\frac{[H(f)]^2+[H(-f)]^2}{2} \tag{1.22}$$

▲

Η φάση που συνδέεται με τον μετασχηματισμό Fourier ενός σήματος $f(x)$ εκφράζεται όπως παρακάτω:

$$\Phi(f) = tan^{-1}\left(\frac{\Im\{F(f)\}}{\Re\{F(f)\}}\right) = tan^{-1}\left(\frac{-H^{0}(f)}{H^{e}(f)}\right)$$
$$\Phi(f) = tan^{-1}\left(\frac{H(-f) - H(f)}{H(f) + H(-f)}\right) \tag{1.23}$$

<u>*Ιδιότητα 3 – Μετατόπιση στον οριζόντιο άξονα*</u>

Ο μετασχηματισμός Hartley της συνάρτησης $f(kx)$, όπου k θετικός αριθμός, ορίζεται όπως παρακάτω:

$$\int_{-\infty}^{\infty} f(kx)cas(2\pi fx)dx = \int_{-\infty}^{\infty} f(x')cas(\frac{2\pi fx'}{k})\frac{dx'}{k}$$
$$= \frac{1}{k}H\left(\frac{f}{k}\right) \tag{1.24}$$

Αν ο k είναι αρνητικός τότε τα όρια ολοκλήρωσης αντιστρέφονται και ο τελευταίος όρος γίνεται $(1/-k)H(f/k)$ ▲

Αρά λοιπόν αν $f(x)$ έχει μετασχηματισμό Hartley $H(f)$, τότε $f(k/x)$ έχει μετασχηματισμό Fourier $(1/|k|)H(f/k)$.

<u>*Ιδιότητα 4 – Αντιστροφή πρόσημου*</u>

Αν η $f(x)$ έχει μετασχηματισμό Hartley $H(f)$, τότε ο μετασχηματισμός της $f(-x)$ είναι $H(-f)$

<u>*Απόδειξη*</u>

Αυτό αποδεικνύεται πολύ εύκολα αν βάλουμε στην προηγούμενη ιδιότητα όπου k το -1. ▲

<u>*Ιδιότητα 5 – Χρονική μετατόπιση*</u>

Αν η $f(x)$ μετατοπιστεί στο χρόνο κατά μια σταθερά T, τότε αντικαθιστώντας όπου $x' = x - T$ ο μετασχηματισμός Hartley γίνεται:

$$H(f) = \int_{-\infty}^{\infty} f(x')cas\left[2\pi f(x'+T)\right]dx' \tag{1.25}$$

Χρησιμοποιώντας την ιδιότητα διάσπασης σε άθροισμα από τον Πίνακα 1 καταλήγουμε τελικά στη σχέση:

$$cas\left[2\pi f(x'+T)\right] = cos(2\pi fx')cos(2\pi fT) + cos(2\pi fx')sin(2\pi fT)$$
$$+sin(2\pi fx')cos(2\pi fT) - sin(2\pi fx')sin(2\pi fT)$$

Ολοκληρώνοντας ξεχωριστά τους τέσσερεις όρους που προκύπτουν και ομαδοποιώντας τον πρώτο με τον τρίτο και τον δεύτερο με τον τέταρτο καταλήγουμε:

$$H(f) = cos(2\pi fT)H(f) + sin(2\pi fT)H(-f) \tag{1.26}$$

▲

Ιδιότητα 6 – Διαμόρφωση

Αν η $f(x)$ διαμορφώνεται από ένα ημιτονοειδές της μορφής $cos(2\pi f_0 x)$, τότε ο μετασχηματισμός Hartley δίνει:

$$H(f) = \int_{-\infty}^{\infty} f(x)cos\left(2\pi f_0 x\right)cas\left(2\pi fx\right)dx$$

$$H(f) = \int_{-\infty}^{\infty} f(x)cos\left(2\pi f_0 x\right)cos\left(2\pi fx\right)dx$$

$$+\int_{-\infty}^{\infty} f(x)cos\left(2\pi f_0 x\right)sin\left(2\pi fx\right)dx \tag{1.27}$$

και τελικά

$$H(f) = \frac{1}{2}H(f - f_0) + \frac{1}{2}H(f + f_0) \tag{1.28}$$

▲

Ιδιότητα 7 – Συνέλιξη

Εάν η $f_1(x)$ έχει μετασχηματισμό Hartley $H_1(f)$ και η $f_2(x)$ έχει $H_2(f)$, τότε η συνέλιξη $f_1(x) * f_2(x)$ είναι ίση με

$$\frac{1}{2}[H_1(f)H_2(f) + H_1(-f)H_2(f) + H_1(f)H_2(-f)$$
$$-H_1(-f)H_2(-f)] \tag{1.29}$$

Απόδειξη

Για να καταλήξουμε σε αυτό το αποτέλεσμα απευθείας, αντικαθιστούμε το ολοκλήρωμα της συνέλιξης στη σχέση

$$f_1(x) * f_2(x) = \int_{-\infty}^{\infty} f_1(\lambda)f_2(x - \lambda)d\lambda \tag{1.30}$$

και χρησιμοποιούμε την ιδιότητα 5. Το αποτέλεσμα έχει ως ακολούθως:

$$H(f) = \int_{-\infty}^{\infty} [f_1(x) * f_2(x)] cas(2\pi fx) dx$$

$$= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} f_1(\lambda) f_2(\lambda) d\lambda \right) cas(2\pi fx) dx = \int_{-\infty}^{\infty} f_1(\lambda) \left(\int_{-\infty}^{\infty} f_2(x - \lambda) cas(2\pi fx) dx \right) d\lambda =$$

$$= \int_{-\infty}^{\infty} f_1(t) [cos(2\pi f\lambda) H_2(f) + sin(2\pi f\lambda) H_2(-f)] d\lambda \qquad (1.31)$$

Παραγοντοποιώντας τον όρο H_2 στα δεξιά και χρησιμοποιώντας τις εξισώσεις (1.12) και (1.15) καταλήγουμε στο επιθυμητό. ▲

Να επισημάνουμε ότι η εξίσωση (1.29) απλοποιείται όταν ισχύουν οι παρακάτω συμμετρίες:

- Αν $f_1(x)$ ή $f_2(x)$ είναι άρτιες ή η $f_1(x)$ περιττή και η $f_2(x)$ άρτια και το αντίστροφο τότε η συνέλιξη $f_1(x) * f_2(x)$ είναι $H_1(f)H_2(f)$
- Αν η $f_1(x)$ είναι περιττή, τότε $f_1(x) * f_2(x)$ είναι ίσο με $H_1(f)H_2(-f)$. Το ίδιο ισχύει αντίστοιχα αν η $f_2(x)$ είναι περιττή, τότε $f_1(x) * f_2(x)$ είναι ίσο με $H_1(-f)H_2(f)$
- Αν και οι δύο συναρτήσεις είναι περιττές, τότε $f_1(x) * f_2(x)$ είναι ίσο με $H_1(f)H_2(f)$.

Ο παραπάνω τρόπος υπολογισμού της συνέλιξης με μετασχηματισμό Hartley, για συναρτήσεις πραγματικής μεταβλητής, είναι προτιμότερος από τον υπολογισμό με τη χρήση του μετασχηματισμού Fourier, που δίνεται από τη σχέση:

$$f_1(x) * f_2(x) = F^{-1}[F(f_1(x) \cdot F(f_2(x)]$$

Αυτό συμβαίνει γιατι έχουμε λιγότερες αριθμητικές πράξεις (τρεις μιγαδικοί μετασχηματισμοί Fourier έναντι τεσσάρων πραγματικών Hartley), μικρότερη πολυπλοκότητα λόγω χρησιμοποίησης πραγματικών τιμών, αντί μιγαδικών και μικρότερες απαιτήσεις σε μνήμη (Οι μιγαδικοί χρειάζονται διπλασια μνήμη για να αποθηκευτούν στον υπολογιστή). Οι P.Duhamel και M.Vetterli στο **[16]** απέδειξαν οτι η χρήση του μετασχηματισμού Hartley για τον υπολογισμό της συνέλιξης είναι αποδοτικότερη, από τους αλγορίθμους που χρησιμοποιούν FFT. Ο αποδοτικός υπολογισμός της συνέλιξης με τη χρήση του μετασχηματισμού

Hartley, μας δίνει και έναν τρόπο αποδοτικού υπολογισμού του αντίστροφου πολυμεταβλητού πολυωνυμικού πίνακα μέσω διακριτής συνέλιξης.

Ιδιότητα 8 – Αυτοσυσχέτιση

Αν $f_1(x)$ έχει μετασχηματισμό Hartley $H_1(f)$, τότε η αυτοσυσχέτιση ορίζεται από τον παρακάτω τύπο

$$f_1(x) \odot f_1(x) = f_1(x) * f_1(-x) = \int_{-\infty}^{\infty} f_1(\lambda) f_1(x+\lambda) d\lambda \tag{1.32}$$

και έχει μετασχηματισμό Harltey

$$\frac{1}{2}[H_1(f)^2 + H_1(-f)^2] = [H^{even}(f)^2] + [H^{odd}(f)^2] \tag{1.33}$$

Απόδειξη

Συγκρίνοντας τις εξισώσεις (1.30) έως (1.32), καταλήγουμε στο συμπέρασμα ότι η συνέλιξη και η αυτοσυσχέτιση έχουν συγκεκριμένες ομοιότητες. Αντικαθιστώντας το ολοκλήρωμα της συσχέτισης από την παραπάνω εξίσωση, στον τύπο του μετασχηματισμού Hartley και χρησιμοποιώντας την ιδιότητα 5 το αποτέλεσμα έχει ως ακολούθως:

$$H(f) = \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} f_1(\lambda) f_1(x+\lambda) d\lambda \right) cas(2\pi f x) dx$$

$$= \int_{-\infty}^{\infty} f_1(x) \left(\int_{-\infty}^{\infty} f_1(x+\lambda) cas(2\pi f x) dx \right) d\lambda \tag{1.34}$$

Χρησιμοποιώντας τις συναρτήσεις της μετατόπισης για $T = -\lambda$ έχουμε ,

$$= \int_{-\infty}^{\infty} f_1(x)[cos(2\pi f\lambda) H_1(f) - sin(2\pi f\lambda) H_1(-f)] d\lambda$$

Παραγοντοποιώντας το H_1 στα δεξιά και χρησιμοποιώντας τις εξισώσεις (1.12) – (1.15) καταλήγουμε στο επιθυμητό. ▲

Ιδιότητα 9 – Γινόμενο

Εάν η $f_1(x)$ πολλαπλασιαστεί με μία δεύτερη συνάρτηση $f_2(x)$, τότε το γινόμενο $f_1(x)f_2(x)$ έχει μετασχηματισμό Hartley

$$\frac{1}{2}[H_1(f)*H_2(f)+H_1(-f)*H_2(f)+H_1(f)*H_2(-f)$$

$$-H_1(-f)*H_2(-f)]$$

$$=H_1^{even}(f)*H_2^{even}(f)-H_1^{odd}(f)*H_2^{odd}(f)+H_1^{even}(f)*H_2^{odd}(f)$$

$$+H_1^{odd}(f)*H_2^{even}(f)$$

▲

<u>*Ιδιότητα 10 – Νιοστή παράγωγος της συνάρτησης*</u>

Η νιοστή παράγωγος της συνάρτησης $f(x)$ υπολογίζεται ως

$$cas'\frac{n\pi}{2}(2\pi f)^n H[(-1)^n f] \tag{1.35}$$

<u>*Απόδειξη*</u>

Αυτή η ιδιότητα αποδεικνύεται με αναδρομική εφαρμογή της εξίσωσης (1.16) στο μετασχηματισμό Fourier της συνάρτησης $df(x)/dx$ και των παραγώγων της μεγαλύτερης τάξης. ▲

Μια περίληψη των παραπάνω ιδιοτήτων περιλαμβάνεται στον παρακάτω πίνακα.

Πίνακας 2: *Στοιχειώδεις ιδιότητες του μετασχηματισμού Hartley*

Ιδιότητα	$f(x)$	$H(f)$
Γραμμικότητα	$f_1(x)+f_2(x)$	$H_1(x)+H_2(x)$
Φάσμα ισχύος	$P(f)=\frac{[H(f)]^2+[H(-f)]^2}{2}$	$\Phi(f)=tan^{-1}$
Μετατόπιση στον οριζόντιο άξονα	$f(kx)$	$(1/\lvert k\rvert)H(f/k)$
Αντιστροφή	$f(-x)$	$H(-f)$
Χρονική μετατόπιση	$f(x-T)$	$H(f)=cos(2\pi fT)H(f)+sin(2\pi fT)H(-f)$
Διαμόρφωση	$f(x)\,cos(2\pi f_0 t)$	$H(f)=\frac{1}{2}H(f-f_0)+\frac{1}{2}H(f+f_0)$
Συνέλιξη	$f_1(x)*f_2(x)$	$\frac{1}{2}[H_1(f)H_2(f)+H_1(-f)H_2(f)+H_1(f)H_2(-f)-H_1(-f)H_2(-f)]$
Αυτοσυσχέτιση	$f_1(x)\odot f_1(x)$	$\frac{1}{2}[H_1(f)^2+H_1(-f)^2]$

Πίνακας 2: *Στοιχειώδεις ιδιότητες του μετασχηματισμού Hartley*

Γινόμενο	$f_1(x)f_2(x)$	$\frac{1}{2}[H_1(f) * H_2(f) + H_1(-f) * H_2(f) + H_1(f) * H_2(-f) - H_1(-f) * H_2(-f)]$
Νιοστή παράγωγος	$f^n(x)$	$f^n(x) = cas' \frac{n\pi}{2} (2\pi f)^n H[(-1)^n f]$

1.6 Ο μετασχηματισμός Hartley σε πολλαπλές διαστάσεις

Για μια συνάρτηση $f(x,y)$ ορίζεται ο δισδιάστατος μετασχηματισμός Hartley

$$H(\upsilon,\nu) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, cas\,[2\pi(\upsilon x + \nu y)]dxdy \tag{1.36}$$

με αντίστοιχο αντίστροφο

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} H(\upsilon,\nu)\, cas\,[2\pi(\upsilon x + \nu y)]d\upsilon d\nu \tag{1.37}$$

Αντίστοιχα ορίζονται και οι μετασχηματισμοί Hartley περισσότερων διαστάσεων για πολυμεταβλητές συναρτήσεις f.

1.7 Ο διακριτός μετασχηματισμός Hartley

Παρόλο που ο μετασχηματισμός Hartley συνεχούς χρόνου είναι απαραίτητος για την επίλυση θεωρητικών προβλημάτων, ο διακριτός μετασχηματισμός είναι αυτός που χρησιμοποιείται, τις περισσότερες φορές, για την επίλυση πραγματικών προβλημάτων, που αφορούν τη θεωρία συστημάτων και ελέγχου. Η πρακτική εφαρμογή του Hartley transform (στο εξής HT) βασίζεται στην διακριτή μορφή του, η οποία μετατρέπει ένα διάνυσμα πραγματικών τιμών $h(t)$, με $t \in [0, n-1]$ σε ένα άλλο διάνυσμα $H(f)$ με πραγματικές τιμές με $f \in [0, n-1]$. Ας θεωρήσουμε τον διακριτό μετασχηματισμό Fourier μιας περιοδικής συνάρτησης με περίοδο NT sec.

$$F(k\Omega_\omega) = \sum_{n=0}^{N-1} f(nT) e^{-jk\Omega_\omega nT} \tag{1.38}$$

και τον αντίστροφό του

$$f(nT) = \frac{1}{N}\sum_{k=0}^{N-1} F(k\Omega_\omega) e^{jk\Omega_\omega nT} \tag{1.39}$$

όπου T η μέγιστη ακρίβεια του χρόνου δειγματοληψίας της συνάρτησης f,

N ο αριθμός των σημείων στην ακολουθία δεδομένων

Ω_ω η μέγιστη ακρίβεια απεικόνισης της συχνότητας σε rad/sec.

$$\Omega_\omega = \frac{2\pi}{NT} \tag{1.40}$$

Κατά τον ίδιο τρόπο, ο διακριτός μετασχηματισμός Hartley (στο εξής DHT) ορίζεται όπως παρακάτω

$$H(k\Omega_\nu) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} h(nT)\, cas\,(k\Omega_\nu nT) \tag{1.41}$$

και ο αντίστροφός του ως

$$h(nT) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} H(k\Omega_\nu)\, cas\,(k\Omega_\nu nT) \tag{1.42}$$

όπου $\Omega_\nu = \Omega_\omega$ σε rad/sec.

Η επιλογή του συντελεστή $1/\sqrt{N}$ έγινε ώστε να ικανοποιείται η ιδιότητα της αυτοαντιστροφής (ο αντίστροφος διακριτός μετασχηματισμός είναι ίδιος με τον εαυτό του). Ανάλογα με την περίπτωση του συνεχούς χρόνου, ο DHT συνδέεται με τον DFT (Discrete Fourier Transform) με τις παρακάτω σχέσεις:

$$H(k\Omega_\nu) = \Re\{F(k\Omega_\omega)\} - \Im\{F(k\Omega_\omega)\} \tag{1.43}$$

$$H(f) = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} h(t)\left(cos\frac{2\pi ft}{n} + sin\frac{2\pi ft}{n}\right) \tag{1.44}$$

$$F(k\Omega_\omega) = \frac{1}{2}\sqrt{N}\left(H(k\Omega_\omega) + \overline{H(k\Omega_\omega)} + \sigma i\left(H(k\Omega_\omega) - \overline{H(k\Omega_\omega)}\right)\right) \tag{1.45}$$

όπου

$$\overline{H(k\Omega_\omega)} = H(N - k\Omega_\omega) \tag{1.46}$$

και το σ υποδηλώνει το πρόσημο του μετασχηματισμού Fourier. Στην ουσία είναι το πρόσημο του εκθέτη στη συνάρτηση πυρήνα και είναι -1 για τον ευθύ μετασχηματισμό και 1 για τον αντίστροφο DFT. Χρησιμοποιώντας την σχέση (1.45) μπορούμε να υπολογίσουμε τον DFT και τον IDFT μέσω του DHT. Το ουσιαστικό κέρδος που προκύπτει από την εφαρμογή του DHT για τον υπολογισμό του DFT, είναι ότι ο υπολογισμός διασπάται σε δυο ανεξάρτητους μετασχηματισμούς Hartley, οι οποίοι μπορούν να εκτελεστούν παράλληλα προς κέρδος χρόνου. Κατάλληλοι αλγόριθμοι που μπορούν να χρησιμοποιηθούν για τον

υπολογισμό του FFT, μέσω της χρήσης του DHT, κατά την οποία υπολογίζονται παράλληλα δύο DFT έχουν παρουσιαστεί στα **[6]**, **[12].**

1.8 Τα πλεονεκτήματα του DHT έναντι του DFT

Έχει επισημανθεί παραπάνω η δυνατότητα της χρήσης του διακριτού μετασχηματισμού Hartley ως εναλλακτικού του Fourier, σε σήματα πραγματικών τιμών. Συγκεκριμένα, ο διακριτός μετασχηματισμός μπορεί να χρησιμοποιείται όπου ο υπολογιστικός χρόνος είναι αναγκαίο να μειώνεται στο ελάχιστο, για παράδειγμα στην επεξεργασία σημάτων πραγματικού χρόνου. Ενδεικτικά αναφέρονται μερικές εφαρμογές του DHT **[35]**:

- Γρήγορος υπολογισμός συνέλιξης, αυτοσυσχέτισης, παρεμβολής, κρουστικής απόκρισης και σχεδίασης πολυμεταβλητών φίλτρων. Ειδικά για τον υπολογισμό της συνέλιξης, ο DHT είναι προτιμότερος από τον DFT, όπως δείξαμε παραπάνω στις ιδιότητες του HT.
- Επεξεργασία ήχου και εικόνας.
- Εφαρμογές ηλεκτρικής ισχύος που αφορούν ηλεκτρικά δίκτυα, γραμμικά χρονικά αμετάβλητα ηλεκτρικά συστήματα, καθώς και μετάδοση ηλεκτρομαγνητικών κυμάτων σε συστήματα μεταφοράς.
- Χρήσεις στην αστρονομία, γεωφυσική, χημική μηχανική κ.τ.λ. Οι περισσότερες από τις παραπάνω εφαρμογές απαιτούν τον υπολογισμό του ολοκληρώματος της συνέλιξης.
- Ο DHT χρησιμοποιεί πραγματικούς αριθμούς και όχι μιγαδικές πράξεις (λιγότεροι υπολογισμοί).
- Ο DHT χρησιμοποιεί τη μισή μνήμη αποθήκευσης (πραγματικοί έναντι μιγαδικών μητρών δεδομένων).
- Για μια ακολουθία μήκους N, ο γρήγορος DHT (FHT ανάλογος του FFT) χρειάζεται $O(Nlog_2N)$ πράξεις πραγματικών αριθμών, ενώ ο DFT χρειάζεται $O(Nlog_2N)$ πράξεις σε μιγαδικούς αριθμούς.
- Ο DHT οδηγεί σε λιγότερα λάθη αποκοπής και στρογγυλοποίησης λόγω λιγότερων πράξεων.

- Ο DHT είναι ο ίδιος του ο αντίστροφος μετασχηματισμός. Δεν απαιτείται ξεχωριστή υπορουτίνα σε γλώσσα προγραμματισμού για τον υπολογισμό του αντιστρόφου.

Το μεγαλύτερο, ίσως, πλεονέκτημα του DHT έναντι του DFT είναι η δυνατότητα επεξεργασίας πραγματικών δεδομένων. Καθώς τα περισσότερα προβλήματα που συναντούμε στην επεξεργασία των σημάτων επεξεργάζονται πραγματικά δεδομένα, είναι δυνατόν να επιτευχθούν μειώσεις του υπολογιστικού κόστους από την χρήση του FHT (Fast Hartley Transform). Αυτό αποδεικνύεται από το γεγονός ότι, από τον υπολογισμό του FFT μιας ακολουθίας N δεδομένων, το αποτέλεσμα θα είναι $2N$ πραγματικά (ή N μιγαδικά) δείγματα, τα μισά από τα οποία είναι πλεονασμός. Ο μετασχηματισμός Hartley από την άλλη πλευρά θα υπολογίσει μόνο N μιγαδικές τιμές, εκτελώντας έτσι μόνο τις μισές αριθμητικές πράξεις και χρησιμοποιώντας τη μισή μνήμη. Για λόγους λοιπόν υπολογιστικών πλεονεκτημάτων, οι οποίοι προκύπτουν είτε από τη συμμετρία της κυματομορφής, είτε από την χρήση πραγματικών ποσοτήτων, ο μετασχηματισμός Hartley προτιμάται αρκετές φορές ως εναλλακτικός του Fourier για την ανάλυση στο πεδίο των συχνοτήτων **[35]**. Το μειονέκτημα του μετασχηματισμού Hartley είναι ότι οι πληροφορίες για το εύρος και τη φάση του μετασχηματισμού Fourier δεν είναι άμεσα προσβάσιμες. Πάρα ταύτα, όπου αυτές οι πληροφορίες είναι αναγκαίες, μπορούν να ανακτηθούν από τις εξισώσεις (1.22) και (1.23) αντίστοιχα.

Θα πρέπει ωστόσο να σημειωθεί, ότι υπάρχουν ειδικοί FFT αλγόριθμοι οι οποίοι μετασχηματίζουν πραγματικές τιμές **[4]**, **[42]**. Αυτοί οι αλγόριθμοι εμφανίζονται εφάμιλλοι του FHT όσον αφορά την διαχείριση της μνήμης, ωστόσο εμφανίζονται λιγότερο ευέλικτοι καθόσον απαιτείται ένας διαφορετικός αλγόριθμος για τον υπολογισμό του αντίστροφου FFT. Σε αντίθεση ο FHT, ο οποίος εμφανίζει την ιδιότητα της αυτοαντιστροφής χρειάζεται μόνο έναν αλγόριθμο για την υλοποίηση του ευθέους ή του αντιστρόφου μετασχηματισμού Hartley. Ένας "αργός" κώδικάς, ο οποίος υλοποιεί τον μετασχηματισμό Hartley μέσω του ορισμού του φαίνεται στο παράρτημα "A" (dht.m).

1.9 Ο πολυμεταβλητός διακριτός μετασχηματισμός Hartley

Ο πολυδιάστατος διακριτός μετασχηματισμός Hartley μιας συνάρτησης $h(t_1,t_2,...,t_d)$, ορίζεται ανάλογα με τον αντίστοιχο πολυμεταβλητό Fourier μετασχηματισμό:

$$H(f_1,f_2,...,f_d)=\sum_{t_1=0}^{n_1-1}\sum_{t_2=0}^{n_2-1}\sum_{t_d=0}^{n_d-1}h(t_1,t_2,...,t_d)\times cas\,[2\pi(\frac{f_1t_1}{n_1}+\frac{f_2t_2}{n_2}+...+\frac{f_dt_d}{n_d})] \quad (1.47)$$

όπου $cas\,\omega=cos\omega+sin\omega$.

Ωστόσο, ο πυρήνας $cas\,[2\pi(\frac{f_1t_1}{n_1}+\frac{f_2t_2}{n_2}+...+\frac{f_dt_d}{n_d})]$ δεν είναι εύκολα διαχωρίσιμος όπως ο αντίστοιχος πυρήνας του FT:

$$\begin{aligned}&exp[i2\pi(\frac{f_1t_1}{n_1}+\frac{f_2t_2}{n_2}+...+\frac{f_dt_d}{n_d})]=\\&exp(i2\pi\frac{f_1t_1}{n_1})\times exp(i2\pi\frac{f_2t_2}{n_2})\times...\times exp(i2\pi\frac{f_dt_d}{n_d})\end{aligned} \quad (1.48)$$

Αυτός ο διαχωρισμός είναι η βάση της εφαρμογής ενός πολυμεταβλητού μετασχηματισμού Fourier με διαδοχικούς μονοδιάστατους FT.

Ο Bracewell στο **[10]** περιέγραψε ένα νέο σχήμα για να ξεπεράσει το πρόβλημα του μη διαχωρίσιμου πυρήνα του HT. Αυτό το σχήμα βασίζεται στην ακόλουθη τριγωνομετρική ταυτότητα:

$$\begin{aligned}2\,cas\,(a+\beta)=&cas\,a\,cas\,\beta+\\&+cas\,a\,cas\,(-\beta)+\\&+cas\,(-a)\,cas\,(\beta)-\\&-cas\,(-a)\,cas\,(-\beta)\end{aligned} \quad (1.49)$$

Χρησιμοποιώντας αυτό το σχήμα, μια πολυδιάστατη ακολουθία δεδομένων μετατρέπεται πρώτα με HT, δημιουργώντας τον παρακάτω προσωρινό μετασχηματισμό T:

$$\begin{aligned}&T(f_1,f_2,...,f_d)=\\&\sum_{t_1=0}^{n_1-1}\sum_{t_2=0}^{n_2-1}\sum_{t_d=0}^{n_d-1}h(t_1,t_2,...,t_d)\,cas\,\frac{2\pi f_1t_1}{n_1}\times cas\,\frac{2\pi f_2t_2}{n_2}\times...\times cas\,\frac{2\pi f_dt_d}{n_d}\end{aligned} \quad (1.50)$$

Χρησιμοποιώντας την ταυτότητα (1.49) μπορούμε να συνδυάζουμε διαδοχικά ζεύγη όρων cas σε ένα μοναδικό άθροισμα εκτελώντας $d-1$ περάσματα. Αυτός ο συνδυασμός των όρων μπορεί να εφαρμοστεί, χρησιμοποιώντας ένα σχήμα ροής

υπολογισμών που επαναχρησιμοποιεί τις ήδη υπολογισμένες τιμές για να υπολογίσει με απλές προσθαφαιρέσεις τις τιμές της συνάρτησης πυρήνα για διαφορετικά ορίσματα. Το σχήμα αυτό μοιάζει με τα φτερά μιας πεταλούδας (butterfly) και εκτελεί τέσσερεις συνδυασμούς σε ένα βήμα.

Το σχήμα αυτό μπορεί να βελτιωθεί για το πρόβλημα των τριών διαστάσεων όπως παρατήρησαν και οι Hong Hao και Bracewell **[11]**. Εάν το σχήμα συνδυάζει τρεις όρους *cas* κάθε φορά, τότε χρησιμοποιεί μόνο τέσσερεις από τους οκτώ όρους που προκύπτουν από το ανάπτυγμα:

$$\begin{aligned} 2\,cas\,(a+\beta+\gamma) = {} & cas\,(-a)\,cas\,\beta\,cas\,\gamma \\ & + cas\,a\,cas\,(-\beta)\,cas\,\gamma \\ & + cas\,a\,cas\,\beta\,cas\,(-\gamma) \\ & - cas\,(-a)\,cas\,(-\beta)\,cas\,(-\gamma) \end{aligned} \tag{1.51}$$

Αυτό το συμπέρασμα μπορεί να χρησιμοποιηθεί για να μειωθούν οι πράξεις, που απαιτούνται για τον συνδυασμό των όρων *cas*, σε πολυδιάστατους μετασχηματισμούς Hartley.

ΚΕΦΑΛΑΙΟ 2

ΑΛΓΟΡΙΘΜΟΙ ΓΡΗΓΟΡΟΥ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ HARTLEY

2.1 Εισαγωγή

Όπως αναπτύχθηκε παραπάνω, ο DHT μοιράζεται τις ίδιες ιδιότητες και έχει παρόμοιες εφαρμογές με τον DFT. Για αυτόν το λόγο, έχουν αναπτυχθεί αποδοτικοί αλγόριθμοι υπολογισμού του DHT παρόμοιοι με τον FFT. Η πρώτη δημοσιευμένη εργασία σε αυτόν τον τομέα θεωρείται ότι είναι αυτή του Bracewell **[7], [8]**, ο οποίος ανέπτυξε radix-2 και radix-4 αλγόριθμους για τον υπολογισμό του FHT. Αργότερα αναπτύχθηκαν και άλλοι αλγόριθμοι, είτε για λόγους που έχουν να κάνουν με την απόδοση, είτε με τη δυσκολία υλοποίησης. Παρακάτω θα αναφερθούμε συνοπτικά στους κυριότερους και θα υλοποιήσουμε σε κώδικα MatLab τον FHT radix-2 του Bracewell **[9]**.

2.2 Ο γρήγορος μετασχηματισμός Hartley (Fast Hartley Transform)

Ο Bracewell στο **[9]** παρουσίασε έναν γρήγορο αλγόριθμο υπολογισμού του HT, ανάλογου του FFT και απέδειξε ότι ο γρήγορος μετασχηματισμός Hartley έχει καλύτερη απόδοση από τον FT, με τον επιπλέον πλεονέκτημα ότι χειρίζεται μόνο πραγματικές τιμές και έτσι είναι εύκολος στην υλοποίησή του. Η βάση του FHT είναι η ακόλουθη αναδρομική σχέση:

$$H(f) = H_{even}(f) + \cos\frac{2\pi f}{n} H_{odd}(f) + \sin\frac{2\pi f}{n} H_{odd}(n - f) \tag{2.1}$$

όπου $H_{even}(f)$ είναι το διάνυσμα του HT όπου όλες οι τιμές έχουν άρτιους δείκτες και $H_{odd}(f)$ είναι το αντίστοιχο διάνυσμα με περιττούς δείκτες. Ως τερματισμός της αναδρομής ορίζεται το γεγονός, ότι ένα διάνυσμα με μια μοναδική τιμή είναι ο ίδιος του ο μετασχηματισμός Hartley, το οποίο ισχύει μόνο όταν το πλήθος των δεδομένων είναι δύναμη του δύο. Εάν όχι, τότε το αρχικό διάνυσμα δεδομένων

συμπληρώνεται με μηδενικά μέχρι την επόμενη δύναμη του δύο **[36].** Θα χρησιμοποιήσουμε μια τροποποιημένη μορφή του παραπάνω αλγορίθμου, που χρησιμοποιεί επανάληψη αντί της αναδρομής, για τον υπολογισμό του Radix-2 FHT. Η χρήση της επαναληπτικής μορφής προσφέρει το πλεονέκτημα της χρήσης παράλληλου κώδικα για επιτάχυνση των υπολογισμών.

Εφόσον ο αλγόριθμος βασίζεται στην εφαρμογή μετασχηματισμών Hartley στα μέρη του διανύσματος δεδομένων, όπου οι δείκτες είναι περιττοί ή άρτιοι αντίστοιχα, μπορούμε να διαχωρίσουμε τον αλγόριθμο σε δύο μέρη, με το πρώτο μέρος να ανατοποθετεί τις τιμές του διανύσματος σύμφωνα με τους δείκτες (άρτιους ή περιττούς) και το δεύτερο μέρος να υπολογίζει τα σταθμισμένα αθροίσματα της (2.1).

Υπάρχουν δύο στρατηγικές, οι οποίες βασίζονται στη σειρά εφαρμογής των μερών αυτών του αλγορίθμου **[36]**. Εμείς θα επιλέξουμε να ανατοποθετήσουμε πρώτα τις τιμές του διανύσματος δεδομένων και έπειτα να εφαρμόσουμε το μετασχηματισμό. Η απαραίτητη ανατοποθέτηση συνίσταται στην τοποθέτηση κάθε στοιχείου σε θέση που καθορίζεται από την bit-αντιστροφή του δείκτη του **[36]**.

Μετά την επανατοποθέτηση, το διάνυσμα περιέχει n μετασχηματισμένα διανύσματα μήκους 1. Το επόμενο βήμα του αλγορίθμου συνδυάζει δύο διαδοχικούς μικρούς μετασχηματισμούς, σε έναν διπλού μήκους. Σε $log(n)$ περάσματα του διανύσματος δεδομένων οι n μετασχηματισμοί μήκους 1 συνδυάζονται στον μετασχηματισμό του αρχικού διανύσματος.

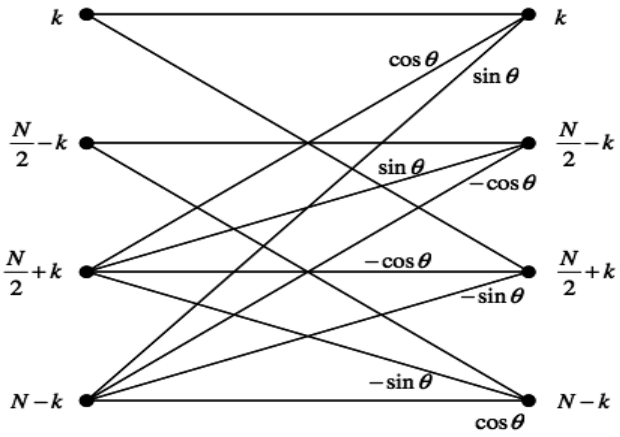
Ο σταθμισμένος συνδυασμός των δύο διαδοχικών μικρών μετασχηματισμών βασίζεται στις παρακάτω αρχές:

- Εξαιτίας της φύσης του μετασχηματισμού οι ακόλουθες εξισώσεις ισχύουν για όλα τα $f \in [0, n-1]$:

$$\begin{aligned} H_{even}(n/2+f) &= H_{even}(f) \\ H_{odd}(n/2+f) &= H_{odd}(f) \end{aligned} \tag{2.2}$$

- Εάν ο σταθμισμένος συνδυασμός όλων των τιμών στα $H(f)$, $H(n/2-f)$, $H(n/2+f)$, $H(n-f)$ γίνεται ταυτόχρονα, οι ημιτονοειδείς και συνημιτονοειδείς όροι της εξίσωσης (2.1) είναι αναγκαίο να υπολογιστούν μόνο μια φορά, για όλες αυτές τις τέσσερεις τιμές.

- Ο συνδυασμός αυτών των τεσσάρων τιμών μπορεί να υπολογιστεί χρησιμοποιώντας ένα σχήμα butterfly **[22],** το οποίο χρησιμοποιείται επίσης και στον FFT.



Σχήμα 3: *Διάγραμμα ροής πληροφορίας (butterfly) στον Radix-2 αλγόριθμο*

Το μόνο πρόβλημα που απομένει, για την επίλυση του παραπάνω αλγορίθμου, είναι ο γρήγορος υπολογισμός των ημιτονοειδών και συνημιτονοειδών όρων για τις διαδοχικές τετράδες τιμών. Στο **[36]** προτείνεται η ακόλουθη αναδρομή για τον υπολογισμό τους:

$$\begin{aligned} a &= 2sin((d/2)^2) \\ b &= sind \\ cos(t+d) &= cost - (a \times cost + b \times sint) \\ sin(t+d) &= sint - (a \times cost - b \times sint) \end{aligned} \tag{2.3}$$

Εναλλακτικά, οι ημιτονοειδείς και συνημιτονοειδείς τιμές μπορούν να προϋπολογιστούν για τις απαιτούμενες συχνότητες. Αυτό φυσικά προϋποθέτει ότι θα πρέπει να τηρείται ένας μεγάλος πίνακας δεδομένων, γεγονός το οποίο αυξάνει το κόστος σε πόρους μνήμης. Λόγω του ότι η αναδρομή γενικότερα είναι πιο αργή από την χρήση επαναλήψεων **[18]** και δεν προσφέρεται για παράλληλους υπολογισμούς, καθώς η επόμενη τιμή εξαρτάται από τον σειριακό υπολογισμό της προηγούμενης, μπορούμε με μικρές αλλαγές να γράψουμε τον αλγόριθμο που περιγράψαμε παραπάνω, σε κώδικα MatLab, μετετρέποντας την αναδρομή σε βρόγχους for (fht.m στο παράρτημα "A").

2.3 Ο κανονικοποιημένος αλγόριθμος FHT

Οι μέθοδοι οι οποίοι προτάθηκαν από τον Bracewell, είναι αρκετά ικανοποιητικοί και επιτυγχάνουν τα επιθυμητά επίπεδα απόδοσης, τόσο όσον αφορά την αριθμητική πολυπλοκότητα, όσο και τις απαιτήσεις μνήμης. Συγκρινόμενοι με τον κοινό μιγαδικό FFT, χρειάζονται μόνο τις μισές αριθμητικές πράξεις και τη μισή μνήμη. Έχουν όμως το πρόβλημα ότι απαιτούν δύο μεγεθών butterfly (και έτσι δυο διαφορετικούς αλγορίθμους) για τον αποδοτικό υπολογισμό **[25]**, **[8]**. Αυτή η ασυμμετρία προκαλεί μείωση της αποδοτικότητας του αλγορίθμου, ενώ δυσκολεύει την υλοποίησή του σε παράλληλες αρχιτεκτονικές **[25].**

Παρόλο που και άλλοι αλγόριθμοι έχουν προταθεί για τον αποδοτικό υπολογισμό του DHT **[15]**, **[16]**, **[39]**, όλοι παρουσιάζουν ως ένα ορισμένο σημείο, την παραπάνω ασυμμετρία. Υπάρχουν και άλλοι αρκετά γνωστοί αλγόριθμοι για τον υπολογισμό του FHT, όπως οι radix-2 και split radix-4 **[15]**, όμως ο radix-4 FHT είναι ο πιο αποδοτικός, καθώς μπορεί να χρησιμοποιηθεί για τον οκταπλασιασμό της ταχύτητας σε παράλληλους υπολογιστές **[25]**.

Για να αντιμετωπιστεί το πρόβλημα της παραπάνω ασυμμετρίας, ο K.J. Jones **[25]** ανέπτυξε έναν αλγόριθμο σταθερού μεγέθους butterfly για τον radix-4 FHT. Το αποτέλεσμα είναι ένας αναδρομικός αλγόριθμος, ο οποίος αναφέρεται κανονικοποιημένος FHT (regularised FHT). Ο παραπάνω αλγόριθμος μπορεί πολύ εύκολα να υλοποιηθεί, χρησιμοποιώντας παράλληλες αρχιτεκτονικές για την επιτάχυνση των υπολογισμών.

Για να καταλήξει στον αλγόριθμο αυτό, ο K.J. Jones διαίρεσε την εξίσωση (1.41) σε τέσσερα μερικά αθροίσματα.

$$\begin{aligned} X[k] = & \sum_{n=0}^{N/4-1} x[4n] \times cas\{2\pi(4n)k/N\} \\ & + \sum_{n=0}^{N/4-1} x[4n+1] \times cas\{2\pi(4n+1)k/N\} \\ & \sum_{n=0}^{N/4-1} x[4n+2] \times cas\{2\pi(4n+2)k/N\} \\ & \sum_{n=0}^{N/4-1} x[4n+3] \times cas\{2\pi(4n+3)k/N\} \end{aligned} \tag{2.4}$$

Όπου,

$$x1[n] = x[4n], \quad x2[n] = x[4n+1], \; x3[n] = x[4n+2], \\ x4[n] = x[4n+3] \tag{2.5}$$

Τα παραπάνω μερικά αθροίσματα της (9) γράφονται:

$$X1[k] = \sum_{n=0}^{N/4-1} x1[n] \times cas\{2\pi nk/(N/4)\} \tag{2.6}$$

$$X2[k] = \sum_{n=0}^{N/4-1} x2[n] \times cas\{2\pi nk/(N/4)\} \tag{2.7}$$

$$X3[k] = \sum_{n=0}^{N/4-1} x3[n] \times cas\{2\pi nk/(N/4)\} \tag{2.8}$$

$$X4[k] = \sum_{n=0}^{N/4-1} x4[n] \times cas\{2\pi nk/(N/4)\} \tag{2.9}$$

Το αποτέλεσμα στο οποίο καταλήγει συνοψίζεται στις παρακάτω σχέσεις:

$$X[0] = X1[0] + X2[0] + X3[0] + X4[0] \tag{2.10}$$

$$X[N/8] = X1[N/8] + \sqrt{2} \times X2[N/8] + X3[N/8] \tag{2.11}$$

$$X[N/4] = X1[0] + X2[0] - X3[0] - X4[0] \tag{2.12}$$

$$X[3N/8] = X1[N/8] - X3[N/8] + \sqrt{2} \times X4[N/8] \tag{2.13}$$

$$X[N/2] = X1[0] - X2[0] + X3[0] - X4[0] \tag{2.14}$$

$$X[5N/8] = X1[N/8] - \sqrt{2} \times X2[N/8] + X3[N/8] \tag{2.15}$$

$$X[3N/4] = X1[0] - X2[0] - X3[0] + X4[0] \tag{2.16}$$

$$X[7N/8] = X1[N/8] - X3[N/8] - \sqrt{2} \times X4[N/8] \tag{2.17}$$

Η παραπάνω μορφή των εξισώσεων συνοψίζει την radix-4 (με βάση το 4) πεταλούδα τύπου III η οποία περιλαμβάνει τρεις διαφορετικές μορφές διπλής πεταλούδας. Το διάγραμμα ροής δεδομένων στον κανονικοποιημένο αλγόριθμο φαίνεται στο σχήμα 4.

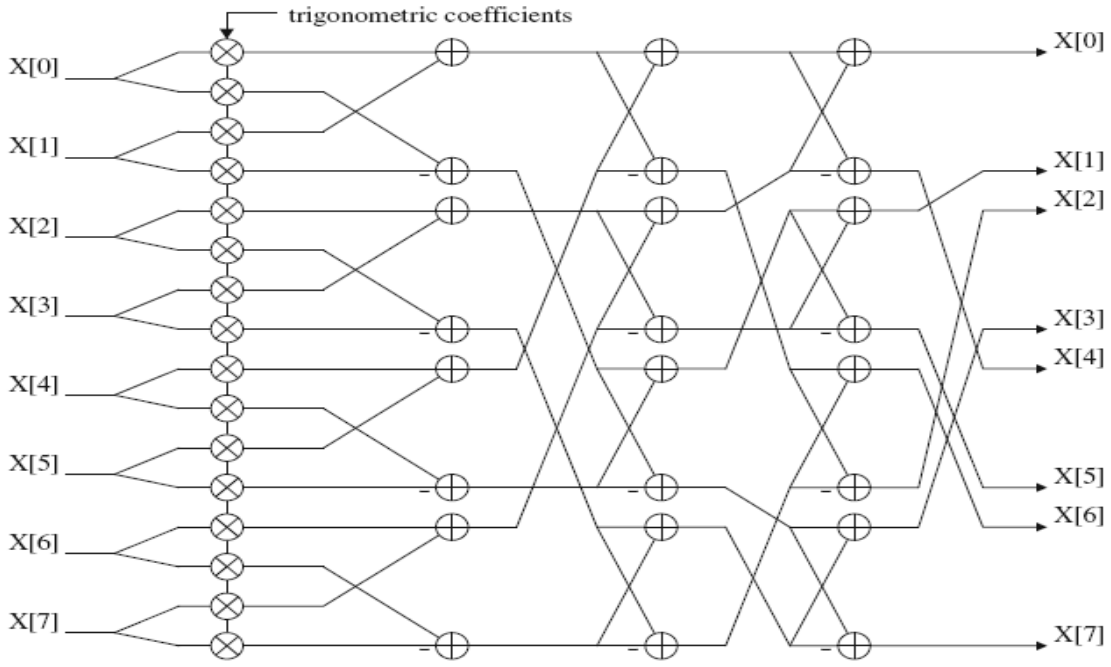
2.4 Υπολογισμός του 1-D και 2-D FFT μεσω FHT

Για τον υπολογισμό του διακριτού μετασχηματισμού Fourier από τον αντίστοιχο μετασχηματισμό Hartley χρησιμοποιήθηκε η σχέση (1.45). Υπολογίστηκαν 2 FHT, ένας για το πραγματικό και ένας για το φανταστικό μέρος του Fourier και εφαρμοστηκε η παραπάνω σχέση. Για να εφαρμοστεί η σχέση αυτή, το διάνυσμα δεδομένων εισόδου περιστράφηκε γύρω από το πρώτο στοιχείο και τοποθετήθηκε

στην επόμενη στήλη του πίνακα temp. Οι παραπάνω ενέργειες έγιναν τόσο για το φανταστικό όσο και το πραγματικό μέρος, ωστε με την αφαίρεση των διπλανών στηλών μεταξύ τους, να αφαιρούνται όλα τα στοιχεία *x[k] - x[N - k]* ταυτόχρονα. Ο αντίστροφος μετασχηματισμός υπολογίζεται απλά, αλλάζοντας το πρόσημο του μετασχηματισμου Fourier. Ο κώδικάς φαίνεται στο παράρτημα "A" (cfht2fft.m).

Για τον υπολογισμό του 2-D μετασχηματισμού Fourier προτιμήθηκε η μέθοδος γραμμής - στήλης. Εφαρμόστηκε δηλαδή διαδοχικά ένας FFT πρώτα στις στήλες και έπειτα στις γραμμές. Ο κώδικάς φαίνεται στο παράρτημα "A" (fft2d.m και ifft2d.m). Φυσικά υπάρχουν πιο εξελιγμένοι αλγόριθμοι από αυτόν της γραμμής - στήλης, για τον υπολογισμό του πολυδιάστατου FFT, αλλά εδώ προτιμήθηκε για λόγους απλότητας.

Σχήμα 4: *Διάγραμμα ροής πληροφορίας στον κανονικοποιημένο Radix-4 αλγόριθμο [25]*



2.5 Σύγκριση αποδοτικότητας των αλγορίθμων υπολογισμού του FFT

Η σύγκριση των ενσωματωμένων αλγορίθμων του Matlab για FFT με τον κώδικα για FFT, που αναπτύχθηκε παραπάνω και για πίνακες διαφόρων διαστάσεων, φαίνεται στον παρακάτω πίνακα (υπολογιστικός χρόνος σε sec).

Πίνακας 3: *Αποτελέσματα απόδοσης FFT του Matlab και FFT μέσω FHT*

Dim	1D Matlab FFT	1D Hartley FFT	2D Matlab FFT	2D Hartley FFT
2	0,000109	0,003390	0,000155	0,004786
4	0,000089	0,001153	0,000125	0,008017
8	0,000063	0,001028	0,000123	0,017976
16	0,000076	0,001258	0,000158	0,035923
32	0,000380	0,001481	0,000217	0,071724
64	0,000048	0,001720	0,000224	0,160619
128	0,000039	0,002148	0,001150	0,398542
256	0,004657	0,002928	0,003867	1,154917
512	0,003377	0,005740	0,020464	3,957442
1024	0,000808	0,009020	0,078908	15,754228
2048	0,009030	0,019411	0,322893	64,835985
4096	0,026484	0,043076	1,206559	264,890956

Είναι εμφανές οτι ο κώδικας 1-D FFT με μετασχηματισμό Hartley χρειάζεται σχεδόν διπλάσιο χρόνο, για τον υπολογισμό, από οτι ο ενσωματωμένος του Matlab. Αυτό οφείλεται στο οτι στον κώδικα υλοποιήθηκε ο radix-2 FHT που πρωτοπαρουσίασε ο Bracewell, ενώ υπάρχουν σήμερα πιο πολύπλοκες και γρήγορες υλοποιήσεις (Regularized FHT). Παρα ταυτα, ο χρόνος υπολογισμού του αλγόριθμου φαίνεται να αυξάνει γραμμικά, σε σχέση με το μέγεθος των δεδομένων εισόδου και έτσι μπορεί κάλλιστα να χρησιμοποιηθεί και για μεγάλους πίνακες χωρίς υψηλό υπολογιστικό κοστος, ειδικά αν σκεφτεί κανείς την απλότητα υλοποίησής του.

Αντίθετα, ο 2-D FFT με μετασχηματισμό Hartley εμφανίζεται σαφώς κατώτερος της αντίστοιχης ενσωματωμένης συνάρτησης του Matlab. Φαίνεται μάλιστα να αυξάνει εκθετικά τον απαιτούμενο χρόνο υπολογισμού. Αυτό οφείλεται στην μη αποδοτική μέθοδο γραμμής στήλης, που εφαρμόστηκε. Η μεγαλη καθυστέρηση στους υπολογισμούς οφείλεται κυρίως στην εφαρμογή nested for loops, οι οποίοι δεν είναι δυνατό να γραφούν λόγω της δομής τους, με vectorized τρόπο στο Matlab. Αυτό το σημείο του αλγορίθμου είναι και το πιο υπολογιστικά εντατικό, προσφέρεται όμως για παραλληλοποίηση όπως θα δούμε στο κεφάλαιο 4.

ΚΕΦΑΛΑΙΟ 3

ΠΟΛΥΩΝΥΜΙΚΟΙ ΠΙΝΑΚΕΣ ΚΑΙ ΠΟΛΥΩΝΥΜΙΚΗ ΠΑΡΕΜΒΟΛΗ

3.1 Εισαγωγή

Η εύρεση της ορίζουσας ενός πολυωνυμικού πίνακα έχει πολλές εφαρμογές στη θεωρία των συστημάτων, καθώς μπορεί να χρησιμοποιηθεί ως ενδιάμεσο στάδιο για τον υπολογισμό του αντιστρόφου ενός πολυωνυμικού πίνακα. Για τον υπολογισμό της ορίζουσας πολυωνυμικού πίνακα έχουν προταθεί διάφορες μέθοδοι, όπως οι συμβολικές και οι μέθοδοι Laverier-Fanteev **[37]**. Οι συμβολικές μέθοδοι έχουν κυρίως εφαρμογή σε γλώσσες ανωτέρου επιπέδου όπως το Matlab, το Mathematica, το Maple κ.τ.λ., ενώ οι μέθοδοι Laverier-Fanteev έχουν το μειονέκτημα ότι δεν είναι τόσο ευσταθείς, εάν εφαρμοστούν με την χρήση γλωσσών γενικής χρήσης όπως η C ή η Fortran **[28]**. Για να ξεπεραστούν αυτές οι δυσκολίες μπορούμε να χρησιμοποιήσουμε άλλες τεχνικές, όπως είναι οι μέθοδοι παρεμβολής. Οι μέθοδοι παρεμβολής μπορούν να χρησιμοποιηθούν τόσο για τον υπολογισμό της ορίζουσας του πολυωνυμικού πίνακα, όσο και για τον υπολογισμό του adjoint του. Έτσι είναι δυνατό να υπολογιστεί και ο αντίστροφος του. Ωστόσο αν χρειαστεί να αυξήσουμε την ταχύτητα και τη σταθερότητα των αλγόριθμων, πρέπει να χρησιμοποιήσουμε διακριτούς μετασχηματισμούς (Fourier Transforms). Παρακάτω θα ασχοληθούμε με την παρεμβολή σε ένα πολυωνυμικό πίνακα για τον υπολογισμό της ορίζουσάς του, με τη χρήση μετασχηματισμού FFT μέσω FHT, ο οποίος αναπτύχθηκε στο κεφάλαιο 2.

3.2 Πολυωνυμικοί πίνακες

3.2.1 Ορισμοί

Ως πολυωνυμικός πίνακας ορίζεται ο πίνακας, του οποίου τα στοιχεία είναι πολυώνυμα μιας ή περισσοτέρων μεταβλητών. Ένας πολυωνυμικός πίνακας μιας μεταβλητής περιγράφεται από την παρακάτω σχέση:

$$P = \sum_{n=0}^{p} A(n)x^n = A(0) + A(1)x + A(2)x^2 + \ldots + A(p)x^p \tag{3.1}$$

όπου $A(i)$ ορίζει ένα σταθερό πίνακα με $A(p)$ διάφορο του μηδενός. Ένα παράδειγμα πολυωνυμικού πίνακα, μιας μεταβλητής, φαίνεται παρακάτω:

$$P = \begin{pmatrix} 1 & x^2 & x \\ 0 & 2x & 2 \\ 3x+2 & x^2\text{-}1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 2 & -1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 3 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} x^2 \tag{3.2}$$

Η ορίζουσα ενός τετραγωνικού πολυωνυμικού πίνακα $n \times n$, η οποία απεικονίζεται από την σχέση $p(s) = detP(s)$, ορίζεται ως:

$$p(s) = p_0 + p_1 s + \ldots + p_k s^k \tag{3.3}$$

το οποίο είναι ένα πολυώνυμο βαθμού $k \le np$. Αντικαθιστώντας στη σχέση του πολυωνυμικού πίνακα πραγματικές ή μιγαδικές τιμές $s_i, i = 0,1,\ldots,k$ λαμβάνουμε $k+1$ σταθερές $p(s_i) = detP(s_i)$, οι οποίες υπολογίζονται ως οι ορίζουσες σταθερών πινάκων.

Ένας πολυωνυμικός πίνακας με ορίζουσα διάφορη του μηδενός, έχει αντίστροφο ο οποίος είναι και αυτός πολυωνυμικός πίνακας. Οι ρίζες του πολυωνυμικού πίνακα στο μιγαδικό επίπεδο είναι τα σημεία όπου ο βαθμός του πίνακα μικραίνει.

3.2.2 Οι πολυωνυμικοί πίνακες στη θεωρία ελέγχου

Η πολυωνυμική προσέγγιση, στην επεξεργασία σημάτων και στον έλεγχο γραμμικών συστημάτων, είναι πλέον ένα καθιερωμένο πεδίο μελέτης της θεωρίας ελέγχου. Τα γραμμικά συστήματα περιγράφονται από πολυωνυμικούς πίνακες και εξισώσεις πολυωνυμικών πινάκων. Επιπρόσθετα, πολλά προβλήματα ελέγχου απαιτούν τον υπολογισμό της ορίζουσας ενός πολυωνυμικού πίνακα. Για παράδειγμα στον εύρωστο έλεγχο 2-D MIMO συστημάτων. Επίσης, ο έλεγχος της ευστάθειας ενός πολυωνυμικού πίνακα πραγματοποιείται μέσω του υπολογισμού της ορίζουσάς του **[27]**. Ο υπολογισμός των χαρακτηριστικών πολυωνύμων ενός ανοιχτού ή κλειστού MIMO συστήματος, απαιτεί τον υπολογισμό της ορίζουσας. Τέλος, οι πόλοι ενός MIMO συστήματος συχνά βρίσκονται υπολογίζοντας τις ρίζες της ορίζουσας ενός πολυωνυμικού πίνακα.

Ας θεωρήσουμε για παράδειγμα το γραμμικό σύστημα συνεχούς χρόνου, με την ακόλουθη συνάρτηση μεταφοράς:

$$A = N(s)D^{-1}(s) = \begin{pmatrix} 1 & s-1 \end{pmatrix} \begin{pmatrix} s+1 & 1 \\ 2 & (s+1)^2 \end{pmatrix}^{-1} \tag{3.4}$$

Για να αποφανθούμε για την ευστάθεια αυτού του συστήματος, θα πρέπει να υπολογίσουμε την ορίζουσα του $D(s)$ και έπειτα να εφαρμόσουμε τα γνωστά κριτήρια ευστάθειας των πολυωνύμων, όπως για παράδειγμα τα κριτήρια Hurwitz ή Routh. Κατά αυτόν τον τρόπο η ορίζουσα του παραπάνω συστήματος είναι $D(s) = -1 + 3s + 3s^2 + s^3$, ή οποία είναι προφανώς μη ευσταθής, καθότι τα πρόσημα των συντελεστών μεταβάλλονται.

Για τον υπολογισμό των οριζουσών πολυωνυμικών πινάκων έχουν αναπτυχθεί διάφορες μέθοδοι όπως οι συμβολικές, κατά τις οποίες ένας πολυωνυμικός πίνακας μπορεί να αναπαρασταθεί με μια συμβολική γλώσσα, όπως το MatLab. Σε αυτή την περίπτωση τα αποτελέσματα του υπολογισμού της ορίζουσας είναι απολύτως ακριβή και για μικρών διαστάσεων πίνακες οι υπολογιστικοί χρόνοι είναι αποδεκτοί. Ωστόσο, για μεγαλύτερο πλήθος δεδομένων, πράγμα που συναντάται αρκετά συχνά στην πράξη, οι υπολογιστικοί χρόνοι γίνονται αρκετά μεγάλοι και καθιστούν αυτήν την προσέγγιση πρακτικά ανέφικτη.

Για το λόγο αυτό έχουν καταβληθεί προσπάθειες για την ανάπτυξη γρήγορων και αξιόπιστων αλγεβρικών ή αριθμητικών αλγορίθμων για πολυωνυμικούς πίνακες. Μερικά παράδειγματα είναι η τροποποιημένη μέθοδος Fadeev-Leverrier (MFL), η μέθοδος αναγωγής ενός πίνακα σε τριγωνικό, καθώς και η πολυωνυμική παρεμβολή, οι οποίες παρουσιάστηκαν στο **[37]**. Κατά την εφαρμογή των αλγορίθμων παρεμβολής, τα πολυώνυμα αναπαρίστανται από τους συντελεστές τους και οι υπολογισμοί γίνονται με μεθόδους αριθμητικής ανάλυσης. Η χρήση της παρεμβολής με αναδρομικό τρόπο έχει αποδειχθεί εξαιρετικά αποδοτική για αυτόν τον σκοπό. **[37]**.

3.3 Η πολυωνυμική παρεμβολή

3.3.1 *Ορισμοί*

Η παρεμβολή αποτελεί μια από τις πλέον διαδεδομένες προσεγγιστικές τεχνικές στον τομέα της αριθμητικής ανάλυσης και των υπολογιστικών μαθηματικών. Με τον όρο παρεμβολή, εννοούμε το πρόβλημα της προσέγγισης μιας συνάρτησης f, της οποίας είναι γνωστές οι τιμές σε διακεκριμένα σημεία $x_i, i = 0,1,2,\ldots,n$, από μια άλλη συνάρτηση g πιο εύχρηστη. Όταν η προσεγγιστική συνάρτηση που χρησιμοποιείται είναι πολυώνυμο, τότε η μέθοδος καλείται «πολυωνυμική παρεμβολή». Το κλασσικό πρόβλημα παρεμβολής με πολυώνυμα διατυπώνεται όπως παρακάτω:

Πρόβλημα 1: Έστω $n+1$ διακεκριμένα σημεία $x_0, x_1, \ldots, x_n$, στα οποία οι τιμές $f(x_0), f(x_1), \ldots, f(x_n)$, της συνάρτησης f είναι γνωστές. Να βρεθεί πολυώνυμο $p_n(x)$ βαθμού n, το οποίο να εμφανίζει τις ίδιες τιμές με την f στα ίδια $n+1$ σημεία. Δηλαδή ψάχνουμε ένα πολυώνυμο $p_n(x)$ τέτοιο ώστε να ικανοποιεί τις πιο κάτω συνθήκες παρεμβολής

$$p_n(x_i) = f(x_i), \text{ για } i = 0,1,\ldots,n \tag{3.5}$$

Τα σημεία $x_0, x_1, \ldots, x_n$ καλούνται σημεία ή κόμβοι παρεμβολής, ενώ το $p_n(x)$ «πολυώνυμο παρεμβολής» βαθμού n.

Αποδεικνύεται οτι το πολυώνυμο αυτό είναι μοναδικό. Το αποτέλεσμα αυτό συνοψίζεται στο παρακάτω θεώρημα:

Θεώρημα 3.1: Για οποιοδήποτε σύνολο $n+1$ διακεκριμένων σημείων $x_0, x_1, \ldots, x_n$ και των αντίστοιχων τιμών της συνάρτησης f υπάρχει μοναδικό πολυώνυμο $p(x) \in P_n$ τέτοιο ώστε

$$f(x_i) = p(x_i) \text{ για } i = 0,1,\ldots,n$$

Απόδειξη: Ένα πολυώνυμο

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

ικανοποιεί τις συνθήκες παρεμβολής, σχέση (3.5), αν και μόνο αν οι συντελεστές a_i αποτελούν λύση του γραμμικού συστήματος

$$a_0 + a_1 x_0 + \ldots\ldots + a_n x_0^{\,n} = f(x_0)$$
$$a_0 + a_1 x_1 + \ldots\ldots + a_n x_1^{\,n} = f(x_1)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_0 + a_1 x_n + \ldots\ldots + a_n x_n^{\,n} = f(x_n)$$

Αυτό το σύστημα έχει μοναδική λύση, αν και μόνο αν η ορίζουσα του δεν μηδενίζεται. Μια τέτοια ορίζουσα καλείται ορίζουσα του Vandermonde και συμβολίζεται με $VDM(x_0,x_1,\ldots,x_n)$. Έτσι

$$VDM(x_0,x_1,\ldots,x_n) = \begin{vmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} & x_1^n \\ . & . & . & \ldots & . & . \\ . & . & . & \ldots & . & . \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} & x_n^n \end{vmatrix}$$

με $VDM(x_0,x_1,\ldots,x_n) = \prod_{0 \le i < j \le n} (x_i - x_j)$.

Αφού τα σημεία παρεμβολής είναι διακεκριμένα, δηλαδή $x_i \neq x_j$ για $i \neq j$, τότε η ορίζουσα δεν μηδενίζεται και το πρόβλημα έχει μια και μοναδική λύση. ▲

3.3.2 Η χρήση του DFT στην παρεμβολή

Είναι αρκετά διαδεδομένη η άποψη ότι λιγότερες αριθμητικές πράξεις οδηγούν σε λιγότερα λάθη στρογγυλοποίησης. Για αυτό το λόγο, αλλά και λόγους εξοικονόμησης υπολογιστικών πόρων, προτιμώνται οι αλγόριθμοι οι οποίοι οδηγούν σε λιγότερες αριθμητικές πράξεις, όπως είναι αυτοί της παρεμβολής. Από την άλλη μεριά, η ακρίβεια της εφαρμογής ενός αλγορίθμου παρεμβολής εξαρτάται επίσης από την σωστή επιλογή των σημείων της. Οι Singhal και Vlach **[38]** απέδειξαν ότι η χρήση μιγαδικών σημείων παρεμβολής είναι εξαιρετικά πιο σταθερή από την παρεμβολή πραγματικών αριθμών. Απέδειξαν επίσης, ότι η πραγματική παρεμβολή καταρρέει πολύ γρήγορα και δεν μπορεί να χρησιμοποιηθεί πρακτικά για πολυωνυμικούς βαθμούς πάνω από 20. Τέλος, κατέληξαν στο συμπέρασμα ότι η παρεμβολή στο μοναδιαίο κύκλο είναι ανώτερη όλων των άλλων διαδικασιών παρεμβολής. Η παρεμβολή Lagrange στον μοναδιαίο κύκλο μειώνει τις αριθμητικές

πράξεις καθώς και τα λάθη, λόγω επιλογής των μιγαδικών σημείων πάνω σε αυτόν. Αυτός ο τρόπος παρεμβολής είναι ουσιαστικά ένας διακριτός μετασχηματισμός Fourier, ο οποίος μπορεί να υπολογιστεί με την χρήση FFT και ο οποίος αποδεικνύεται γρηγορότερος και πιο ακριβής για μεγαλύτερους πολυωνυμικούς βαθμούς **[14].** Για τους λόγους αυτούς, η παρεμβολή στο μοναδιαίο κύκλο είναι η μέθοδος που θα χρησιμοποιήσουμε παρακάτω, για την εύρεση της ορίζουσας πολυωνυμικού πίνακα.

Τα κύρια πλεονεκτήματα των αλγορίθμων παρεμβολής, που χρησιμοποιούν τον διακριτό μετασχηματισμό Fourier, συνοψίζονται όπως παρακάτω:

- Είναι πολύ αποδοτικοί αλγόριθμοι, διαθέσιμοι σε μορφή λογισμικού και υλικού.
- Μπορούν να χρησιμοποιηθούν για παραλληλοποίηση των υπολογισμών και την ελαχιστοποίηση του χρόνου που απαιτείται.

Ειδικά τις τελευταίες δύο δεκαετίες σημειώθηκε εκτεταμένη χρήση των διακριτών μετασχηματισμών Fourier, κυρίως λόγω της υπολογιστικής ταχύτητας και ακρίβειάς τους. Μερικά παραδείγματα είναι ο υπολογισμός της συνάρτησης μεταφοράς ενός γενικευμένου πολυδιάστατου συστήματος **[1]** και η λύση διοφαντικών εξισώσεων με πολυωνυμικούς πίνακες.

3.3.3 Υπολογισμός ορίζουσας πολυωνυμικού πίνακα με παρεμβολή

Οι N. Karampetakis και A. Evripidou **[28]** παρουσίασαν έναν αλγόριθμο υπολογισμού του αντιστρόφου ενός πολυωνυμικού πίνακα 2 διαστάσεων και κατέληξαν, ότι ο υπολογισμός της ορίζουσας και του αντιστρόφου ενός $n \times n$ πίνακα γίνονται αποδοτικότερα, με την χρήση παρεμβολής με μετασχηματισμό FFT. Ο μετασχηματισμός FFT χρησιμοποιήθηκε για να λυθεί το πρόβλημα της μη ευσταθούς παρεμβολής Hermite-Lagrange για πολυδιάστατους πολυωνυμικούς πίνακες, καθώς και της μη αποδοτικότητάς της από άποψη χρόνου. Ο τρόπος με τον οποίο κατέληξαν στον αλγόριθμο συνοψίζεται παρακάτω:

Έστω ένας τετραγωνικός πολυωνυμικός πίνακας $A(x,y)$, με διαστάσεις $n \times n$, με στοιχεία $a_{i,j}$, πολυώνυμα 2 μεταβλητών x και y, του οποίου θέλουμε να υπολογίσουμε την ορίζουσα. Δηλώνουμε με k τον μέγιστο βαθμό του πίνακα

$A(x,y)$ ως προς την μεταβλητή x και με l τον μέγιστο βαθμό του πίνακα ως προς την μεταβλητή y. Έστω $p(x,y)$ το πολυώνυμο που αποτελεί την ορίζουσα του $A(x,y)$. Τότε ο μέγιστος βαθμός του πολυώνυμου $p(x,y)$ ως προς τη μεταβλητή x θα είναι

$$M_1 = k \times n \tag{3.6}$$

ενώ ως προς την μεταβλητή y θα είναι

$$M_2 = l \times n \tag{3.7}$$

Έστω οι πεπερασμένες ακολουθίες $X(k_1,k_2)$ και $\tilde{X}(r_1,r_2), k_i,r_i = 0,1,\ldots,M_i$. Για να αποτελούν αυτές οι δύο ακολουθίες ένα ζεύγος DFT, οι ακόλουθες εξισώσεις θα πρέπει να ισχύουν:

$$\tilde{X}(r_1,r_2) = \sum_{k_1=0}^{M_1}\sum_{k_2=0}^{M_2} X(k_1,k_2) W_1^{-k_1 r_1} W_2^{-k_2 r_2} \tag{3.8}$$

$$X(k_1,k_2) = \frac{1}{R}\sum_{r_1=0}^{M_1}\sum_{r_2=0}^{M_2} \tilde{X}(r_1,r_2) W_1^{k_1 r_1} W_2^{k_2 r_2} \tag{3.9}$$

Όπου

$$W_i = e^{\frac{2\pi j}{M_i+1}} \forall i = 1,2 \tag{3.10}$$

$$R = (M_1+1)(M_2+1) \tag{3.11}$$

Οι $X, \tilde{X}$ είναι συναρτήσεις πινάκων διακριτών ορισμάτων με διαστάσεις $n \times n$. Οι μιγαδικοί αριθμοί W_i, οι οποίοι ορίζονται στην εξίσωση (3.10) καλούνται σημεία Fourier. Η εξίσωση (3.8) είναι ο ευθύς μετασχηματισμός Fourier ενώ η εξίσωση (3.9) ο αντίστροφός του. Παρακάτω παρουσιάζεται ο αλγόριθμος παρεμβολής με χρήση FFT.

<u>*Αλγόριθμος 1:*</u> Υπολογισμός ορίζουσας πολυωνυμικού πίνακα $A(x,y)$ με την χρήση DFT.

Βήμα 1: Υπολογίζουμε τα M_1, M_2, τους μέγιστους βαθμούς ως προς x και y, αντίστοιχα της ορίζουσας $p(x,y)$ για $A(x,y) \in \mathfrak{R}[x,y]^{n\times n}$. Έστω

$$l = max\{deg_x a_{ij}(x,y), i,j = 1,2,\ldots,n\}$$

$$k = max\{deg_y a_{ij}(x,y), i,j = 1,2,\ldots,n\}$$

Τότε $M_1 = nl,\ M_2 = nk$. Οπότε το πολυώνυμο παρεμβολής $p(x,y)$ δίνεται από την εξίσωση:

$$p(x,y) = \sum_{i_1=0}^{M_1} \sum_{i_2=0}^{M_2} p_{i_1,i_2} x^{i_1} y^{i_2}$$

Βήμα 2 : Υπολογίζουμε το πλήθος των σημείων παρεμβολής από τη σχέση

$$R = (M_1 + 1)(M_2 + 1)$$

Βήμα 3 : Ορίζουμε τα σημεία παρεμβολής $(x,y) = (u_1(r_1), u_2(r_2))$ όπου

$$u_i(r_j) = W_i^{-rj},\ W_i = e^{\frac{2\pi j}{M_i+1}},\ \ i = 1,2$$

$$r_1 = 0,1,\ldots,M_1 \text{ και } r_2 = 0,1,\ldots,M_2$$

Βήμα 4: Αντικαθιστούμε τα σημεία $(u_1(r_1), u_2(r_2))$ στον πίνακα $A(x,y)$ και υπολογίζουμε την ορίζουσα σε κάθε σημείο

$$\tilde{p}_{r_1 r_2} = detA(u_1(r_1), u_2(r_2)) = \sum_{i_1=0}^{M_1} \sum_{i_2=0}^{M_2} p_{i_1,i_2} W_1^{-i_1 r_1} W_2^{-i_2 r_2}$$

Αν όλες οι τιμές $\tilde{p}_{r_1,r_2}$ είναι κοντά στο μηδέν και κάτω από μια τιμή κατωφλίου, η οποία εξαρτάται από την επιθυμητή ακρίβεια των υπολογισμών, η ορίζουσα του πολυωνυμικού πίνακα είναι μηδενική και ως εκ τούτου ο πίνακας δεν είναι αντιστρέψιμος. Είναι προφανές ότι τα $[\tilde{p}_{r_1,r_2}]$ και $[\tilde{p}_{i_1,i_2}]$ αποτελούν ζεύγος μετασχηματισμού Fourier.

Βήμα 5: Χρησιμοποίηση του αντίστροφου DFT ώστε να υπολογιστούν οι συντελεστές p_{i_1,i_2}

$$p_{i_1,i_2} = \frac{1}{R} \sum_{r_1=0}^{M_1} \sum_{r_2=0}^{M_2} \tilde{p}_{r_1,r_2} W_1^{r_1 i_1} W_2^{r_2 i_2} \tag{3.12}$$

όπου $l_i = 0,1,\ldots,M_i,\ i = 1,2$ και $R = (M_1 + 1)(M_2 + 1)$

Βήμα 6 : Υπολογισμός της ορίζουσας από την σχέση

$$p(x,y) = \sum_{i_1=0}^{M_1} \sum_{i_2=0}^{M_2} p_{i_1,i_2} x^{i_1} y^{i_2}$$

Ο παραπάνω αλγόριθμος εφαρμόστηκε στην γλώσσα προγραμματισμού MatLab, χρησιμοποιώντας τις συναρτήσεις για FFT με τη χρήση FHT, που αναπτύχθηκαν στο κεφάλαιο 2. Ο κώδικας φαίνεται στο παράρτημα "Α"

(interpolated_determinant.m). Κατά τη συγγραφή του κώδικα έγινε προσπάθεια να αποφευχθούν οι βρόγχοι for, προϋπολογίζοντας τους δείκτες i, j των δεδομένων του πίνακα παρεμβολής, προς κέρδος υπολογιστικού χρόνου. Ο αλγόριθμος για πίνακα 2 διαστάσεων μπορεί να επεκταθεί πολύ εύκολα και για πολυδιάστατους πολυωνυμικούς πίνακες χρησιμοποιώντας ακριβώς τα ίδια βήματα. Οι Hromcik και Sebek παρουσίασαν στο **[21]** έναν παρόμοιο αλγόριθμο υπολογισμού της ορίζουσας ενός πολυδιάστατου πολυωνυμικού πίνακα με την χρήση FFT.

3.3.4 Πειραματικός έλεγχος

Υπολογιστικοί έλεγχοι εφαρμόστηκαν για την σύγκριση της αποδοτικότητας του αλγορίθμου με συμβολική γλώσσα και με χρήση παρεμβολής με FHT, για τυχαίους πολυωνυμικούς πίνακες 2 διαστάσεων, μεγέθους 8x8 και διαφόρων πολυωνυμικών βαθμών. Για την δημιουργία τυχαίων πολυωνυμικών πινάκων χρησιμοποιήθηκε ο κώδικας του παραρτήματος "A" (create_bivariate_poly_matrix.m). Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα (υπολογιστικός χρόνος σε sec).

Πίνακας 4: *Αποτελέσματα απόδοσης πολυωνυμικής παρεμβολής*

Max Polynomial Power	Sequencial Symbolic Calculation	Sequencial FHT Interpolation
1	0,547466	3,530134
2	2,160438	30,531318
3	5,862685	76,409196
4	13,626698	134,926233
5	30,142386	230,071147
6	52,712246	293,271465
7	85,538860	463,024805
8	123,510596	570,499409
9	209,553168	745,836339
10	278,800820	924,649908

Παρατηρούμε οτι ο αλγόριθμος της παρεμβολής χρειάζεται σχεδόν τον τριπλάσιο χρόνο για να υπολογίσει την ορίζουσα του πολυωνυμικού πίνακα. Αυτό οφείλεται στο ότι το εντατικό υπολογιστικά μέρος του αλγορίθμου είναι η αντικατάσταση

των σημείων παρεμβολής στον τύπο υπολογισμού της ορίζουσας και ο υπολογισμός αυτής. Αυτό το παρατηρεί κανείς εύκολα αν συγκρίνει τους χρόνους υπολογισμού του FFT από τον πίνακα 3, με τους χρόνους υπολογισμού της ορίζουσας από τον πίνακα 4. Ο υπολογισμός των σημείων παρεμβολής είναι όμως εύκολα διαχωρίσιμος σε μικρότερους υπολογισμούς, που μπορούν να υπολογιστούν παράλληλα, με αποτέλεσμα μεγάλα οφέλη στον υπολογιστικό χρόνο, όπως θα δείξουμε στο κεφάλαιο 4.

Από την άλλη πλευρά, ο αλγόριθμος παρεμβολής μπορεί να βελτιωθεί περαιτέρω, αν επιλέξουμε το ελάχιστο αναγκαίο πλήθος σημείων παρεμβολής. Στον παραπάνω αλγόριθμο επιλέξαμε ένα άνω φράγμα για το πλήθος των σημείων παρεμβολής, το οποίο εξαρτάται από το μέγεθος του πίνακα και το οποίο επιδρά αρνητικά στον υπολογιστικό χρόνο, όσο αυξάνεται το μέγεθος του. Οι Henrion και Sebek στο **[20]** παρουσίασαν έναν αλγόριθμο για τον υπολογισμό του πολυωνυμικού βαθμού δ της ορίζουσας ενός πολυωνυμικού πίνακα, χρησιμοποιώντας πίνακες Toeplitz, ο οποίος οδηγεί σε ένα αποδοτικότερο τρόπο επιλογής του πλήθους των σημείων παρεμβολής. Ετσι, λαμβάνοντας μόνο το απαραίτητο πλήθος σημείων παρεμβολής είναι δυνατό να μειωθεί δραστικά το υπολογιστικό κόστος.

3.3.5 Συμπεράσματα

Σε αυτό το κεφάλαιο παρουσιάστηκε ένας αλγόριθμος υπολογισμού της ορίζουσας ενός 2-D πολυωνυμικού πίνακα με τη χρήση παρεμβολής και μετασχηματισμού Hartley. Ο αλγόριθμος εφαρμόστηκε σε MatLab και συγκρίθηκε υπολογιστικά με την ενσωματωμένη συνάρτηση συμβολικού υπολογισμού. Η υλοποίηση του MatLab εμφανίζεται πιο αποδοτική λόγω της χρησιμοποίησης ενός απλοϊκού radix-2 αλγορίθμου FHT και λόγω του μεγάλου υπολογιστικού κόστους κατά τον υπολογισμό της ορίζουσας, στα σημεία παρεμβολής. Το υπολογιστικό κόστος μπορεί να μειωθεί με τη χρήση παράλληλων υπολογισμών, τόσο κατά τον υπολογισμό του FHT όσο και κατά τον υπολογισμό της ορίζουσας στα σημεία παρεμβολής. Όπως δείξαμε στο κεφάλαιο 2, έχουν μελετηθεί αλγόριθμοι FHT οι οποίοι είναι έως και 8 φορές ταχύτεροι από τον αλγόριθμο Bracewell, χρησιμοποιώντας παράλληλους υπολογιστές **[25]**. Από την άλλη πλευρά, η χρήση

συμβολικών πινάκων και μεθόδων μπορεί να προτιμηθεί σε περιπτώσεις όπου δεν δίνονται όλοι οι συντελεστές του πολυωνυμικού πίνακα ή δεν είναι όλοι αριθμοί.

ΚΕΦΑΛΑΙΟ 4

ΥΛΟΠΟΙΗΣΗ ΠΑΡΑΛΛΗΛΟΥ ΑΛΓΟΡΙΘΜΟΥ ΕΥΡΕΣΗΣ ΟΡΙΖΟΥΣΑΣ ΠΟΛΥΩΝΥΜΙΚΟΥ ΠΙΝΑΚΑ

4.1 Εισαγωγή

Για αρκετά χρόνια η επιστήμη των υπολογιστών χρησιμοποιούσε για την επίλυση υπολογιστικών προβλημάτων τον σειριακό υπολογισμό, κατά τον οποίο ένας επεξεργαστής ήταν υπεύθυνος να εκτελέσει ακολουθιακά τις εντολές κάποιου προγράμματος. Ωστόσο, βαθμιαία άρχισε να ωριμάζει και να χρησιμοποιείται ο παράλληλος υπολογισμός για την επίλυση απαιτητικών και χρονοβόρων προβλημάτων.

Ο παράλληλος υπολογισμός αφορά στην συνεταιριστική και ταυτόχρονη επεξεργασία δεδομένων από περισσότερους από ενός επεξεργαστές ή υπολογιστές, αποσκοπώντας στην γρήγορη επίλυση σύνθετων υπολογιστικών προβλημάτων. Οι επεξεργαστές, οι οποίοι συνήθως είναι του ίδιου τύπου, βρίσκονται σε κοντινή απόσταση, έχουν τεράστια υπολογιστική δύναμη και είναι υπεύθυνοι να εκτελούν κάποιους παράλληλους υπολογισμούς **[26].** Στην περίπτωση των παράλληλων υπολογιστών υπάρχει ένα δίκτυο μεγάλης ταχύτητας, που τους συνδέει και τους επιτρέπει να ανταλλάσουν δεδομένα.

Η σημαντικότητα του παράλληλου υπολογισμού έγκειται στο ότι μπορούμε να διεκπεραιώσουμε πολύπλοκους και μεγάλους υπολογισμούς, τους οποίους ένας σειριακός υπολογιστής δεν θα μπορούσε να εκτελέσει ή να μειώσουμε δραματικά το υπολογιστικό και ενδεχόμενα το οικονομικό κόστος.

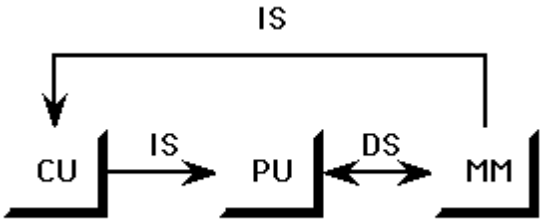
4.2 Παράλληλες αρχιτεκτονικές και Μοντέλα

Η ύπαρξη περισσοτέρων από ενός επεξεργαστή σε ένα σύστημα, συνήθως δυσκολεύει την μοντελοποίηση και το σχεδιασμό αλγορίθμων και πολλές φορές αυξάνει δυσανάλογα το μέγεθος του κώδικα. Σε αντίθεση με τα ακολουθιακά προγράμματα, οι παράλληλοι αλγόριθμοι εξαρτώνται ισχυρά από την αρχιτεκτονική των υπολογιστών για τους οποίους έχουν σχεδιαστεί. Ο προγραμματιστής πριν

ξεκινήσει τη σχεδίαση και υλοποίηση ενός παράλληλου αλγορίθμου, θα πρέπει να γνωρίζει καλά την αρχιτεκτονική και τις δυνατότητες του συστήματος. Παρακάτω φαίνονται οι κυριότερες αρχιτεκτονικές - μοντέλα υπολογισμού που παρουσιάζουν συνήθως ενδιαφέρον **[47].**

- SISD: Single Instruction stream (IS), Single Data stream (DS) - σειριακό

Στο μοντέλο SISD υπάρχει μόνο ένας επεξεργαστής (PU) και εκτελεί μόνο σειριακούς υπολογισμούς. Όπως φαίνεται και στο σχήμα 5, υπάρχει μια μονάδα έλεγχου, ένας επεξεργαστής και μια τοπική μνήμη (MM).



Σχήμα 5: *Μοντέλο υπολογισμού SISD*

- MISD: Multiple Instruction stream, Single Data stream

Σε αυτό το μοντέλο έχουμε N επεξεργαστές. Κάθε ένας έχει την δική του μονάδα έλεγχου (CU) και μοιράζονται μια κοινόχρηστη μνήμη, οπού βρίσκονται τα δεδομένα. Σε κάθε βήμα οι επεξεργαστές μπορούν να εκτελέσουν διαφορετική εντολή στο ίδιο στοιχείο

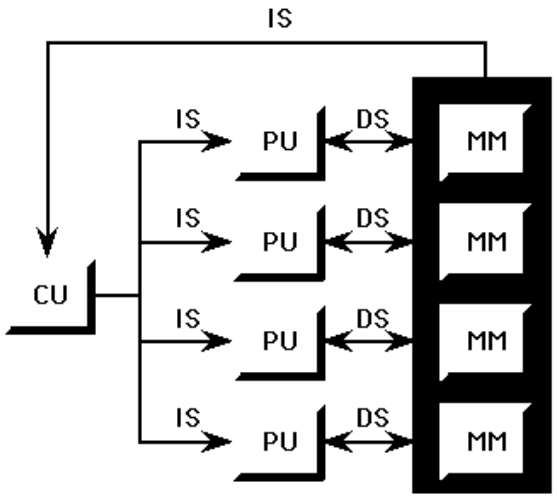
- SIMD: Single Instruction stream, Multiple Data stream

Εδώ έχουμε N επεξεργαστές. Ο κάθε ένας έχει την δική του τοπική μνήμη και διαφορετικό ρεύμα δεδομένων (σχήμα 6).

- MIMD: Multiple Instruction stream, Multiple Data stream

Υπάρχουν N επεξεργαστές, N ρεύματα εντολών και N ρεύματα δεδομένων. Οι επεξεργαστές μπορούν να εκτελέσουν διαφορετικά προγράμματα, πάνω σε διαφορετικά δεδομένα, για να επιλύουν διαφορετικά υποπροβλήματα. Οι επεξεργαστές δουλεύουν συγχρονισμένα και σε κάθε βήμα (clock - cycle). όλοι οι επεξεργαστές εκτελούν την ίδια εντολή πάνω σε διαφορετικά δεδομένα. Υπάρχουν δυο τρόποι επικοινωνίας:

- Μέσω κοινόχρηστης μνήμης (PRAM) ή
- Μέσω Δικτύου Αλληλοσυνδεσης



Σχήμα 6: *Μοντέλο υπολογισμού SIMD*

4.3 Αξιολόγηση παράλληλων αλγορίθμων

Για να μπορέσουμε να αναλύσουμε θεωρητικά τους παράλληλους αλγόριθμους και τέλος να τους αξιολογήσουμε ως προς την αποδοτικότητα τους είναι αναγκαίο να εισάγουμε τις παρακάτω έννοιες **[26]**:

- Πλήθος Επεξεργαστών $P(n)$: Ο αριθμός των επεξεργαστών που χρησιμοποιήθηκαν για να επιλύσουν ένα πρόβλημα μεγέθους n.
- Παράλληλος Χρόνος Εκτέλεσης $Tp(n)$: Είναι ο χρόνος που χρειάζονται οι p επεξεργαστές, δουλεύοντας παράλληλα, ώστε να επιλύσουν ένα πρόβλημα μεγέθους n.
- Κόστος $C(n)$: Είναι το γινόμενο του παράλληλου χρόνου εκτέλεσης επί τον αριθμό των επεξεργαστών που χρησιμοποιήθηκαν για να επιλυθεί παράλληλα, ένα πρόβλημα μεγέθους n. $C(n) = Tp(n) \times P(n)$.
- Χρόνος καλύτερου σειριακού αλγορίθμου $T^*(n)$: Είναι ο χρόνος που χρειάζεται ένας επεξεργαστής για να επιλύσει ένα πρόβλημα εκτελώντας τον καλύτερο σειριακό αλγόριθμο.
- Επιτάχυνση $Sp(n)$: λόγος του χρόνου εκτέλεσης του καλύτερου σειριακού αλγορίθμου που επιλύει ένα πρόβλημα Α, δια το χρόνο εκτέλεσης του παράλληλου αλγορίθμου που επιλύει το πρόβλημα Α μεγέθους n. $Sp(n) = T^*(n)/Tp(n)$.

Ο χρόνος και το κόστος του κάθε αλγορίθμου μπορούν να χρησιμοποιηθούν ως μέτρο σύγκρισης με άλλους. Έχοντας λοιπόν δύο παράλληλους αλγορίθμους,

έστω Α και Β, μπορούμε να πούμε ότι ο Α είναι πιο αποδοτικός από άποψη χρόνου από τον Β αν $T_A(n) = O(T_B(n))$ ενώ είναι πιο αποδοτικός σε θέμα κόστους από τον Β αν ισχύει ότι $C_A(n) = O(C_B(n))$. Το ποια προσέγγιση είναι καλύτερη εξαρτάται από την εκάστοτε εφαρμογή. Αν η ταχύτητα του αλγορίθμου δεν είναι το πρωτεύον ζήτημα, τότε το κόστος είναι προτιμότερο.

Τέλος, προκειμένου να κρίνουμε αν ένας αλγόριθμος είναι βέλτιστος, πρέπει να συγκρίνουμε το κόστος του με τον χρόνο του καλύτερου σειριακού αλγορίθμου που επιλύει το ίδιο πρόβλημα. Ένας παράλληλος αλγόριθμος θεωρείται βέλτιστος αν το κόστος εκτέλεσης του είναι μικρότερο από τον χρόνο εκτέλεσης του ταχύτερου σειριακού $C(n) = O(T^*(n))$.

4.4 Υλοποίηση παράλληλου αλγορίθμου FFT μέσω FHT

4.4.1 Εισαγωγή

Στο κεφάλαιο 3 αναφερθήκαμε στο γεγονός οτι ένας παράλληλος αλγόριθμος μπορεί να αποφέρει πολλαπλά οφέλη στον υπολογισμό της παρεμβολής ενός πολυωνυμικού πίνακα. Αφενός, μπορεί να μειώσει το χρόνο υπολογισμού του μετασχηματισμού, αφετέρου μπορεί να διαμοιράσει τους υπολογισμούς των σημείων παρεμβολής. Για την υλοποίηση των παράλληλων αλγορίθμων χρησιμοποιήθηκε το Matlab Parallel Toolbox. Συνοπτικές οδηγίες για τη χρήση του και μια σύντομη περιγραφή του παρουσιάζονται στο παράρτημα "Β".

Το Matlab Parallel Toolbox είναι ουσιαστικά ένα μοντέλο SIMD το οποίο επιτρέπει την παράλληλη επεξεργασία είτε τοπικά, αξιοποιώντας τους πυρήνες ενός πολυπύρηνου επεξεργαστή σε έναν υπολογιστή, είτε μεμακρυσμένα μέσω δικτύου σε πολλούς υπολογιστές. Επιλέξαμε να χρησιμοποιήσουμε το τοπικό μοντέλο διασύνδεσης για λόγους απλότητας και λόγω ελλείψεως ενός κατάλληλου cluster υπολογιστών. Οι υπολογισμοί πραγματοποιήθηκαν σε υπολογιστή Intel Core2Duo 3GHz (διπύρηνου επεξεργαστή) με 8 GB RAM και λειτουργικό Linux.

4.4.2 Υλοποίηση του παράλληλου 1-D FFT μέσω FHT

Για την υλοποίηση του παράλληλου αλγορίθμου FFT, μοιραστήκαν οι υπολογισμοί του μετασχηματισμού Hartley στους δυο πυρήνες του υπολογιστή. Ο κάθε πυρήνας εκτέλεσε ένα μετασχηματισμό Hartley είτε στο πραγματικό, είτε

στο φανταστικό μέρος του FFT. Υπολογίστηκαν παράλληλα οι προσθαφαιρέσεις των στοιχείων $H(\kappa)+H(N-\kappa)$ και $H(\kappa)-H(N-\kappa)$ για το πραγματικό και φανταστικό μέρος και στο τέλος συνδυάστηκαν για να μας δώσουν το μετασχηματισμό. Ο κώδικας που υλοποίησε τον παραπάνω αλγόριθμο φαίνεται στο παράρτημα "A" (p_cfht2ffft.m, p_fft1d.m, p_ifft1d.m). Ο αντίστροφος 1-D μετασχηματισμός υπολογίζεται από τον ίδιο κώδικα, αλλάζοντας απλά το πρόσημο του μετασχηματισμού.

4.4.3 Υλοποίηση του παράλληλου 2-D FFT μέσω FHT

Για την υλοποίηση του παράλληλου 2-D FFT εφαρμόστηκε ο αλγόριθμος γραμμή-στήλη ως εξής. Εκτελέστηκε από κάθε πυρήνα ο σειριακός FFT στο μισό πλήθος των γραμμών του πίνακα και έπειτα στο μισό πλήθος των στηλών του. Για να γίνει αυτό, ο αρχικός πίνακας διαμερίστηκε σε 4 ισομεγέθη block, όπως φαίνεται παρακάτω:

$$P=\begin{pmatrix} x+2y & 3y & 0 & 4xy \\ 3y^2+1 & 9 & 2x^2+1 & 2y+5 \\ 4 & xy^3+2 & 2 & 3y \\ 7x & -1 & 5x & 2x+5 \end{pmatrix}=\begin{pmatrix} \begin{pmatrix} x+2y & 3y \\ 3y^2+1 & 9 \end{pmatrix} & \begin{pmatrix} 0 & 4xy \\ 2x^2+1 & 2y+5 \end{pmatrix} \\ \begin{pmatrix} 4 & xy^3+2 \\ 7x & -1 \end{pmatrix} & \begin{pmatrix} 2 & 3y \\ 5x & 2x+5 \end{pmatrix} \end{pmatrix} \quad \text{ή}$$

$$P=\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

Εφαρμόστηκε ο FFT πρώτα στους πίνακες A, B από τον πρώτο πυρήνα και στους C, D από το δεύτερο. Έπειτα, αφού αντάλλαξαν τους νέους υπολογισμένους πίνακες B και C μεταξύ τους, οι πυρήνες εφάρμοσαν τον FFT παράλληλα στους A, C και B, D (εφαρμογή κατά στήλες). Το κέρδος είναι διπλό. Αφενός παραλληλοποιείται η διαδικασία με ίσο αντιστάθμισμα φορτίου, κερδίζοντας χρόνο και αφετέρου εξοικονομούμε μνήμη αφού ο κάθε πυρήνας έχει στη διάθεση του μόνο το μισό πίνακα κάθε φορά. Το πρόγραμμα που εφαρμόζει τον αλγόριθμο φαίνεται στο παράρτημα "A" (fft2d.m, ifft2d.m). Ο αντίστροφος 2-D μετασχηματισμός υπολογίζεται από τον ίδιο κώδικα, αλλάζοντας απλά το πρόσημο του μετασχηματισμού.

4.4.4 Αξιολόγηση αλγορίθμων

Στον παρακάτω πίνακα φαίνεται ο υπολογιστικός χρόνος των σειριακών και των παράλληλων αλγορίθμων που υλοποιήθηκαν για πολυωνυμικούς πίνακες διαφόρων διαστάσεων (υπολογιστικός χρόνος σε sec).

Πίνακας 5: *Αποτελέσματα απόδοσης FFT του Matlab και FFT μέσω FHT*

Dim	1D Parallel FFT	1D Hartley FFT	2D Parallel FFT	2D Hartley FFT
2	0,002731	0,003390	0,007925	0,004786
4	0,001503	0,001153	0,007706	0,008017
8	0,001547	0,001028	0,009415	0,017976
16	0,001974	0,001258	0,020022	0,035923
32	0,002990	0,001481	0,100897	0,071724
64	0,005147	0,001720	0,108130	0,160619
128	0,010403	0,002148	0,228714	0,398542
256	0,011265	0,002928	0,619298	1,154917
512	0,007297	0,005740	2,162293	3,957442
1024	0,007393	0,009020	8,219254	15,754228
2048	0,020145	0,019411	36,245447	64,835985
4096	0,047162	0,043076	154,281063	264,890956

Παρατηρούμε εύκολα ότι δεν υπάρχει υπολογιστικό κέρδος από την εφαρμογή του παράλληλου 1D-FFT. Αυτό συμβαίνει γιατί το κόστος επικοινωνίας και ανταλλαγής δεδομένων είναι υψηλό σε σχέση με το κόστος υπολογισμού του αλγορίθμου και υπερκαλύπτει το όποιο υπολογιστικό κέρδος. Αντίθετα στον 2-D FFT ο παράλληλος κώδικας κάνει το μισό χρόνο του σειριακού αλγορίθμου. Αυτό συνέβη γιατί το εντατικό υπολογιστικά κομμάτι παραλληλοποιείται πλήρως και ελαχιστοποιείται το κόστος ανταλλαγής δεδομένων. Άρα λοιπόν η παραλληλοποίηση του κώδικα μπορεί να επιφέρει σημαντικά υπολογιστικά οφέλη για μετασχηματισμούς μεγαλύτερων διαστάσεων.

4.5 Υλοποίηση παράλληλου αλγορίθμου παρεμβολής για τον υπολογισμό της ορίζουσας πολυωνυμικού πίνακα.

4.5.1 Περιγραφή αλγορίθμου

Όπως είδαμε κεφάλαιο 3, το εντατικότερο υπολογιστικά μέρος του αλγορίθμου παρεμβολής, είναι η εκτίμηση της ορίζουσας του πίνακα στα σημεία Fourier. Για να μειωθεί το κόστος του αλγορίθμου, δημιουργήθηκε ένας πίνακας ο

οποίος περιείχε τα υπολογισμένα σημεία παρεμβολής και διαχωρίστηκε στη μέση έτσι ώστε ο κάθε πυρήνας του επεξεργαστή να έχει το ήμισυ των στηλών του. Έπειτα ο κάθε πυρήνας εκτιμά παράλληλα την ορίζουσα στα σημεία παρεμβολής που έχει στη διάθεσή του. Τέλος, οι δυο πίνακες που προκύπτουν συνδυάζονται στον τελικό πίνακα, πάνω στον οποίο εφαρμόζεται ο παράλληλος αλγόριθμος IFFT μέσω FHT της παραγράφου 4.4.3. Ο κώδικας που υλοποιεί τον παράλληλο αλγόριθμο παρεμβολής εμφανίζεται στο παράρτημα "A" (p_interpolated_ determinant.m).

Στον παρακάτω πίνακα φαίνονται οι χρόνοι υπολογισμού (υπολογιστικός χρόνος σε sec) με χρήση σειριακής και παράλληλης παρεμβολής με FHT για τυχαίους πολυωνυμικούς πίνακες, 2 διαστάσεων, μεγέθους 8x8 και διαφόρων πολυωνυμικών βαθμών.

Πίνακας 6: *Αποτελέσματα απόδοσης παράλληλης πολυωνυμικής παρεμβολής*

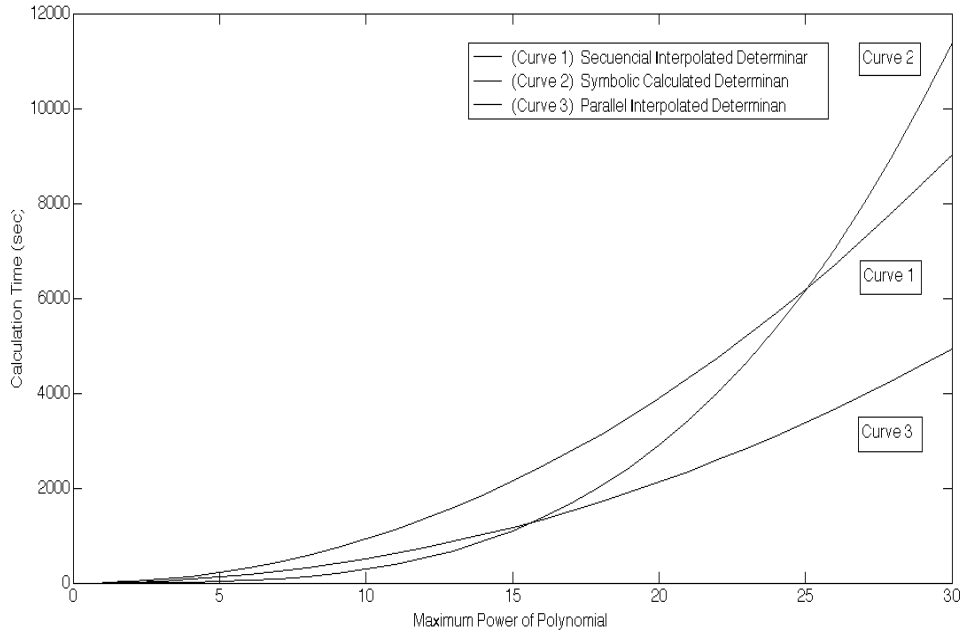
Max Polynomial Power	Parallel FHT Interpolated Determinant	Sequencial FHT Interpolated Determinant	Symbolic Calculated Determinant (Matlab)
1	2,116959	3,530134	0,547466
2	17,061854	30,531318	2,160438
3	38,295589	76,409196	5,862685
4	75,936228	134,926233	13,626698
5	114,996397	230,071147	30,142386
6	177,537591	293,271465	52,712246
7	245,361021	463,024805	85,538860
8	309,433030	570,499409	123,510596
9	409,111245	745,836339	209,553168
10	505,584238	924,649908	278,800820

4.5.2 Συμπεράσματα

Από τον παραπάνω πίνακα μπορούμε να παρατηρήσουμε οτι ο υπολογιστικός χρόνος μειώθηκε στο μισό με τη χρήση παράλληλου κώδικα. Αυτό οφείλεται μεν στην παραλληλοποίηση του 2D IFFT, αλλά περισσότερο στην παραλληλοποίηση των υπολογισμών εκτίμησης της ορίζουσας στα σημεία παρεμβολής.

Για να καταλήξουμε σε πιο ασφαλή συμπεράσματα, σε ότι αφορά την αποδοτικότητα του παράλληλου αλγορίθμου υπολογισμού της ορίζουσας ενός

πολυωνυμικού πίνακα με τη χρήση της παρεμβολής, εργαστήκαμε ως εξής. Αρχικά έγινε προσαρμογή καμπύλης στα δεδομένα του πίνακα 6 και έπειτα με extrapolation υπολογίστηκαν εκτιμήσεις για τους υπολογιστικούς χρόνους για μεγαλύτερους πολυωνυμικούς βαθμούς. Για την προσαρμογή της καμπύλης χρησιμοποιήθηκε το μοντέλο της εκθετικής συνάρτησης. Από τις παραπάνω εκτιμήσεις αποδεικνύεται ότι ο παράλληλος αλγόριθμος παρεμβολής ξεπερνά την απόδοση της ενσωματωμένης συμβολικής συνάρτησης του MatLab για πολυωνυμικό βαθμό 16 και πάνω, ενώ ο σειριακός για 25. Επίσης για πολυωνυμικό βαθμό 30, η παράλληλη παρεμβολή χρειάζεται λιγότερο από το μισό χρόνο υπολογισμού από την ενσωματωμένη του Matlab. Τα παραπάνω συμπεράσματα φαίνονται στο σχήμα 7.



Σχήμα 7: *Προσαρμογή Καμπύλης στα πειραματικά δεδομένα του πίνακα 6.*

Για να επαληθευτούν τα αποτελέσματα της προσαρμογής καμπύλης έγινε μία μέτρηση του υπολογιστικού χρόνου και για τους τρεις αλγορίθμους, για τυχαίο πίνακα πολυωνυμικού βαθμού 20. Οι εκτιμήσεις βρέθηκαν πολύ κοντά στα πειραματικά δεδομένα, εκτός του υπολογιστικού χρόνου του συμβολικού αλγορίθμου, ο οποίος ήταν αρκετά αυξημένος σε σχέση με την εκτίμηση. Τα αποτελέσματα φαίνονται στον πίνακα 7 (υπολογιστικός χρόνος σε sec).

Εδώ θα πρέπει να σημειωθεί, ότι ο υποδιπλασιασμός του υπολογιστικού χρόνου έγινε όχι χρησιμοποιώντας επιπρόσθετο hardware ή υπολογιστική ισχύ, αλλα αξιοποιώντας τους ήδη υπάρχοντες υπολογιστικούς πόρους με αποδοτικό τρόπο.

Επιπλέον επειδή η παρεμβολή είναι ένα γενικότερο εργαλείο υπολογισμού, ο παράλληλος αλγόριθμος παρεμβολής μπορεί να επιφέρει τα ίδια θεαματικά αποτελέσματα μείωσης του χρόνου υπολογισμού και κατά την εφαρμογή του για την εύρεση του αντιστρόφου και όχι μονον, ενός πολυμεταβλητού πολυωνυμικού πίνακα.

Πίνακας 7: *Υπολογιστικοί χρόνοι για τυχαίο πίνακα πολυωνυμικού βαθμού 20.*

Max Polynomial Power	Estimated Sequential Interpolated Determinant	Calculated Symbolic Determinant	Estimated Parallel Interpolated Determinant
20	3882.4	2906.0	2126.7

Max Polynomial Power	Calculated Sequential Interpolated Determinant	Calculated Symbolic Determinant	Calculated Parallel Interpolated Determinant
20	3728.3	3500.8	2141.3

4.6 Επίλογος - Μελλοντική εργασία

Στην παρούσα εργασία παρουσιάστηκε ένας αλγόριθμος υπολογισμού της ορίζουσας ενός πολυωνυμικού πίνακα δυο μεταβλητών με τη χρήση παρεμβολής και μετασχηματισμού Hartley. Η μέθοδος της παρεμβολής προτιμάται γενικότερα από τις συμβολικές μεθόδους, ως ένας απλός αλγόριθμος αριθμητικής ανάλυσης, για μεγάλους και πολύπλοκους πολυωνυμικούς πίνακες πολλών μεταβλητών.

Ο μετασχηματισμός Hartley χρησιμοποιήθηκε ως ενδιάμεσος του υπολογισμού του FFT, γιατί παρέχει ένα γρήγορο και αποδοτικό τρόπο υπολογισμού του. Τα υπολογιστικά του οφέλη είναι αφενώς μεν ότι διαιρεί τον FFT σε ανεξάρτητα μέρη, τα οποία μπορούν να υπολογιστούν παράλληλα, αφετέρου η υλοποίηση του FHT σε παράλληλους υπολογιστές μπορεί να αποφέρει μέχρι και οχταπλασιασμό της ταχύτητας υπολογισμού, για τον κανονικοποιημένο FHT. Αυτό είναι δυνατό με τη χρήση cluster 8 υπολογιστών ή ενός οχταπύρηνου

επεξεργαστή, όπου κάθε πυρήνας υπολογίζει μία εξοδο από μία είσοδο, σύμφωνα με το σχήμα 4.

Οι υλοποιήσεις των αλγορίθμων που χρησιμοποιήθηκαν στην παρούσα εργασία εμφανίζουν μερικές αδυναμίες και συγκεκριμένα:

- Υλοποιήθηκαν σε MatLab.

Η χρήση μιας γλώσσας χαμηλότερου επιπέδου (όπως η C ή assembly) θα οδηγούσε σε μικρότερα και γρηγορότερα προγράμματα υπολογισμού του FFT.

- Ο αλγόριθμος υλοποίησης του FHT δεν ήταν ο αποδοτικότερος.

Ο κανονικοποιημένος FHT έχει αποδειχθεί ως ο γρηγορότερος μέχρι σήμερα **[25]**.

- Υλοποιήθηκαν σε έναν υπολογιστή με τη χρήση MatLab Parallel Toolbox.

Εναλλακτικά θα μπορούσε να χρησιμοποιηθεί ένα cluster υπολογιστών. Η παραλληλοποίηση κώδικα με χρήση C και MPI (Message Passing Interface) έχει αποδειχθεί ιδανική για αυτό το σκοπό.

Για την υλοποίηση ενός τέτοιου cluster θα αρκούσαν 8 ανεξάρτητοι κόμβοι (υπολογιστές ή πυρήνες επεξεργαστή) σε συνδεσμολογία υπερκύβου διάστασης d=3. Κάθε κόμβος, σε αυτή την περίπτωση αναλαμβάνει:

ο να υπολογίσει το 1/8 του κανονικοποιημένου FHT, σύμφωνα με το σχήμα 4.

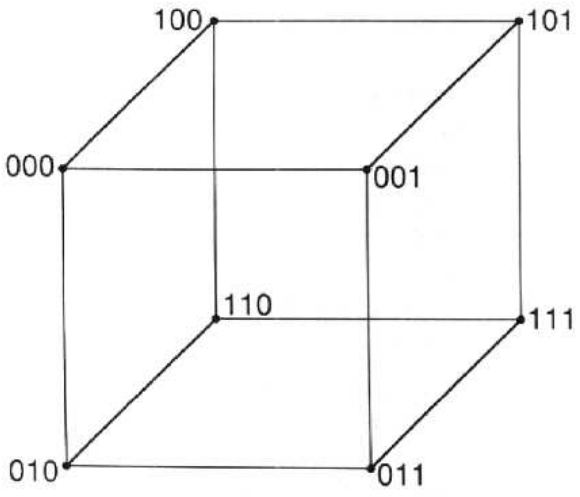
ο να εκτιμήσει την ορίζουσα στο 1/8 των σημείων παρεμβολής, διαχωρίζοντας τον πίνακα των σημείων Fourier με τρόπο ανάλογο του παράλληλου αλγορίθμου παρεμβολής του κεφαλαίου 4.

ο να εκτελέσει το 1/8 των υπολογισμών των 1D-FFT και 2D-FFT μέσω FHT, διαχωρίζοντας το υπολογιστικό φορτίο σύμφωνα με τους παράλληλους αλγορίθμους που περιγράφθηκαν στο κεφάλαιο 4.

Παρακάτω και για λόγους πληρότητας θα αναφερθούμε συνοπτικά στη συνδεσμολογία του υπερκύβου **[49]**. Στην τοπολογία διασύνδεσης υπερκύβου, ο αριθμός των επεξεργαστών είναι πάντα ακριβώς δύναμη του 2. Αν ο αριθμός των επεξεργαστών είναι 2^d, τότε το d ονομάζεται διάσταση του υπερκύβου. Κάθε επεξεργαστής θα έχει έναν αριθμό του οποίου η αναπαράσταση στο δυαδικό σύστημα έχει d bits. Ο ορισμός του σχήματος διασύνδεσης του υπερκύβου είναι: κάθε επεξεργαστής με δυαδικό αριθμό i έχει απευθείας συνδέσεις με όλους τους επεξεργαστές που έχουν δυαδικό αριθμό j, έτσι ώστε το j να διαφέρει από το i

ακριβώς κατά ένα δυαδικό ψηφίο (κώδικας Gray). Για παράδειγμα, ένας υπερκύβος με d=3 θα έχει οκτώ επεξεργαστές που είναι αριθμημένοι ως εξής: 000, 001, 010, 011, 100, 101, 110, 111. Ο επεξεργαστής 000 θα έχει απευθείας συνδέσεις με τους ακόλουθους επεξεργαστές: 001, 010, 100. Ο επεξεργαστής 011 θα έχει απευθείας συνδέσεις με τους επεξεργαστές: 111, 001, 010 κ.ο.κ.

Το σχήμα 8 παρουσιάζει έναν υπερκύβο με διάσταση d=3. Για αυτή την μικρή διάσταση, ο υπερκύβος μοίαζει ίδιος με την τοπολογία τρισδιάστατου πλέγματος. Όμως για μεγαλύτερες διαστάσεις η δομή του υπερκύβου είναι διαφορετική. Σε ένα πλέγμα κάθε επεξεργαστής έχει καθορισμένο πλήθος συνδέσεων, που είναι ανεξάρτητες από το μέγεθος του πλέγματος. Στον υπερκύβο το πλήθος των συνδέσεων για κάθε επεξεργαστή αυξάνεται, ανάλογα με την αύξηση του αριθμού των επεξεργαστών και είναι πάντα ίσος με τη διάσταση του υπερκύβου.



Σχήμα 8: *Ένας υπερκύβος διάστασης d=3.*

- Ο αλγόριθμος παρεμβολής χρησιμοποιεί ένα άνω φράγμα για την επιλογή του πλήθους των σημείων παρεμβολής που εξαρτάται από το μέγεθος του πίνακα. Οι Henrion Sebek απέδειξαν στο **[20]** οτι η παρεμβολή μπορεί είναι ευσταθής και για μικρότερο πλήθος σημείων παρεμβολής.

- Η παρεμβολή χρησιμοποιήθηκε για τον υπολογισμό μόνο της ορίζουσας πολυωνυμικού πίνακα.

Η παρεμβολή ως γενικότερο υπολογιστικό εργαλείο, μπορεί να χρησιμοποιηθεί, στην παράλληλη μορφή της, για τον υπολογισμό αντιστρόφων πολυμεταβλητών πολυωνυμικών πινάκων, αυθαίρετα μεγάλου μεγέθους και πολυωνυμικών βαθμών, χρησιμοποιώντας τα ίδια βήματα. Ο τρόπος με τον οποίο διαχωρίζονται οι υπολογισμοί των σημείων παρεμβολής μας δίνει την δυνατότητα να έχουμε υποδιπλασιασμό του χρόνου παρεμβολής με 2 παράλληλους πυρήνες, υποτετραπλασιασμό με 4, κ.ο.κ.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Antoniou, G., (2001). Transfer function computation for generalized n-dimensional systems. *Journal of the Franklin Institute* 338, 83–90.

[2] Antsaklis, P. J., Zhiqiang Gao, (1993). Polynomial and rational matrix interpolation: Theory and control applications, *International Journal of Control.* Vol.58, No.2. pp. 349-404.

[3] Aykanat, C., Dervis, A. (1995). Efficient Fast Hartley Transform Algorithms for Hypercube-Connected Multicomputers, *IEEE Transactions on Parallel and Distributed Systems, Vol. 6, 561-577*

[4] Bergland, G.D., (1968). A fast Fourier transform algorithm for real-valued series, Commum. ACM, 1968, 11, (10), pp. 703–710

[5] Blackfor, L. S., Choi, J., Cleary, A., Dazevedo, A., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, J., Petitet. A., Stanley. K., Walker, D., and Whaley, R.C. (1997). ScaLAPACK User's Guide, *Society for Industrial and Applied Mathematics.*

[6] Bracewell, R.N., (1978). The Fourier transform and its applications, *McGraw-Hill*

[7] Bracewell, R.N., (1984). The fast Hartley transform. *Proc. IEEE* 72(8)

[8] Bracewell, R.N., (1986) The Hartley Transform*, Oxford University Press, New York*

[9] Bracewell, R. N., (1990). Assessing the Hartley transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(12):2174

[10] Bracewell, R. N., Buneman, O., Hao, H., Villasenor, J., (1986). Fast two-dimensional Hartley transform. *Proceedings of the IEEE*, 74:1282–1283

[11] Bracewell, R. N., Hao, H., (1987). A three-dimensional DFT algorithm using the fast Hartley transform. *Proceedings of the IEEE*, 75(2):264–266

[12] Brigham, E.O., (1988). The fast Fourier transform and its applications, *Prentice-Hall, Englewood Cliffs*

[13] Chapman, B., Bodin, F., Hill L., Merlin, J., Viland. J., and Wollenweber, F. (1999). FITS – A light-weight integrated programming environment, *Lecture Notes in Computer Science 1685, 125-134.*

[14] Cooley, J. W., Lewis, P. A. W., Welch, P. D., (1967). The Fast Fourier Transform Algorithm and its Applications. *IBM Corp., Research Paper* RC-1743

[15] Duhamel, P., (1986). Implementations of split-radix FFT algorithms for complex, real and realsymmetric data. *IEEE Trans. ASSP* 34(2), 285–295

[16] Duhamel, P., Vetterli, M., (1987). Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data. *IEEE Trans. ASSP* 35(6), 818–824

[17] Foster, I., (1995). Designing and building parallel programs: Concepts and tools for parallel software engineering, *Addison-Wesley.*

[18] Goldschlager, L., Lister, A., (2000), Εισαγωγή στη σύγχρονη επιστήμη των υπολογιστών, *Εκδόσεις Δίαυλος*

[19] Hartley, R., V., L., (1942). A more symmetrical Fourier analysis applied to transmission problems, *Proc. IRE,* 30, 144–150

[20] Henrion, D., Sebek, M., (1999). Improved Polynomial Matrix Determinant Computation, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications,* Vol. 46 pp 1307 – 1308

[21] Hromcik, M. and Sebek, M. (1999). Numerical and Symbolic Computation of Polynomial Matrix Determinant, *Proceedings of the 38th Conference on Decision & Control Phoenix, Arizona USA* WeM04 15:20 1887-1888

[22] Ifeachor, E. C., and Jervis, B. W., (1993). *Digital Signal Processing: A Practical Approach.* Addison-Wesley

[23] Ionescu, F., Jalba, A., Ionescu, M, (2000). Parallel Implementation of Fast Hartley Transform (FHT) in Multiprocessor Systems, *Euro-Par 2000, LNCS 1900, pp. 532-535 Springer-Verlag Berlin Heidelberg*

[24] Jones, K. J., (2006) Design and parallel computation of regularised fast Hartley transform, *IEE Proceedings Vision, Image & Signal* Processing Vol. 153 pp. 70-78

[25] Jones, K., (2010). The Regularized Fast Hartley Transform Optimal Formulation of Real-Data Fast Fourier Transform for Silicon-Based Implementation in Resource Constrained Environments, *Springer Science+Business Media*

[26] Joseph, Jaja,. (1992). An introduction to Parallel Algorithms, *Addison-Welsey*

[27] Kailath, T., (1980). Linear Systems, *Prentice Hall, New Jersey*

[28] Karampetakis, N.P., Evripidou, A., (2010). On the computation of the inverse of a two-variable, *Multidimensional Systems and Signal Processing*

[29] Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). Introduction to parallel computing design and analysis of parallel algorithms, *Benjaming-Cummings.*

[30] Lau, L., (1996). Implementation of scientific applications in a heterogeneous distributed network, *PhD Thesis, University of Queensland, Department of Mathematics.*

[31] Merlin, J., Baden, S., Fink, S., and Chapman, B., (1999). Multiple data parallelism with HPF and KeLP, *Journal of Future Generation Computer Systems 15, 393-405.*

[32] Nieplocha, J., Harisson R. J., and Littlefield R.J. (1994), Global arrays: A portable shared-memory programming model for distributed memory computers, *Proceedings of Supercomputing 94, 340-349.*

[33] Oppenheim, A.V., Schafer, R.W. (1989). Discrete-time signal Processing, *Prentice-Hall*

[34] Plastino, A., Ribeiro, C. C., and Rodriguez N., (1999). A tool for SPMD application development with support for load balancing, *Proceedings of the ParCo Parallel Computing Conference.*

[35] Poularikas, A.D., (2010). *Transforms and Applications Handbook 3rd Edition.* CRC Press Taylor & Francis Group

[36] Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., (1998). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge UP

[37] Sebek, M., Pejchova, S., Henrion, D., (1996). Numerical Methods for Zeros and Determinant of Polyonomial Matrix, *Proceedings of the 4th IEEE Mediteranean Symposium on New Directions in Control and Automation*, pp.488-491., Chania, Crete, Greece, June 1996

[38] Singhal, K., Vlach, J., (1973). Accuracy and Speed of Real and Complex Interpolation, *Computing* Vol.11, Number 2, pp. 147-158

[39] Sorensen, H.V., Jones, D.L., Burrus, C.S., Heideman, M.T., (1985). On Computing the Discrete Hartley Transform. *IEEE ASSP* 33, 1231–1238

[40] Theußl, T., Tobler, R. F., Groller, E., (2000) The Multi-Dimensional Hartley Transform as a Basis for Volume Rendering, *WSCG 2000 Conference Proceedings, February 2000*

[41] Thomadakis, Michael E., and Jyh-Charn Liu, (1996). An Efficient Steepest- Edge Simplex Algorithm for SIMD Computers, *Proc. Of the International conference on Super-Computing, ICS '96, pp. 286-293.*

[42] Uniyal, P.R., (1994). Transforming real-valued sequences: fast Fourier versus fast Hartley transform algorithms, *IEEE Trans. Signal Process., 42, (11), pp. 3249–3254*

[43] Vologiannidis, S., Karampetakis, N.P. (2002). Inverses of Multivariable Polynomial Matrices by Discrete Fourier Transforms, *Multidimensional Systems & Signal Processing,* Vol.15, Issue 4, pp. 341-361

[44] Wang, Z., Hunt, B. R., (1985). The Discrete W Transform, *Appl. Math. Comput.*, Vol. 16, pp. 19–48

[45] Yang, W. Y., Chang, T. G., Song, I. H., Cho, Y. S., Heo, J., Jeon, W. G., Lee, J. W., Kim, J. K., (2009). Signals and Systems with MATLAB, *Springer-Verlag Berlin Heidelberg*

[46] Ευρυπίδου, Α., (2009). Πολυμεταβλητή πολυωνυμική παρεμβολή Μεταπτυχιακή διπλωματική εργασία *ΑΠΘ Μεταπτυχιακό Πρόγραμμα Σπουδών «Θεωρητική πληροφορική και θεωρία συστημάτων ελέγχου»*

[47] Νικολάου, Π., (2009). Προσομοίωση και πειραματική αξιολόγηση εύρωστων παράλληλων αλγορίθμων στο μοντέλο SB-PRAM, *Διπλωματική εργασία, Πανεπιστήμιο Κύπρου Τμήμα Πληροφορικής*

[48] Πλόσκας, Ν., (2009). Παράλληλος προγραμματισμός περιστροφικών αλγορίθμων εξωτερικών σημείων τύπου simplex, *Διπλωματική εργασία Μεταπτυχιακού Προγράμματος στην Εφαρμοσμένη Πληροφορική, Πανεπιστήμιο Μακεδονίας*

[49] Τεχνικές Παράλληλου Προγραμματισμού (http://www.it.uom.gr/project/parallel).

ΠΑΡΑΡΤΗΜΑ Α

ΚΩΔΙΚΕΣ ΑΛΓΟΡΙΘΜΩΝ ΣΕ MATLAB

```
%************************************************************************
%
%  sine_tests determines which sine function is computed efficiently in
%  MatLab.
%
%************************************************************************

A= -100+200.*rand(100000,1); % Initialize variables
arg0=pi/4;
arg1=sqrt(2);

t1=tic;
cos(A);
toc(t1)

t2=tic;
sin(A);
toc(t2)

t3=tic;
sin(A)+cos(A);
toc(t3)

t4=tic;
arg1*sin(A+arg0);
toc(t4)

t5=tic;
arg1*cos(A-arg0);
toc(t5)
```

```
function fOut=dht(fIn)

%**************************************************************************
%
%  dht computes a "slow" 1-D Hartley Transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 0 to N-1.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, real vector fIn with rows, the data to be transformed.
%
%    Output, real vector fOut, the transformed data.
%
%**************************************************************************

[m,n]=size(fIn); % Check if transpose is needed
if n>m
    fIn=fIn'; N=n; flag=1;
else
    N=m; flag=0;
end;

fOut=zeros(N,1);
arg0=2*pi/N;
arg1=pi/4;
arg2=sqrt(2);
n=(0:N-1)';

for k=0:N-1 % Begin 1-D Hartley Transform
   fOut(k+1)=sum(fIn.*(sin(arg0*n*k+arg1))); % cas(x)=sqrt(2)*sin(x+pi/4)
end         % End 1-D Hartley Transform
```

```
if flag==1;
    fOut=fOut';
end;

fOut=arg2*fOut./sqrt(N);
end
```

```
function fOut=fht(fIn)

%**************************************************************************
%
%  fht computes a radix-2 1D Fast Hartley Transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, real vector fIn with rows, the data to be transformed.
%
%    Output, real vector fOut, the transformed data.
%
%  Remarks:
%
%    Length of fIn() Should be 2^p. If it is not, the code pads input
%    with zeros and removes them at end of transform.
%
%    Sequential FHT Algorithm
%    Input is bit-reversed in Code in fIn(1..N)
%    Output in Normal Order in f(1..N)
%
%**************************************************************************

[k,l]=size(fIn); % Check if transpose is needed
if l>k
    fIn=fIn'; n=l; flag=1;
else
    n=k; flag=0;
end;

[f,ldn]=log2(n);
Nold=0; % To be used later as flag
```

```
if (f~=0.5)  % Pad array with zeros so that length is power of 2
    fIn=[fIn;zeros(2^ldn-n,1)];
    Nold=n;  % Keep to cut off redundant values at end of transform
    n=2^ldn;
end;

fIn=bitrevorder(fIn); % Reverse bit order

for ldm=1:ldn
    m=2^ldm;
    mh=m/2;
    m4=m/4;
    arg=pi/mh;

    for r=0:m:n-m
        for j=1:m4-1 % Hartley shift
            k=mh-j;

            u=fIn(r+mh+j+1);
            v=fIn(r+mh+k+1);

            c=cos(j*arg);
            s=sin(j*arg);

            fIn(r+mh+j+1)=c*u+s*v;
            fIn(r+mh+k+1)=s*u-c*v;
        end

        for j=0:mh-1
            u=fIn(r+j+1);
            v=fIn(r+j+mh+1);

            fIn(r+j+1)=u+v;
            fIn(r+j+mh+1)=u-v;
        end
    end
end

% Truncate last values if original data length was not power of 2
if (Nold~=0)
    fIn=fIn(1:Nold);
    n=Nold;
end

if flag==1; % Transpose if needed
    fIn=fIn';
end;

fOut=fIn/sqrt(n); % Make DHT it's own inverse.
end
```

```
function fOut = cfht2fft(fIn,fft_s)

%**************************************************************************
%
%  cfht2fft computes a complex 1D-FFT or 1D-IFFT through
%  1-D Hartley Transform. Forward or Inverse transform depends on fft_s.
%  Use -1 for forward and 1 for inverse transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex vector fIn with rows, the data to be transformed.
%
%    Input, Integer fft_s, the Fourrier transform sign
%
%    Output, complex vector fOut, the transformed data.
%
%**************************************************************************

[m,n]=size(fIn); % Check if transpose is needed
if n>m
    fIn=fIn'; flag=1; N=n;
else
    flag=0; N=m;
end;

% Compute FHT for real and imaginary parts
temp(:,1)=fht(real(fIn));
temp(:,3)=fht(imag(fIn));
temp(:,2)=[temp(1,1);flipud(temp(2:end,1))];
temp(:,4)=[temp(1,3);flipud(temp(2:end,3))];

realPart=temp(:,1)+temp(:,2)-fft_s*(temp(:,3)-temp(:,4));
```

```
imPart=temp(:,3)+temp(:,4)+fft_s*(temp(:,1)-temp(:,2));

% Comply with Matlab FFT function
fOut=sqrt(N)*0.5*(realPart+1i*imPart);

if flag==1; % Transpose
    fOut=fOut';
end;

end
```

```
function fOut=fft1d(fIn)

%**************************************************************************
%
%  fft1d computes a complex 1D-FFT through 1-D Hartley Transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex vector fIn with columns, the data to be transformed.
%
%    Output, complex vector fOut, the transformed data.
%
%**************************************************************************

fOut=cfht2fft(fIn,-1);

end
```

```
function fOut=ifft1d(fIn)

%*************************************************************************
%
%  ifft1d computes a complex 1D-IFFT through 1-D Hartley Transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex vector fIn with rows, the data to be transformed.
%
%    Output, complex vector fOut, the transformed data.
%
%*************************************************************************

N=length(fIn);
fOut=cfht2fft(fIn,1)/N;

end
```

```
function fOut=fft2d(fIn)

%**************************************************************************
%
%  fft2d computes a complex 2D-FFT through 1-D Hartley Transform using the
%  row-column method.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex array fIn, the data to be transformed.
%
%    Output, complex array fOut, the transformed data.
%
%**************************************************************************

[m,n]=size(fIn);

for i=1:m; % Trasform rows
       fIn(i,:)=cfht2fft(fIn(i,:),1);
end;

for i=1:n; %Transform columns
       fIn(:,i)=cfht2fft(fIn(:,i),-1);
end;

fOut=fIn(1:m,1:n);
end
```

```
function fOut=ifft2d(fIn)

%**************************************************************************
%
%  ifft2d computes a complex 2D-IFFT through 1-D Hartley Transform using
%  the row-column method.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex array fIn, the data to be transformed.
%
%    Output, complex array fOut, the transformed data.
%
%**************************************************************************
[m,n]=size(fIn);

for i=1:m; % Trasform rows
       fIn(i,:)=cfht2fft(fIn(i,:),-1);
end;

for i=1:n; %Transform columns
       fIn(:,i)=cfht2fft(fIn(:,i),1);
end;

fOut=fIn(1:m,1:n)/(m*n);
end
```

```
function fOut = p_cfht2fft(fIn,fft_s)

%**************************************************************************
%
%  p_cfht2fft computes a complex 1D-FFT or 1D-IFFT through parallel
%  1-D Hartley Transform. Forward or Inverse transform depends on fft_s.
%  Use -1 for forward and 1 for inverse transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex vector fIn with rows, the data to be transformed.
%
%    Input, Integer fft_s, the Fourrier transform sign
%
%    Output, complex vector fOut, the transformed data.
%
%**************************************************************************
if (labindex==1) % Lab1 is master node
    [m,n]=size(fIn); % Check if transpose is needed
    if n>m
        fIn=fIn'; flag=1; N=n;
    else
        flag=0; N=m;
    end;

    labSend(fIn,2,0); % Send input vector to Lab 2
    labSend(fft_s,2,1); % Send fourrier transform sign to Lab2

    re(:,1)=fht(real(fIn)); % Compute FHT for real part
    re(:,2)=[re(1,1);flipud(re(2:end,1))]; % fht(N-k)

    re(:,3)=re(:,1)+re(:,2); re(:,4)=re(:,1)-re(:,2);
```

```
    labSend(re(:,3:4),2,2); % Send real sumdiffs to Lab2
    im=labReceive(2,3); % Receive imaginary sumdiffs from Lab2

    realPart=re(:,3)-fft_s*im(:,2);

    imPart=labReceive(2,4); % Receive imaginary part from Lab2

    % Comply with Matlab FFT function
    fOut=sqrt(N)*0.5*(realPart+1i*imPart);

    if flag==1; % Transpose
        fOut=fOut';
    end;
end;

if (labindex==2) % Lab2 executes from now on
    fIn=labReceive(1,0);
    fft_s=labReceive(1,1); % Receive fourrier transform sign from Lab1

    im(:,1)=fht(imag(fIn)); % Compute FHT for imaginary part
    im(:,2)=[im(1,1);flipud(im(2:end,1))]; % fht(N-k)

    im(:,3)=im(:,1)+im(:,2); im(:,4)=im(:,1)-im(:,2);

    re=labReceive(1,2); % Receive real sumdiffs from Lab1
    labSend(im(:,3:4),1,3); % Send imaginary sumdiffs to Lab1

    imPart=im(:,1)+fft_s*re(:,2);

    labSend(imPart,1,4); % Send imaginary part to Lab1

end;
end
```

```
function fOut=p_fft2d(fIn)

%**************************************************************************
%
%  fft2d computes a complex parallel 2D-FFT through 1-D Hartley Transform
%  using the row-column method.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex array fIn, the data to be transformed.
%
%    Output, complex array fOut, the transformed data.
%
%**************************************************************************
if (labindex==1) % Lab1 is master node
    [m,n]=size(fIn);
    br_row=floor(m/2); br_col=floor(n/2); % Points where array seperates

    labSend(fIn(br_row+1:end,:),2,0); % Send lowest rows to Lab2

    for i=1:br_row; % Trasform upper rows
        fIn(i,:)=cfht2fft(fIn(i,:),1);
    end;

    labSend(fIn(1:br_row,br_col+1:end),2,1); % Send right upper block
    fIn(br_row+1:end,1:br_col)=labReceive(2,2); % Receive left down block

    for i=1:br_col; % Transform left columns
        fIn(:,i)=cfht2fft(fIn(:,i),-1);
    end;

    tmp=labReceive(2,3); % Receive right columns from Lab2
```

```
    fOut=[fIn(:,1:br_col) tmp];
end;

if (labindex==2) % Lab2 executes from now on
    fIn=labReceive(1,0); % Receive rows for transform

    [m,n]=size(fIn);
    br_col=floor(n/2);

    for i=1:m; % Trasform lower rows
        fIn(i,:)=cfht2fft(fIn(i,:),1);
    end;

    tmp=labReceive(1,1); % Receive right upper block
    labSend(fIn(:,1:br_col),1,2); % Send left down block

    fIn=[tmp;fIn(:,br_col+1:end)]; % Create columns for transform

    [~,n]=size(fIn); %Transform columns
    for i=1:n;
       fIn(:,i)=cfht2fft(fIn(:,i),-1);
    end;

    labSend(fIn,1,3); % Send right columns to Lab1
end;

end
```

```
function fOut=p_ifft2d(fIn)

%**************************************************************************
%
%  fft2d computes a complex parallel 2D-FFT through 1-D Hartley Transform
%  using the row-column method.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 1 to N.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%  Parameters:
%
%    Input, complex array fIn, the data to be transformed.
%
%    Output, complex array fOut, the transformed data.
%
%**************************************************************************
if (labindex==1) % Lab1 is master node
    [m,n]=size(fIn);
    br_row=floor(m/2); br_col=floor(n/2); % Points where array seperates

    labSend(fIn(br_row+1:end,:),2,0); % Send lowest rows to Lab2

    for i=1:br_row; % Trasform upper rows
        fIn(i,:)=cfht2fft(fIn(i,:),-1);
    end;

    labSend(fIn(1:br_row,br_col+1:end),2,1); % Send right upper block
    fIn(br_row+1:end,1:br_col)=labReceive(2,2); % Receive left down block

    for i=1:br_col; % Transform left columns
        fIn(:,i)=cfht2fft(fIn(:,i),1);
    end;

    tmp=labReceive(2,3); % Receive right columns from Lab2
```

```
    fOut=[fIn(:,1:br_col) tmp];
end;

if (labindex==2) % Lab2 executes from now on
    fIn=labReceive(1,0); % Receive rows for transform

    [m,n]=size(fIn);
    br_col=floor(n/2);

    for i=1:m; % Trasform lower rows
        fIn(i,:)=cfht2fft(fIn(i,:),-1);
    end;

    tmp=labReceive(1,1); % Receive right upper block
    labSend(fIn(:,1:br_col),1,2); % Send left down block

    fIn=[tmp;fIn(:,br_col+1:end)]; % Create columns for transform

    [~,n]=size(fIn); %Transform columns
    for i=1:n;
       fIn(:,i)=cfht2fft(fIn(:,i),1);
    end;

    labSend(fIn,1,3); % Send right columns to Lab1
end;

end
```

```
function pMatrixOut=create_bivariate_poly_matrix(maxPower)

%**************************************************************************
%
%  create_bivariate_poly_matrix computes a random 8x8 bivariate polynomial
%  matrix.
%
%  Discussion:
%
%    Firstly random coeffs are generated and transformed into random poly-
%    nomials by function create_bivariate_poly. The function is called 64
%    times to fill an 8x8 matrix with bivariate polynomials.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    MatLab Help
%
%  Parameters:
%
%    Input, integer maxPower, the maximum polynomial degree.
%
%    Output, sym bivariate polynomial array fOut.
%
%**************************************************************************

pMatrixOut=sym(zeros(8,8)); syms x y; % Initialize variables

for i=1:8
    for j=1:8 % Fill polynomial matrix
        pMatrixOut(i,j)=create_bivariate_poly(maxPower);
    end;
end;

end

function pOut=create_bivariate_poly(maxPower)

syms x y; % Initialize variables

u=randi(4,1,1); % How many terms will the monomial have

% Monomial coeffs and powers
p=[randi(4,u,1) randi(maxPower,u,1) randi(maxPower,u,1) randi(4,u,1)];

pOut=0;

for i=1:u % Add monomials to create the polynomial
    pOut=pOut+p(i,1)*x^p(i,2)*y^p(i,3)+p(i,4);
end;

end
```

```
function coeffs=interpolated_determinant(A,l,k)

%**************************************************************************
%
%  This is a test function to compute the determinant of 2-D polynomial
%  matrix using interpolation and Fast Hartley Transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 0 to N-1.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%**************************************************************************

syms x y % Initialize variables

[m n]=size(A); % Polynomial matrix dimensions

M1=l*m; M2=k*n;

R=(M1+1)*(M2+1); % Number of interpolation points

W1=exp(2*pi*1i/(M1+1));
W2=exp(2*pi*1i/(M2+1)); % General case where M1~=M2

r1=0:M1; r2=0:M2; % Calculate Interpolation points
u1=W1.^(-r1); u2=W2.^(-r2);

detA=zeros(M1+1,M2+1); % Genarate indices
i=1:M1+1; tmp=ones(1,M2+1); i=kron(i,tmp);
j=1:M2+1; tmp=ones(1,M1+1); j=kron(tmp,j);
tmp=[i;j];

for k=1:R % Evaluate determinant on Interpolation points
    i=tmp(1,k); j=tmp(2,k);
    x=u1(i); y=u2(j);
    detA(i,j)=det(eval(A));
```

```
end;

coeffs=ifft2d(detA); % Compute polynomial coeficients
end
```

```
function coeffs=p_interpolated_determinant(A,l,k)

%**************************************************************************
%
%  This is a test function to compute the determinant of 2-D polynomial
%  matrix using parallel interpolation and Fast Hartley Transform.
%
%  Discussion:
%
%    The discrete Hartley transform h of a set of data a is
%
%      h(i) = 1/sqrt(N) * sum (0<=j<=N-1) a(j) * cas(2*pi*i*j/N)
%
%    The data and coefficients are indexed from 0 to N-1.
%
%    With the above normalization factor of 1/sqrt(N), the Hartley
%    transform is its own inverse.
%
%    This routine is provided for illustration and testing.  It is
%    inefficient relative to optimized routines.
%
%  Licensing:
%
%    This code is distributed under the GNU GPLv3 license.
%    Copy of the GPLv3 License can be found in the following URL:
%    http://www.gnu.org/licenses/gpl-3.0.html
%
%  Modified:
%
%    10 Nov 2011
%
%  Author:
%
%    Dimitrios Politis
%
%  Reference:
%
%    Ralph Hartley,
%    A More Symmetrical Fourier Analysis Applied to Transmission Problems,
%    Proceedings of the Institute of Radio Engineers,
%    Volume 30, pages 144-150, 1942.
%
%**************************************************************************

if (labindex==1) % Lab1 is master node
    syms x y % Initialize variables

    [m n]=size(A); % Polynomial matrix dimensions

    M1=l*m; M2=k*n;

    R=(M1+1)*(M2+1); % Number of interpolation points

    W1=exp(2*pi*1i/(M1+1));
    W2=exp(2*pi*1i/(M2+1)); % General case where M1~=M2

    r1=0:M1; r2=0:M2; % Calculate Interpolation points
    u1=W1.^(-r1); u2=W2.^(-r2);

    detA=zeros(M1+1,M2+1); % Genarate indices
    i=1:M1+1; tmp=ones(1,M2+1); i=kron(i,tmp);
    j=1:M2+1; tmp=ones(1,M1+1); j=kron(tmp,j);
    tmp=[i;j];

    labSend(tmp,2,1);
    labSend(u1,2,2); labSend(u2,2,3);
    labSend(R,2,4);
```

```
    for k=1:floor(R/2) % Evaluate determinant on Interpolation points
        i=tmp(1,k); j=tmp(2,k);
        x=u1(i); y=u2(j);
        detA(i,j)=det(eval(A));
    end;

    temp=labReceive(2,5);
    detA=detA+temp;

    coeffs=ifft2d(detA); % Compute polynomial coeficients
end;

if (labindex==2) % Lab2 goes from now on
    syms x y % Initialize variables

    tmp=labReceive(1,1);
    u1=labReceive(1,2); u2=labReceive(1,3);
    R=labReceive(1,4);

    for k=floor(R/2)+1:R % Evaluate determinant on Interpolation points
        i=tmp(1,k); j=tmp(2,k);
        x=u1(i); y=u2(j);
        detA(i,j)=det(eval(A));
    end;

    labSend(detA,1,5);
end;

end
```

ΠΑΡΑΡΤΗΜΑ Β

ΣΗΜΕΙΩΣΕΙΣ ΧΡΗΣΗΣ MATLAB PARALLEL TOOLBOX

Εισαγωγή

Σκοπός του παραρτήματος αυτού είναι η παρουσίαση των κυριότερων δυνατοτήτων του Parallel Computing Toolbox, του λογισμικού Matlab R2011b. Το Parallel Computing Toolbox επιτρέπει στον χρήστη του περιβάλλοντος Matlab να μεταφέρει το φόρτο εργασίας, από μία σύνοδο του Matlab (τον πελάτη), σε άλλες συνόδους του Matlab, οι οποίες ονομάζονται εργάτες (workers) ή εργαστήρια (labs). Ο χρήστης έχει τη δυνατότητα να χρησιμοποιήσει μεγάλο πλήθος εργατών ή εργαστηρίων ώστε να επωφεληθεί από την κατανεμημένη ή την παράλληλη επεξεργασία, αντίστοιχα.

Ένα χαρακτηριστικό της εργαλειοθήκης Parallel Computing Toolbox είναι η δυνατότητα να τρέχει σε έναν τοπικό δρομολογητή, τόσους εργάτες ή εργαστήρια όσοι και οι πυρήνες του επεξεργαστή στον υπολογιστή πελάτη. Το χαρακτηριστικό αυτό δίνει τη δυνατότητα να εκτελούνται κατανεμημένες και παράλληλες εργασίες, χωρίς να απαιτείται μία απομακρυσμένη συστοιχία ή το λογισμικό Matlab Distributed Computing Server. Στην περίπτωση αυτή, όλες οι ενέργειες που απαιτούνται για τον πελάτη, τον δρομολογητή και τον υπολογισμό των διεργασιών εκτελούνται στον ίδιο υπολογιστή.

Διαδραστική Παράλληλη Λειτουργία (pmode)

Στην παράγραφο αυτή παρουσιάζεται η λειτουργία pmode, η οποία εργάζεται ως τοπικός δρομολογητής και τρέχει τόσα εργαστήρια στο τοπικό μηχάνημα μέσω του Matlab, όσοι οι πυρήνες του επεξεργαστή. Για την παράλληλη λειτουργία δεν απαιτείται εξωτερική συστοιχία ή δρομολογητής.

Για να ξεκινήσει η παράλληλη λειτουργία (pmode) πληκτρολογούμε την εξής εντολή:

```
>> pmode start local 2
```

Η εντολή αυτή ξεκινά δυο τοπικά εργαστήρια, δημιουργεί μία παράλληλη συστοιχία και ανοίγει το παράθυρο παράλληλων εντολών.

Εάν τεθεί μία μεταβλητή στη παράλληλη λειτουργία, τότε τίθεται σε όλα τα εργαστήρια.

```
P>> x = pi
```

Μία μεταβλητή, όμως με το ίδιο όνομα δεν είναι απαραίτητο να έχει την ίδια τιμή σε όλα τα εργαστήρια.

Η συνάρτηση labindex επιστρέφει την ταυτότητα κάθε εργαστηρίου, στο οποίο ανήκει η παράλληλη εργασία.

```
P>> x = labindex
```

Η συνάρτηση numlabs επιστρέφει το συνολικό αριθμό των εργαστηρίων που χρησιμοποιούνται στην παράλληλη εργασία.

```
P>> all = numlabs
```

Για να τερματιστεί η παράλληλη λειτουργία και να επιστρέψει ο χρήστης στο Matlab αρκεί να εκτελέσει την παρακάτω εντολή.

```
P>> pmode exit
```

Οι κυριότερες εντολές για τη δημιουργία του κώδικα της διεργασίας

Η διαβίβαση δεδομένων μεταξύ των εργαστηρίων μίας παράλληλης εργασίας πραγματοποιείται με τη χρήση των κατάλληλων συναρτήσεων. Οι κυριότερες από αυτές τις συναρτήσεις είναι οι εξής:

- numlabs

Επιστρέφει το συνολικό αριθμό των εργαστηρίων που λειτουργούν παράλληλα στην τρέχουσα εργασία.

- labindex:

Χρησιμοποιείται για τη διευκρίνηση του αριθμού που χαρακτηρίζει κάθε εργαστήριο.

- labSend(data, destination, tag)

Χρησιμοποιείται για την αποστολή δεδομένων σε προκαθορισμένο παραλήπτη – εργαστήριο. Η μεταβλητή data είναι τα δεδομένα που στέλνονται σε συγκεκριμένο εργαστήριο. Η μεταβλητή destination είναι ο αριθμός (labindex) του εργαστηρίου που θα λάβει τα δεδομένα. Η μεταβλητή tag είναι μια ετικέτα που μπορεί να διευκρινίσει ποιο δεδομένο αποστέλλεται.

- data = labReceive(source,tag)

Χρησιμοποιείται για τη λήψη δεδομένων από έναν προκαθορισμένο αποστολέα – εργαστήριο. Η μεταβλητή data είναι τα δεδομένα που λαμβάνονται από ένα συγκεκριμένο εργαστήριο. Η μεταβλητή source είναι ο αριθμός (labindex) του εργαστηρίου που στέλνει τα δεδομένα. Η μεταβλητή tag είναι ένα επίθεμα το οποίο μπορεί να διευκρινίσει ποιο δεδομένο παραλαμβάνεται.

- labBroadcast(senderlab, data)

Χρησιμοποιείται για την αποστολή δεδομένων από το κεντρικό εργαστήριο προς τα υπόλοιπα. Η μεταβλητή senderlab είναι ο αριθμός (labindex) του κεντρικού εργαστηρίου που στέλνει τα δεδομένα. Η μεταβλητή data είναι τα δεδομένα που στέλνονται από το κεντρικό εργαστήριο. Για να λάβει ένα εργαστήριο δεδομένα που έχουν σταλεί με τη χρήση της εντολή labBroadcast πρέπει να χρησιμοποιήσει την ίδια την εντολή labBroadcast (senderlab) και όχι την εντολή labReceive.

- is_data_availiable = labProbe(source, tag)

Η εντολή αυτή χρησιμεύει στον έλεγχο των εισερχόμενων μηνυμάτων χωρίς να προηγηθεί η λήψη τους. Η μεταβλητή is_data_availiable είναι ένας αριθμός (0 ή 1), ο οποίος δείχνει εάν κάποιο μήνυμα είναι έτοιμο για να παραληφθεί. Η μεταβλητή source είναι ο αριθμός (labindex) του εργαστηρίου που στέλνει τα δεδομένα. Η μεταβλητή tag είναι ένα επίθεμα το οποίο διευκρινίζει ποιο δεδομένο παραλαμβάνεται.

- labBarrier

Η εντολή αυτή αποτελεί την πιο απλή μορφή συλλογικής επικοινωνίας. Δεν σχετίζεται με την ανταλλαγή δεδομένων, αλλά αντίθετα παρέχει έναν μηχανισμό για το συγχρονισμό όλων των εργαστηρίων που την καλούν. Κάθε εργαστήριο σταματάει προσωρινά έως ότου όλες οι διεργασίες να καλέσουν τη συνάρτηση αυτή.