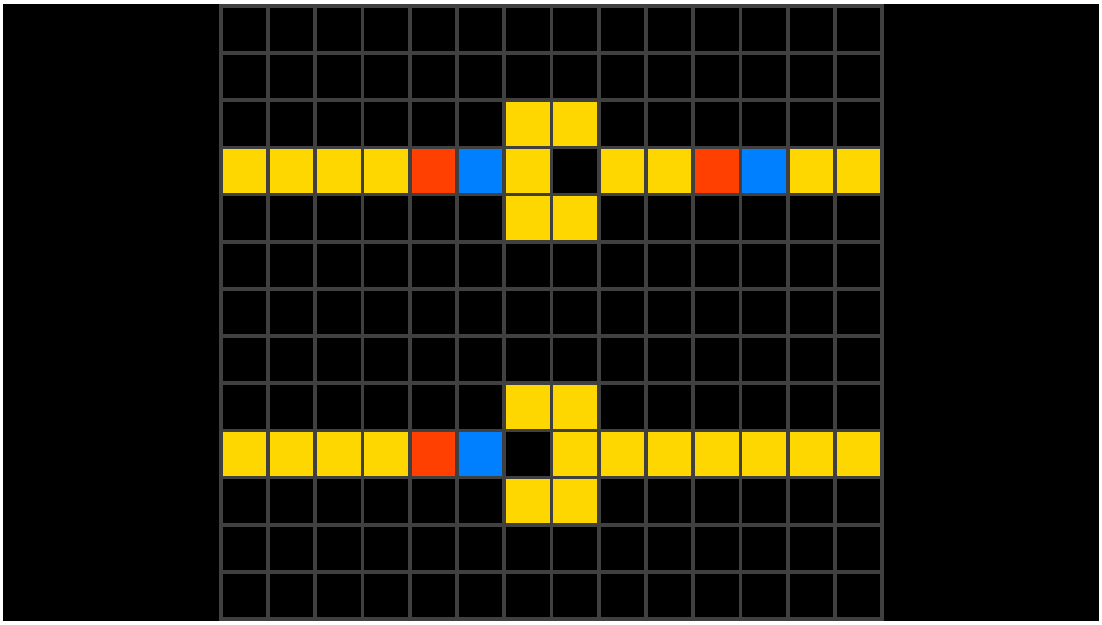
Ιούνιος 2017

ΑΤΟΜΙΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ: «Wireworld»: Υλοποίηση και πειραματισμοί.

Το **«Wireworld»** είναι ένα κυτταρικό αυτόματο (**cellular automaton**) ,Turing-complete, παρόμοιο του «Conway's Game of Life», το οποίο αναπτύχθηκε από τον **Brian Silverman** και παρουσιάστηκε το **1987** στο πρόγραμμα του «Phantom Fish Tank» κάτω από το όνομα της εταιρίας "Logo Computer Systems". Τον Ιανουάριο του 1990 το περιοδικό "Scientific American" έγραψε ένα άρθρο για το «Wireworld» με αποτέλεσμα να γίνει ευρύτερα γνωστό στην επιστημονική κοινότητα.

Πιο αναλυτικά το «Wireworld» είναι ένα σύνολο τετράγωνων κυττάρων τα οποία αλληλοεπιδρούν σε ένα πλέγμα σύμφωνα με αυστηρούς κανόνες δημιουργώντας ένα κυτταρικό αυτόματο. Το αυτόματο αυτό έχει διακριτά χρονικά διαστήματα που ονομάζονται γενιές (generations).



Διπλό κλικ για animation.

Παραπάνω βλέπουμε μια δίοδο στο Wireworld. Ο "clock generator" είναι εκτός σχήματος.
(πηγή: Wikipedia.org)

Ένα κύτταρο στο «Wireworld» μπορεί να έχει 4 καταστάσεις (αριθμημένες από το 0 έως το 3):

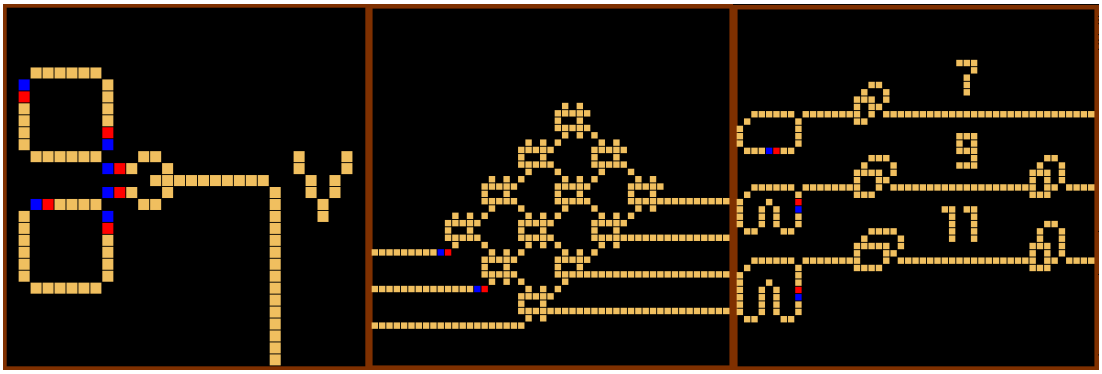
0. Κενό – Empty (Μαύρο)
1. Κεφάλη ηλεκτρονίου - Electron head (Μπλέ)
2. Ουρά ηλεκτρονίου - Electron tail (Κόκκινο)
3. Αγωγός – Conductor (Κίτρινο)

Υπάρχουν οι παρακάτω κανόνες που καθορίζουν ποια θα ειναι η κατάσταση ενός κυττάρου στην επόμενη γενιά του:

- Κενό, αν είναι κενό.
- Ουρά ηλεκτρονίου, αν είναι κεφαλή ηλεκτρονίου.
- Αγωγός, αν είναι ουρά ηλεκτρονίου.

- Κεφαλή ηλεκτρονίου, αν η γειτονιά Moore του ,δηλαδή τα κύτταρα πάνω, κάτω, δεξιά, αριστερά και διαγώνια του (σύνολο 8), έχει ένα ή δύο κεφαλές ηλεκτρονίου.

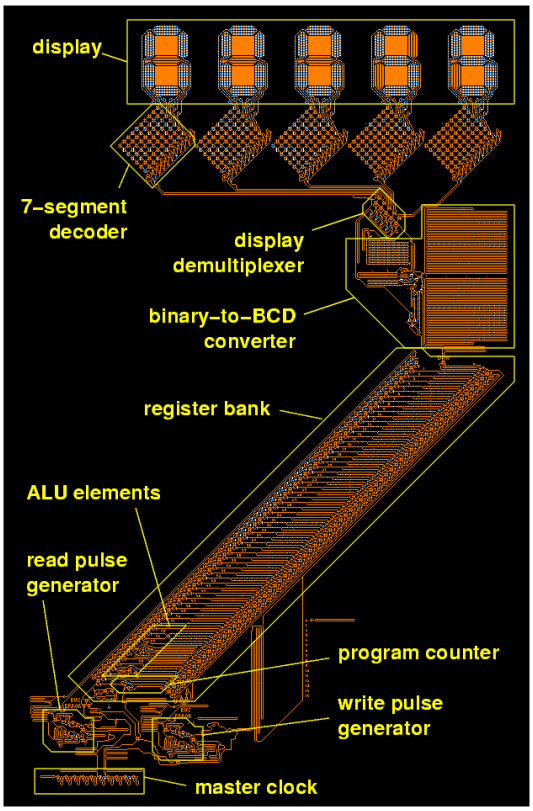
Οι συγκεκριμένες ιδιότητες δεν επιτρέπουν μόνο την δημιουργία διόδων αλλά και λογικών πυλών, clock generators (δηλαδή την περιοδική δημιουργία ηλεκτρονίου) , ROM και RAM δημιουργώντας κυκλώματα.



Αριστερά: Στιγμιότυπο πύλης OR ,Κέντρο: Στιγμιότυπο ROM , Δεξιά: Στιγμιότυπο RAM 7,9,11 κύκλων.
(πηγή: karlscherer.com)

Γενικότερα, όπως ειπώθηκε και προηγουμένως, το «Wireworld» είναι Turing-complete. Άρα, είναι δυνατή η ανάπτυξη turing equivalent υπολογιστικής μηχανής με «Wireworld».

Ένα τέτοιο παράδειγμα είναι το παρακάτω πλέγμα «Wireworld» το οποίο υπολογίζει με την σειρά τους πρώτους αριθμούς που είναι μικρότεροι από 65.536 (2^16) και τους εμφανίζει. Για να ελέγξει αν ο περιττός P είναι πρώτος αριθμός, τον ελέγχει με τους περιττούς αριθμούς από 3 έως P-2.



(Πηγή: www.quinapalus.com)

Το συγκεκριμένο πλέγμα σχεδιάστηκε από τον Michael Fryers και αποτελείται από 62.037 κύτταρα. Πράγματι, κάνοντας το simulate στο συγκεκριμένο πλέγμα εμφανίζονται στο πάνω μέρος με την σειρά οι πρώτοι αριθμοί. Όμως είναι εύκολο να διαπιστωθεί πως λόγω της μορφής του αλγόριθμου που χρησιμοποιεί το πλέγμα, κάθε επόμενος υποψήφιος πρώτος αριθμός τείνει να είναι πιο δύσκολο να εξεταστεί. Ενδεικτικά, στο παραπάνω στιγμιότυπο το πλέγμα βρίσκεται περίπου στην 13.500.500^{η} γενιά και εμφανίζει τον 17ο πρώτο αριθμό (το 59), ενώ για να εμφανιστεί ο 36^{ος} πρώτος αριθμός (το 151) χρειάζονται 100.000.000 γενιές συνολικά.

Δημιουργώντας έναν Simulator για το Wireworld

Ο βασικός στόχος που τέθηκε στο συγκεκριμένο project ήταν η προσομοίωση του παραπάνω πλέγματος. Φυσικά υπάρχουν αρκετά εργαλεία, όπως το open-source Golly, και αρκετοί αλγόριθμοι για διάφορες γλώσσες (https://rosettacode.org/wiki/Wireworld) για αυτό το πρόβλημα. Ωστόσο, αναπτύχθηκε και υπάρχει στα αρχεία της εργασίας**, στον φάκελο WirePrime**, ένα ξεχωριστό simulator του Wireworld, ονόματι WirePrime, χάρη του παραπάνω πλέγματος. Στον φάκελο αυτό και στους υποφακέλους του υπάρχουν διάφορες εκδόσεις του προγράμματος οι οποίες θα αναλυθούν στην συνέχεια σύμφωνα με την σειρά δημιουργία τους, καθώς η μια αποτελεί βελτίωση της άλλης.

- **GCC/Vers0**

Είναι η πρωτότυπη έκδοση του simulator WirePrime, η οποία αναπτύχθηκε με Dev-C++ σε C και έγινε compile με το TDM-GCC 4.9.2 32-bit. Ο κώδικας μπορεί να χωριστεί στα εξής κομμάτια:

-Input του πλέγματος: Το παραπάνω πλέγμα υπάρχει στο φάκελο "PatternsRLE" της εργασίας (μαζί και με άλλα πλέγματα που περιέχει εξαρχής το πρόγραμμα simulate του Wireworld, Golly) σε μορφή αρχείου *.rle (Extended RLE) με όνομα "primes.rle". Η περιγραφή του format αρχείου .rle βρίσκεται στον ιστότοπο http://golly.sourceforge.net/Help/formats.html#rle που με την βοήθειά της έχει αναπτυχθεί κατάλληλος κώδικας ο οποίος «φορτώνει» σε δισδιάστατο πίνακα κατάλληλου μεγέθους το πλέγμα σε μορφή .rle το οποίο δίνεται ως argument στο πρόγραμμα, δίνοντας σε κάθε στοιχείο του πίνακα την τιμή που του αντιστοιχεί (από το 0 έως το 3).

-Δημιουργία επόμενης γενιάς: Δημιουργεί έναν νέο πίνακα ίδιου μεγέθους με τον πίνακα που περιέχει το πλέγμα και έπειτα, για κάθε γενιά, βρίσκει και σημειώνει στον νέο πίνακα την κατάσταση κάθε κυττάρου στην επόμενη γενιά (εξετάζοντας την τιμή του κυττάρου στον αρχικό πίνακα και την γειτονιά Moore του). Τέλος αντιγράφει τον πίνακα που περιέχει την νέα γενιά στην θέση του πίνακα της παλιάς γενιάς και επαναλαμβάνει την διαδικασία.

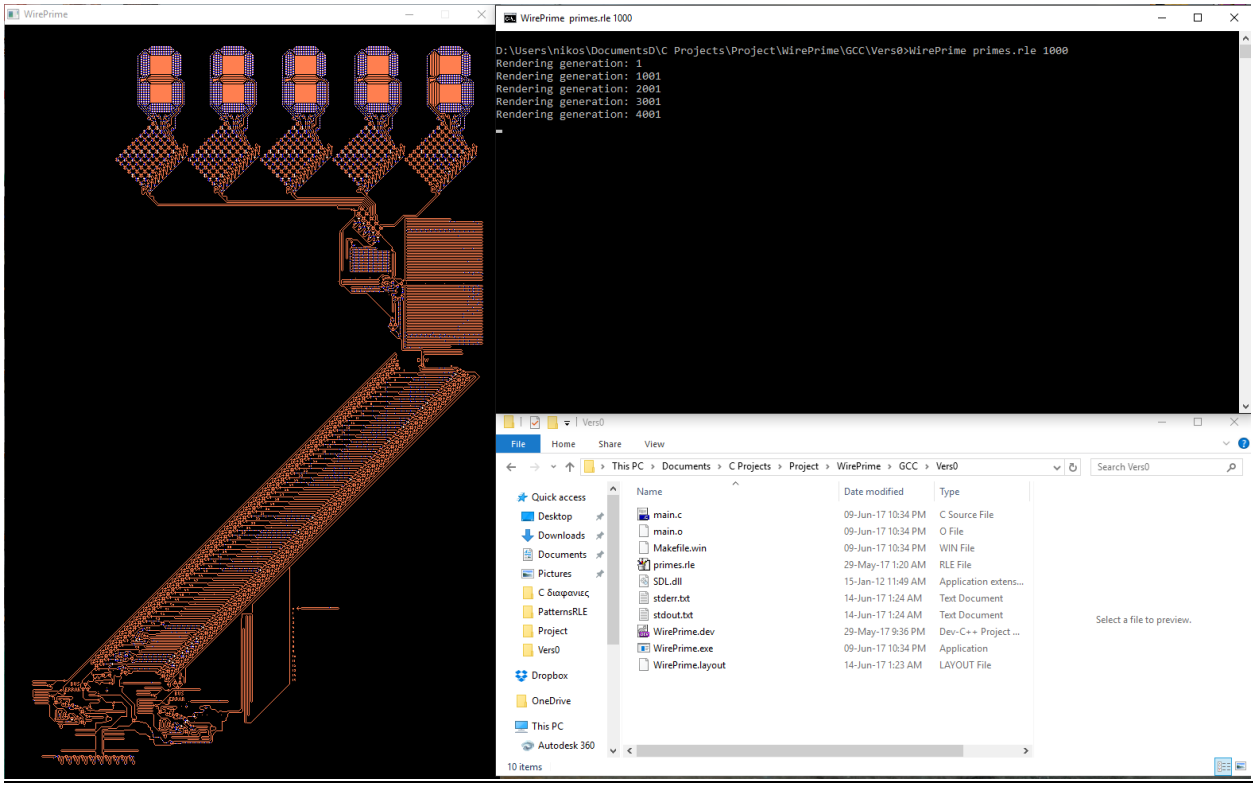
-Απεικόνιση γενιάς: Χρησιμοποιώντας την βιβλιοθήκη SDL 1.2.15 (Simple Directmedia Layer) δημιουργεί ένα παράθυρο κατάλληλων διαστάσεων και ανά ορισμένες γενιές, που ορίζονται από τον χρήστη (default ανά 1.000 γενιές), ανανεώνει το παράθυρο παρουσιάζοντας την τελευταία γενιά. Προσθέτει στο πρόγραμμα ένα dependence στο DDL: SDL.ddl το οποίο υπάρχει στον φάκελο. Η βιβλιοθήκη μπορεί να βρεθεί στον εδώ αλλά και στα αρχεία της εργασίας.

Το WirePrime για να εκτελεστεί πρέπει από το τερματικό να δοθούν ορισμένα arguments με την εξής σειρά*:*

WirePrime (όνομα .rle που περιέχει το πλέγμα) (κάθε πόσες γενιές να γίνει ανανέωση του παραθύρου με την τελευταία γενιά)

Για παράδειγμα:

WirePrime primes.rle 1000



- **GCC/Vers1**

Αποτελεί optimize (βελτίωση) του Vers0.
Στο Vers0 για την δημιουργία μιας νέας γενιάς το πρόγραμμα εξετάζει όλα τα στοιχεία του πίνακα που περιέχει την τρέχουσα γενιά. Όμως ορισμένα κύτταρα είναι κενά, δηλαδή η εξέταση της τιμής τους στην επόμενη γενιά είναι άσκοπη ,καθώς θα παραμείνουν κενά, και ξοδεύει από τον επεξεργαστή χρόνο. Για αυτό το λόγο στο Vers1 (και έπειτα) μόλις το πρόγραμμα κάνει input το πλέγμα δημιουργεί έναν πίνακα ο οποίος περιέχει τις θέσεις μόνο των κυττάρων που αξίζει να βρεθεί η τιμή τους στις επόμενες γενιές και την στέλνει στο τμήμα του προγράμματος που δημιουργεί την επόμενη γενιά.

- **GCC/Vers2**

Αποτελεί optimize (βελτίωση) του Vers1.
Ο πολλαπλασιασμός είναι μια ακριβή πράξη και στις εκδόσεις Vers0, Vers1 εμφανίζεται στο τμήμα δημιουργίας νέας γενιάς αρκετά συχνά . Για αυτό το λόγο στην έκδοση Vers2 μειώνουμε στο ελάχιστο τους πολλαπλασιασμούς αποθηκεύοντας τα γινόμενα επαναλαμβανόμενων πολλαπλασιασμών σε μεταβλητές.

- **GCC/Vers3**

Αποτελεί optimize (βελτίωση) του Vers2.
Στο Vers2 και στις προηγούμενες εκδόσεις όταν δημιουργείται ο πίνακας της νέας γενιάς και συμπληρώνεται, αντιγράφεται στον πίνακα της παλιάς γενιάς. Μία τέτοια διαδικασία μειώνει την ταχύτητα του προγράμματος μας καθώς το πρόγραμμα δημιουργεί χιλιάδες νέες γενιές ανά δευτερόλεπτο και σε κάθε γενιά πρέπει να κάνει copy ένα μεγάλο block μνήμης. (Στο πλέγμα του primes.rle, κάνει copy 515.9 KB ανά γενιά.)

Στο Vers3 όμως το πρόγραμμα κάνει swap - εναλλάσσει τους δείκτες των δύο block μνήμης, του πίνακα με την νέα γενιά και του πίνακα της παλιάς γενιάς, αποφεύγοντας την αντιγραφή στην μνήμη. Το συγκεκριμένο κόλπο δεν επηρεάζει το υπόλοιπο πρόγραμμα, παρά μόνο ως προς την ταχύτητά του.

- **GCC/Vers4**

Αποτελεί optimize (βελτίωση) του Vers3 σε ορισμένες μόνο πλατφόρμες.
Οι πίνακες αλλάζουν τύπο από char σε uint_fast8_t. Σε αρχιτεκτονική x86 δεν παρατηρήθηκε κάποια αύξηση επιδόσεων, ωστόσο σε κάποια άλλη μπορεί να υπάρχει.

Σε όλες τις εκδόσεις ζητήσαμε στο Dev-C++ από τον complier μέγιστο optimize με compatibility με επεξεργαστές πυρήνα Intel Conroe (2006).
Οι βιβλιοθήκη SDL 1.2 και οι απαραίτητες ρυθμίσεις είναι ενσωματωμένες στα αρχεία του project. Αν υπάρχει πρόβλημα οδηγίες εγκατάστασης της βιβλιοθήκης υπάρχουν εδώ .

Έπειτα, με στόχο να δημιουργηθεί multi-threading έκδοση του προγράμματος επιλέχθηκε να γίνει αλλαγή compiler και να συνεχίσουμε την ανάπτυξη του προγράμματος σε Visual C++ στο Visual Studio 2017 , καθώς μας προσφέρει την βιβλιοθήκη “ppl.h” (Parallel Patterns Library). Οι επόμενες δύο εκδόσεις που αναφέρονται αναπτύχθηκαν ως ενδιάμεσα στάδια της έκδοσης “Visual C++ with PPL”.

- **Visual C**

Αποτελεί την μεταφορά της Vers4 σε Visual C.
Για την μεταφορά του προγράμματος και την μεταγλώττιση του απαιτήθηκε η αλλαγή της βιβλιοθήκης του συστήματος «απεικόνισης γενιάς» του προγράμματος από SDL 1.2 σε SDL 2 (που μπορεί να βρεθεί στα αρχεία της εργασίας και εδώ) ,καθώς η πρώτη δεν υποστήριζε την Visual C++ 15.0. Επίσης έγιναν και άλλες μικρές αλλαγές. Δημιουργήθηκαν εκτελέσιμα αρχεία για Windows x86 και Windows x64. Πλέον το πρόγραμμα έχει dependence στο: SDL2.ddl (το .ddl για x86 διαφέρει από το αντίστοιχο για x64).

- **Visual C++**

Αποτελεί την μεταφορά του κώδικα της έκδοσης Visual C σε Visual C++.
Χρειάστηκαν μόνο μικρές αλλαγές.
Δημιουργήθηκαν εκτελέσιμα αρχεία για Windows x86 και Windows x64.

- **Visual C++ with PPL**

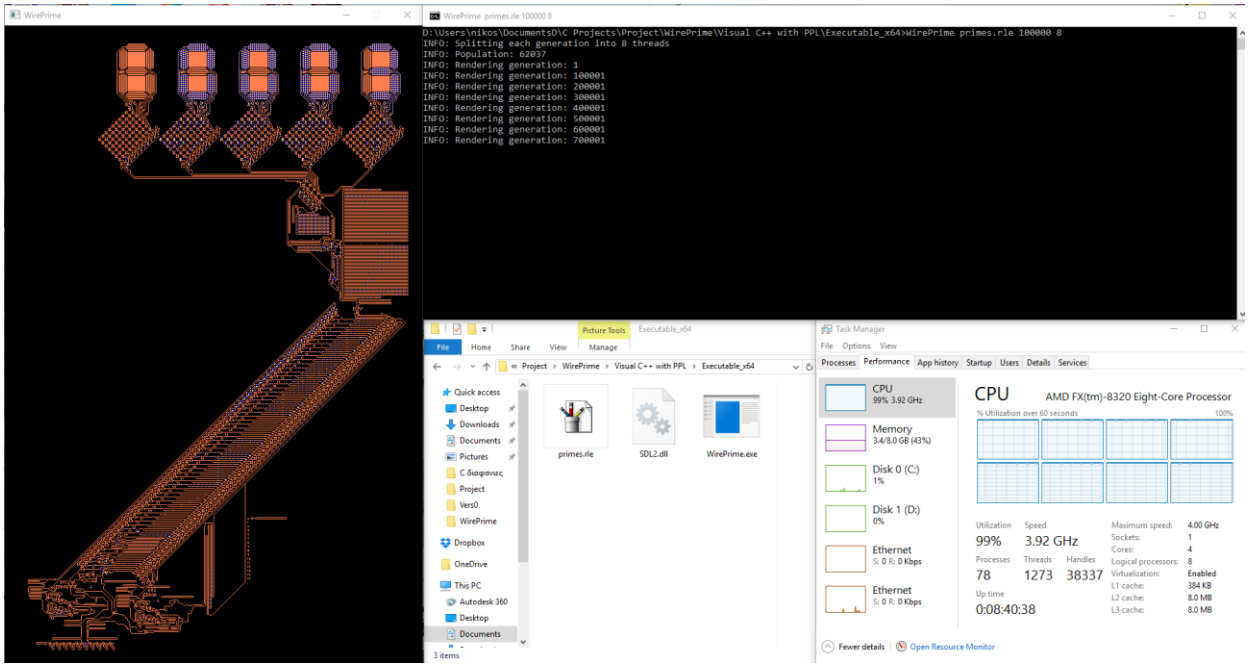
Σε αυτήν την έκδοση γίνεται χρήση της βιβλιοθήκης ppl.h (Parallel Patterns Library) για να παραλληλοποιηθεί η διαδικασία της δημιουργίας νέας γενιάς. Ο χρήστης μέσω ενός επιπλέον argument (βλέπε παρακάτω) ενημερώνει το πρόγραμμα σε πόσα threads να διαμερίζεται η «δουλειά» δημιουργίας νέας γενιάς (default 2 threads). Αναλυτικότερα, μοιράζει τον πίνακα που περιέχει τις θέσεις των κυττάρων που αξίζει να βρεθεί η τιμή (βλέπε GCC/Vers1) σε (threads) ίσα κομμάτια και αναθέτει κάθε ένα σε ένα διαφορετικό thread.
Δημιουργήθηκαν εκτελέσιμα αρχεία για Windows x86 και Windows x64.

Η συγκεκριμένη έκδοση του WirePrime για να εκτελεστεί πρέπει να δοθούν από το τερματικό ορισμένα arguments με την εξής σειρά:

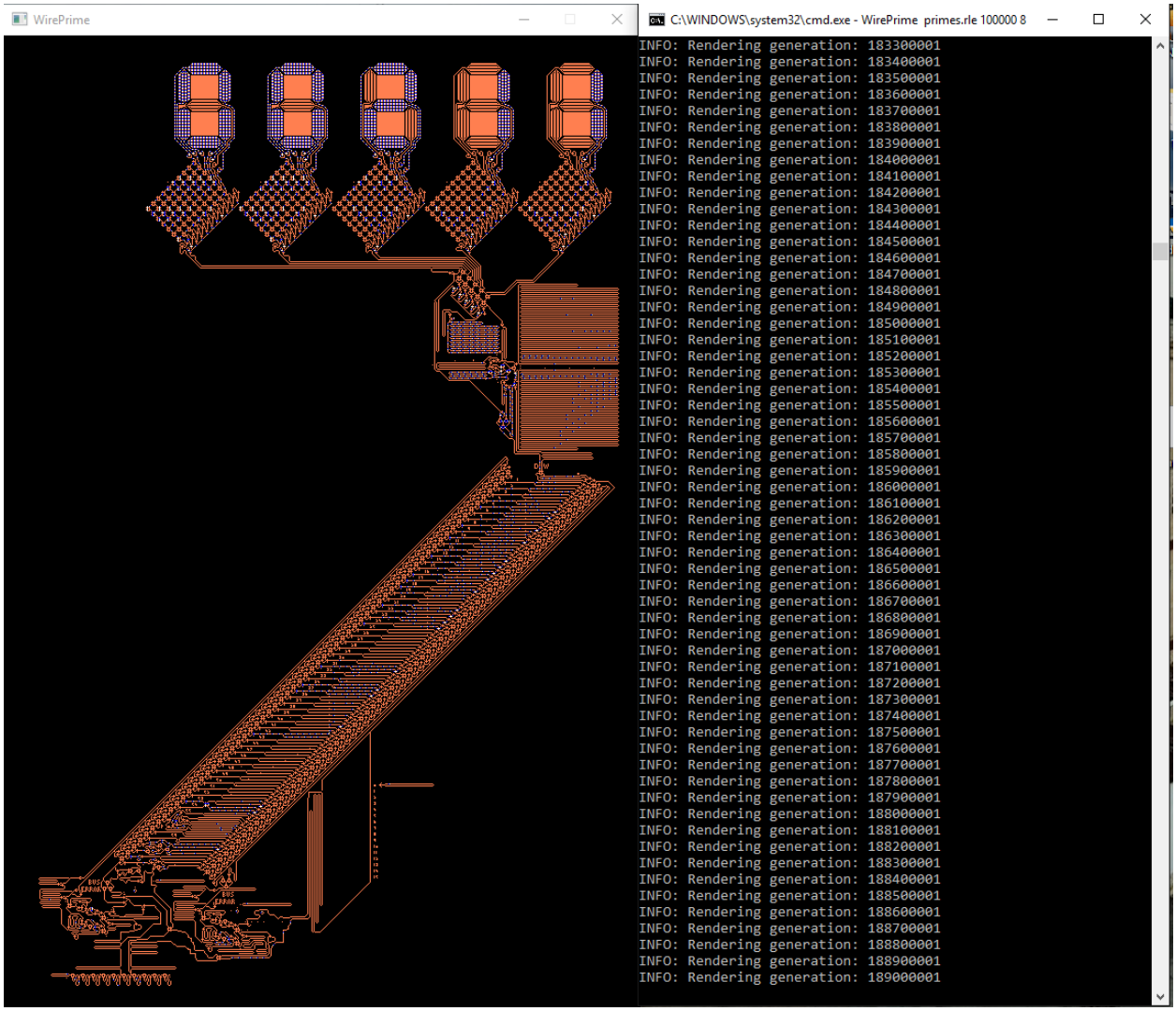
WirePrime (όνομα .rle που περιέχει το πλέγμα) (κάθε πόσες γενιές να γίνει ανανέωση του παραθύρου με την τελευταία γενιά) (threads που θέλουμε να χρησιμοποιήσει το πρόγραμμα στην δημιουργία νέας γενιάς)

Για παράδειγμα, αν έχουμε επεξεργαστή με 8 threads (λογικούς πυρήνες) και θέλουμε την πλήρη χρήση του για την δημιουργία νέων γενιών:

WirePrime primes.rle 10000 8



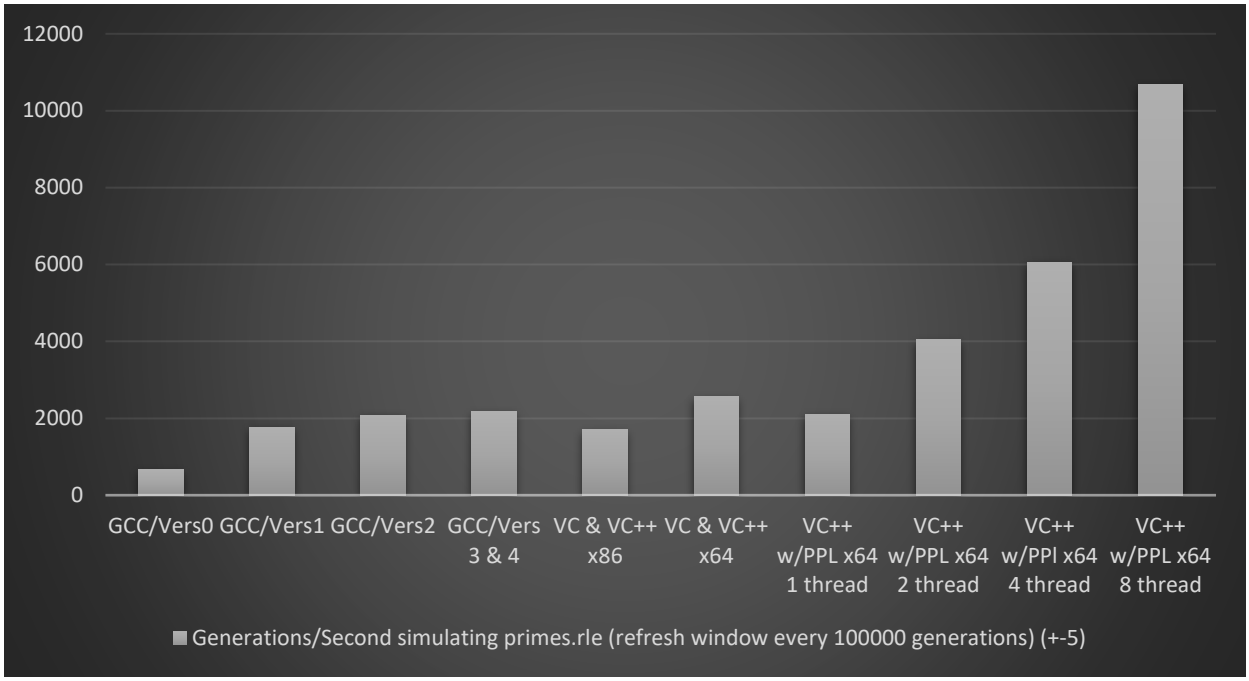
Επίσης, η έκδοση Visual C++ with PPL x64 έπειτα από περίπου 5:00 ώρες λειτουργίας.



Στις τελευταίες τρεις εκδόσεις ζητήσαμε στο Visual Studio από τον compiler (μόνο) μέγιστο optimize.
Δεν απαιτείται το Microsoft Visual C++ 2017 Redistributable. Έχουν γίνει οι απαραίτητες ρυθμίσεις.
Οι βιβλιοθήκη SDL 2 και οι απαραίτητες ρυθμίσεις είναι ενσωματωμένες στα αρχεία του project. Αν υπάρχει πρόβλημα οδηγίες εγκατάστασης της βιβλιοθήκης υπάρχουν εδώ .
Η έκδοση του WirePrime: Visual C++ with PPL σε ορισμένες περιπτώσεις μπορεί να stress – άρει τον επεξεργαστή σε υπέρμετρο βαθμό.

Τέλος, αξίζει να συγκριθούν οι εκδόσεις του προγράμματος για να διαπιστωθεί ο αντίκτυπος κάθε αλλαγής στις επιδόσεις του, καθώς η αύξηση των επιδόσεων του ήταν ένας από τους στόχους. Η VC++ w/PPL x86 έκδοση παραλείπεται.

Αξιοσημείωτες είναι οι αλλαγές της έκδοσης GCC/Vers1 και το πόσο ευνοήθηκε ο σειριακός αλγόριθμος από την παραλληλοποίηση του.



CPU: AMD FX-8320 @ 4.2GHz (8 cores / 8 threads)

Συνοψίζοντας το «Wireworld» είναι ένα απλό κυτταρικό αυτόματο με αρκετές δυνατότητες. Η χρήση του Η/Υ για μεγάλα πλέγματα είναι αναγκαία. Το πρόγραμμα που αναπτύχθηκε στα πλαίσια του συγκεκριμένου project μπορεί να κάνει simulate οποιοδήποτε πλέγμα του Wireworld. Όμως, δεν ξεπερνά τα πιο σύνθετα εργαλεία ,όπως το Golly, τα οποία έχουν πιο αποτελεσματικούς αλγόριθμους simulate. Να σημειωθεί πως για παράδειγμα στο “Conway's Game of Life” υπάρχει ο αλγόριθμος “Hashlife” ο οποίο επιταχύνει πολλαπλά την διαδικασία δημιουργίας γενιάς. Ωστόσο στο project ένας απλός αλγόριθμος που αποτελεί κοινότοπο στο simulate του Wireworld έγινε optimize και παραλληλοποιήθηκε, δημιουργήθηκε ένα μοντέρνο σύστημα απεικόνισης γενιάς , εξερευνήθηκαν σύγχρονες βιβλιοθήκες παραλληλοποίησης αλγορίθμων.

Πηγές

- https://www.quinapalus.com/wires11.html
- http://psoup.math.wisc.edu/mcell/rullex_rtab.html#WireWorld
- https://en.wikipedia.org/wiki/Cellular_automaton
- https://en.wikipedia.org/wiki/Wireworld
- https://en.wikipedia.org/wiki/Hashlife

Εργαλεία

- Golly για πλέγμα και μετατροπή πλεγμάτων σε .rle: http://golly.sourceforge.net/
- SDL: https://www.libsdl.org/
- Dev C++: http://www.bloodshed.net/devcpp.html
- Visual Studio 2017: https://www.visualstudio.com/
- Documentation για “ppl.h”: https://msdn.microsoft.com/el-gr/library/dd492418.aspx
- Documentation για εντολές του "concrt.h" που χρησιμοποιήθηκαν: https://msdn.microsoft.com/en-us/library/dd470841.aspx