Πανεπιστήμιο Δυτικής Μακεδονίας
Τμήμα Μηχανικών Πληροφορικής & Τηλεπικοινωνιών

Αρχές και λειτουργίες του OpenACC

Παναγιωτίδου Παναγιώτα
ΑΜ:729

Επιβλέπων Καθηγητής: Δρ. Μηνάς Δασυγένης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Σκοπός της ενότητας

- Εισαγωγή στο OpenACC
- Τα βασικά στοιχεία του OpenACC
- Παραλληλοποίηση βρόχων
- Παραδείγματα της δομής των οδηγιών kernels, parallel και atomic
- Η χρήση του OpenACC
- Διαφορές μεταξύ των directives kernels και parallel
- Παράδειγμα: Επίλυση εξίσωσης Laplace
- Επιτάχυνση με ενοποιημένη μνήμη
- Βιβλιοθήκες χρόνου εκτελεσης
- Προχωρημένη Ανάλυση του κώδικα του OpenACC-χρήση του εργαλείου PGPROF
- Προχωρημένη Ανάλυση του κώδικα του OpenACC-χρήση του εργαλείου Visual Profiler της NVIDIA
- Συμπεράσματα Ενότητας
- Βιβλιογραφία

Εισαγωγή στο OpenACC

Τι είναι το OpenACC;

- Είναι ενα πρότυπο προγραμματισμού παραλληλοποίησης και έχει σχεδιαστεί για να απλοποιεί τον παράλληλο προγραμματισμό ετερογενών συστημάτων CPU/GPU.
  - Συντάχθηκε από: την NVIDIA, Cray, PGI και CAPS
  - Μπορεί να υποστηρίξει τις γλώσσες προγραμματισμού: Fortran, C, (και ίσως) C++.
  - Το χρονολογικό πλαίσιο ανάπτυξης του OpenACC παρατίθεται παρακάτω:
    - 2010 - Το OpenACC ιδρύθηκε από CAPS, Cray, PGI και NVIDIA, για την ενοποίηση των οδηγιών για τους επιταχυντές που αναπτύσσονται ανεξάρτητα από CAPS, Cray και PGI
    - 2011 - Κυκλοφορία της έκδοσης του OpenACC 1.0.
    - 2013 - Κυκλοφορία του OpenACC 2.0 , προσθέτοντας υποστήριξη για αδόμητη διαχείριση δεδομένων και διευκρινίζοντας τη γλώσσα των προδιαγραφών.
    - 2015 - Το OpenACC 2.5 κυκλοφόρησε, περιέχει κυρίως διευκρινίσεις με κάποια επιπλέον χαρακτηριστικά.
    - 2017- Κυκλοφορία της τελευταίας έκδοσης των προδιαγραφών, έκδοση OpenACC 2.6.

OpenACC®
DIRECTIVES FOR ACCELERATORS

Υποστήριξη Μεταγλωττιστών (1/3)

- Πολλαπλοί μεταγλωττιστές προσφέρουν φορητότητα, αποσφαλμάτωση, σταθερότητα
  - Η υποστήριξη του OpenACC είναι διαθέσιμη σε εμπορικούς μεταγλωττιστές από τον PGI (από την έκδοση 12.6), και από τον Cray (μόνο για το hardware της Cray)
  - Ο OpenUH είναι ένας μεταγλωττιστής ,βασιζόμενος στον Open64 ανοιχτού κώδικα OpenACC που υποστηρίζει C και FORTRAN και αναπτύχθηκε από την ομάδα HPCTools από το Πανεπιστήμιο του Χιούστον.
  - Ο OpenARC είναι ένας μεταγλωττιστής ανοικτού κώδικα C που αναπτύχθηκε στο Εθνικό Εργαστήριο Oak Ridge για να υποστηρίξει όλα τα χαρακτηριστικά της προδιαγραφής OpenACC 1.0.
  - Ο accULL είναι ένας πειραματικός μεταγλωττιστής ανοικτού κώδικα και έχει αναπτυχθεί από το Πανεπιστήμιο της La Laguna (μόνο για τη γλώσσα C).
  - Ο IPMACC είναι ένας μεταγλωττιστής ανοικτού κώδικα C που αναπτύχθηκε από το Πανεπιστήμιο της Βικτώρια και μεταφράζει το OpenACC σε CUDA, OpenCL και ISPC. Προς το παρόν, υποστηρίζονται μόνο οι ακόλουθες οδηγίες: data, kernels, loop και cache.

Υποστήριξη Μεταγλωττιστών (2/3)

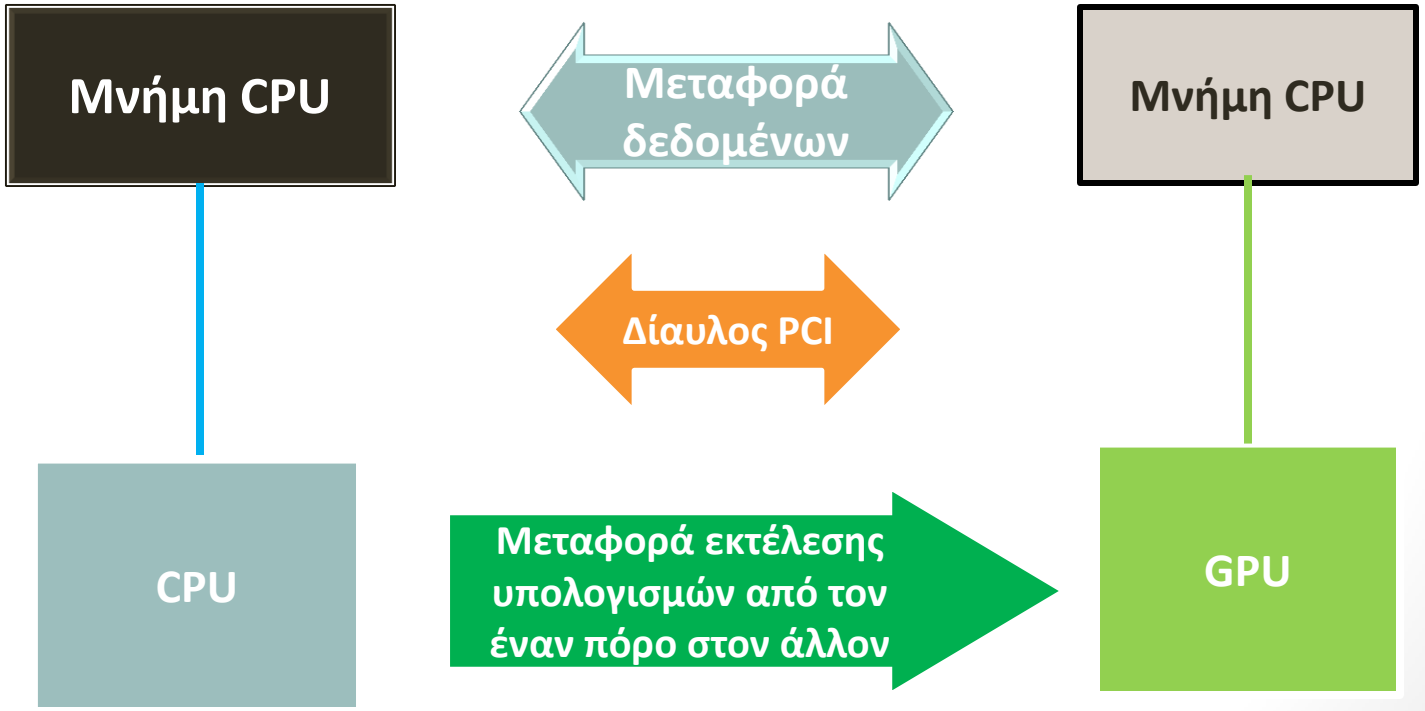
- Η υποστήριξη του GCC για το OpenACC επήλθε αρκετά αργότερα. Τον Σεπτέμβριο του 2013 ανακοινώθηκε μια υλοποίηση που στοχεύει στη GPU από τη Samsung, η οποία μετέφρασε το OpenACC 1.1 σε OpenCL-με τη δυνατότητα προσθήκης σχολίων στον κώδικα-. Η ανακοίνωση μιας "πραγματικής" εφαρμογής ακολούθησε δύο μήνες αργότερα, αυτή τη φορά από τη NVIDIA και βασίστηκε στο OpenACC 2.0.
- Αυτό δημιούργησε κάποια διαμάχη, καθώς η υλοποίηση θα αφορούσε μόνο τη γλώσσα PTX assembly της NVIDIA, για την οποία δεν ήταν διαθέσιμος κανένας αθροιστής ανοιχτού κώδικα ή χρόνου εκτέλεσης.
- Η πειραματική υποστήριξη για το OpenACC / PTX κατέληξε στο GCC της έκδοση 5.1. Οι σειρές GCC6 και GCC7 περιλαμβάνουν μια πολύ πιο βελτιωμένη υλοποίηση της προδιαγραφής OpenACC 2.0a.
- Σε σύγκριση με το GCC 7, η σειρά έκδοσης GCC 8 περιέχει κάποιες αλλαγές. Η κατάσταση εφαρμογής στο openacc-gcc-8-branch βασίζεται στη σειρά έκδοσης GCC 8, με πρόσθετη υποστήριξη για την προδιαγραφή OpenACC 2.5 (GCC 8.1 κυκλοφόρησε στις 2018-05-02).
- Όπως και στο OpenMP, ο προγραμματιστής μπορεί να σχολιάσει στον πηγαίο κώδικα της C , C ++ και Fortran για να προσδιορίσει τις περιοχές που πρέπει να επιταχυνθούν, χρησιμοποιώντας τις οδηγίες (directives) του μεταγλωττιστή, καθώς και πρόσθετες λειτουργίες.
- Όπως στο OpenMP 4.0 και το νεότερο, ο κώδικας μπορεί να ξεκινήσει τόσο στην CPU όσο και στη GPU .

Υποστήριξη Μεταγλωττιστών (3/3)

- Στο International Supercomputing Conference το 2012 αποδείχθηκε ότι το OpenACC λειτουργεί σε επιταχυντές της Nvidia, της AMD και της Intel, χωρίς δεδομένα απόδοσης.
- Πιο συγκεκριμένα στις 12 Νοεμβρίου 2012, στο ίδιο συνέδριο, παρουσιάστηκε ένα σχέδιο της προδιαγραφής OpenACC έκδοση 2.0.
- Παρουσιάστηκαν νέες προτεινόμενες δυνατότητες οι οποίες περιλαμβάνουν νέους ελέγχους για τη μεταφορά δεδομένων (όπως καλύτερη διαχείριση των μη δομημένων δεδομένων και βελτιωμένη υποστήριξη για μη συνεχή μνήμη) καθώς και υποστήριξη για άμεσες κλήσεις συναρτήσεων και διαχωρισμένη μεταγλώττιση (επιτρέποντας τη δημιουργία και επαναχρησιμοποίηση βιβλιοθηκών επιταχυνόμενου κώδικα).
- Το OpenACC 2.0 κυκλοφόρησε επίσημα τον Ιούνιο του 2013.
- Ο κύριος τρόπος προγραμματισμού στο OpenACC είναι οι οδηγίες (directives), με τρόπο παρόμοιο με το OpenMP 3.x για ομοιογενές σύστημα ή για το προγεννέστερο OpenHMPP.
- Οι προδιαγραφές περιλαμβάνουν επίσης μια βιβλιοθήκη χρόνου εκτέλεσης που ορίζει διάφορες λειτουργίες υποστήριξης.

Βασική ιδέα του OpenACC

- Για αποδοτικότητα, διαχωρείστε την κίνηση δεδομένων και υπολογίστε την μεταφορά εκτέλεσης των υπολογισμών μεταξύ των πόρων.



Στόχοι της φορητότητας του OpenACC

Φορητότητα μεταγλωττιστή

- Διαφορετικοί μεταγλωττιστές πρέπει να υποστηρίζουν τις ίδιες οδηγίες (directives/pragmas) και βιβλιοθήκες κατά του χρόνου εκτέλεσης.

Στόχοι της φορητότητας του OpenACC

Φορητότητα συσκευής

- Σχεδιάστηκε με την προυπόθεση να είναι αρκετά υψηλού επιπέδου για να υποστηρίζει οποιοδήποτε σημερινό ή μελλοντικό επιταχυντή.
- Εξαλείφει την ανάγκη για ξεχωριστές διακλαδώσεις του κώδικα για τη CPU και τις GPUs.

Στόχοι της φορητότητας του OpenACC

Φορητότητα εκτέλεσης

- Δεδομένου ότι το OpenACC αναφέρει μόνο τον κώδικα, ο καλογραμμένος κώδικας πρέπει να έχει καλές επιδόσεις είτε πρόκειται για τη CPU είτε για τη GPU.

GPU & GPGPU

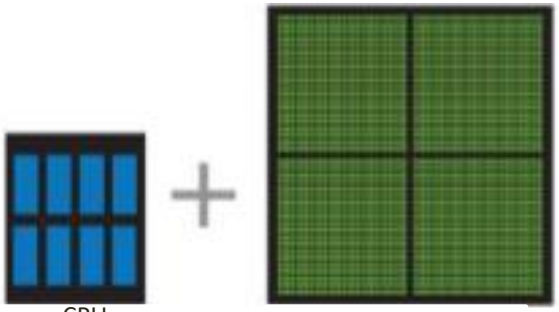
- Αρχικά, η μονάδα επεξεργασίας γραφικών (GPU) είναι υπεύθυνη για το χειρισμό γραφικών υπολογιστών και επεξεργασίας εικόνας.
- Πραδοσιακά, η GPU είναι γνωστή ως “video card”.

GPU & GPGPU

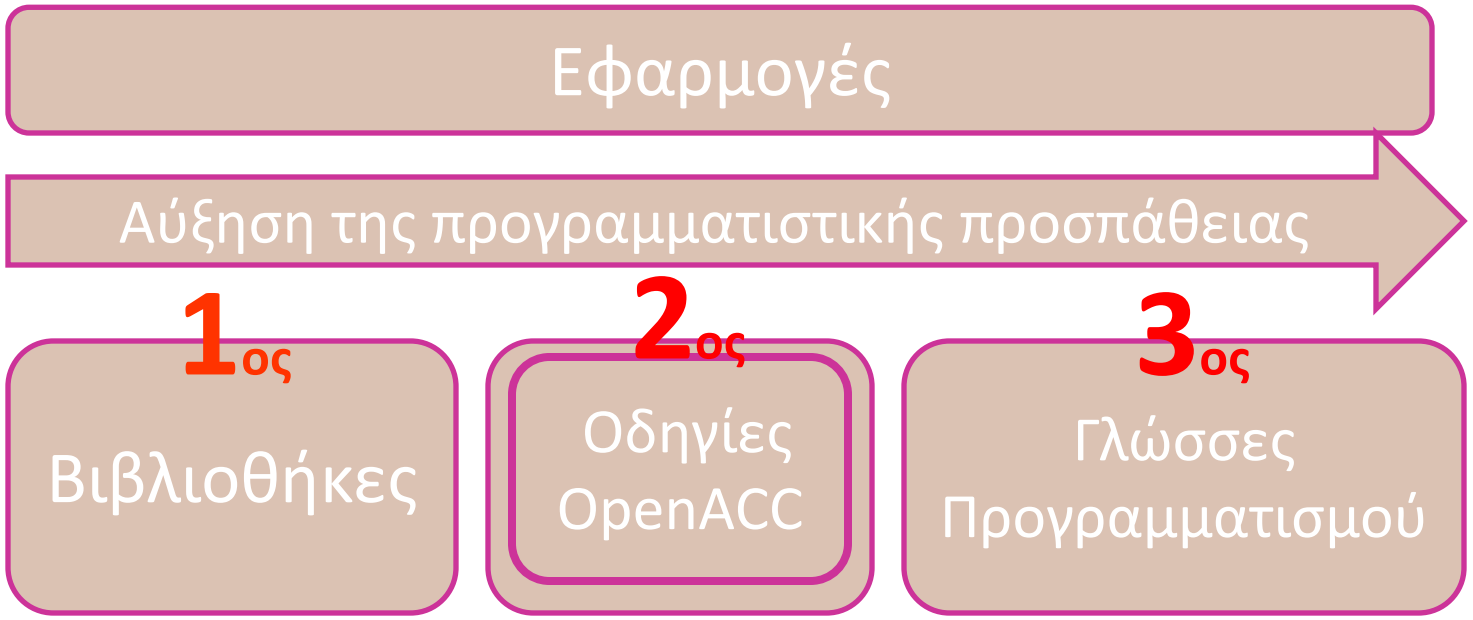
- Η εξαιρετικά παράλληλη δομή της την καθιστά αποδοτική για τα παράλληλα προγράμματα.
- Σήμερα, οι GPU χρησιμοποιούνται για εργασίες που εκτελούσαν παλιότερα οι CPU, όπως η υπολογιστική πληροφορική. Για τέτοιου είδους εργασίες η GPU ονομάζεται General Purpose GPU (GPGPU).
- Σε πολλές περιπτώσεις, ένα παράλληλο πρόγραμμα λειτουργεί ταχύτερα σε GPU παρά σε CPU.

GPU & GPGPU

- Σημειώστε ότι ένα σειριακό πρόγραμμα τρέχει πιο γρήγορα στη CPU από ο,τι στη GPU.
- Ο πιο δημοφιλής τύπος GPU στον κόσμο των υπολογιστών υψηλής απόδοσης είναι η GPU της NVIDIA, όπου και θα επικεντρωθούμε.



3 τρόποι για να επιταχυνθούν οι εφαρμογές στη GPU



Έτοιμες βιβλιοθήκες με κώδικα

Εύκολη επιτάχυνση εφαρμογών

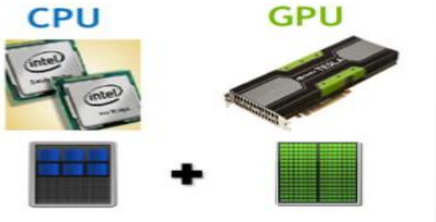
Μέγιστη Ευελιξία

Οι CUDA βιβλιοθήκες είναι διαλειτουργικές με το OpenACC

Οι CUDA γλώσσες είναι επίσης διαλειτουργικές με το OpenACC

Η GPU είναι ένας επιταχυντής

- Η GPU είναι μια συσκευή βασισμένη στο σύστημα της CPU. Η GPU συνδέεται με τη CPU μέσω του διαύλου PCI (Peripheral Component Interconnect (PCI)), ο οποίος αποτελεί ένα κανάλι ή μια διαδρομή μεταξύ των στοιχείων σε έναν υπολογιστή.
- Το πρόγραμμα μπορεί να παραλληλοποιηθεί και να επιταχυνθεί στη GPU.
- Η CPU και η GPU έχουν ξεχωριστές μνήμες.
- Η μεταφορά δεδομένων μεταξύ CPU και GPU απαιτείται για προγραμματισμό. Μεταξύ της CPU και GPU απαιτείται μεταφορά δεδομένων.



Το μοντέλο του επιταχυντή GPU στο OpenACC

- Για να εξασφαλιστεί η φορητότητα του OpenACC σε όλες τις αρχιτεκτονικές, καθορίζεται ένα αφηρημένο μοντέλο επιτάχυνσης του προγραμματισμού.

- Εκθέτει πολλαπλά επίπεδα παραλληλοποίησης, καθώς και μια ιεραρχία μνημών με διαφορετικούς βαθμούς ταχύτητας και διευθυνσιοδότησης.

Το μοντέλο του επιταχυντή GPU στο OpenACC

- Στόχος αυτού του μοντέλου είναι να διασφαλίσει ότι το OpenACC θα εφαρμόζεται σε περισσότερες από μία αρχιτεκτονικές όπως και να μπορεί να χρησιμοποιηθεί και σε μελλοντικές συσκευές.

- Στον πυρήνα του, το OpenACC υποστηρίζει την μεταφορά-μεταφόρτωση των υπολογισμών και δεδομένων από την host συσκευή στη συσκευή επιτάχυνσης.

Το μοντέλο του επιταχυντή GPU στο OpenACC

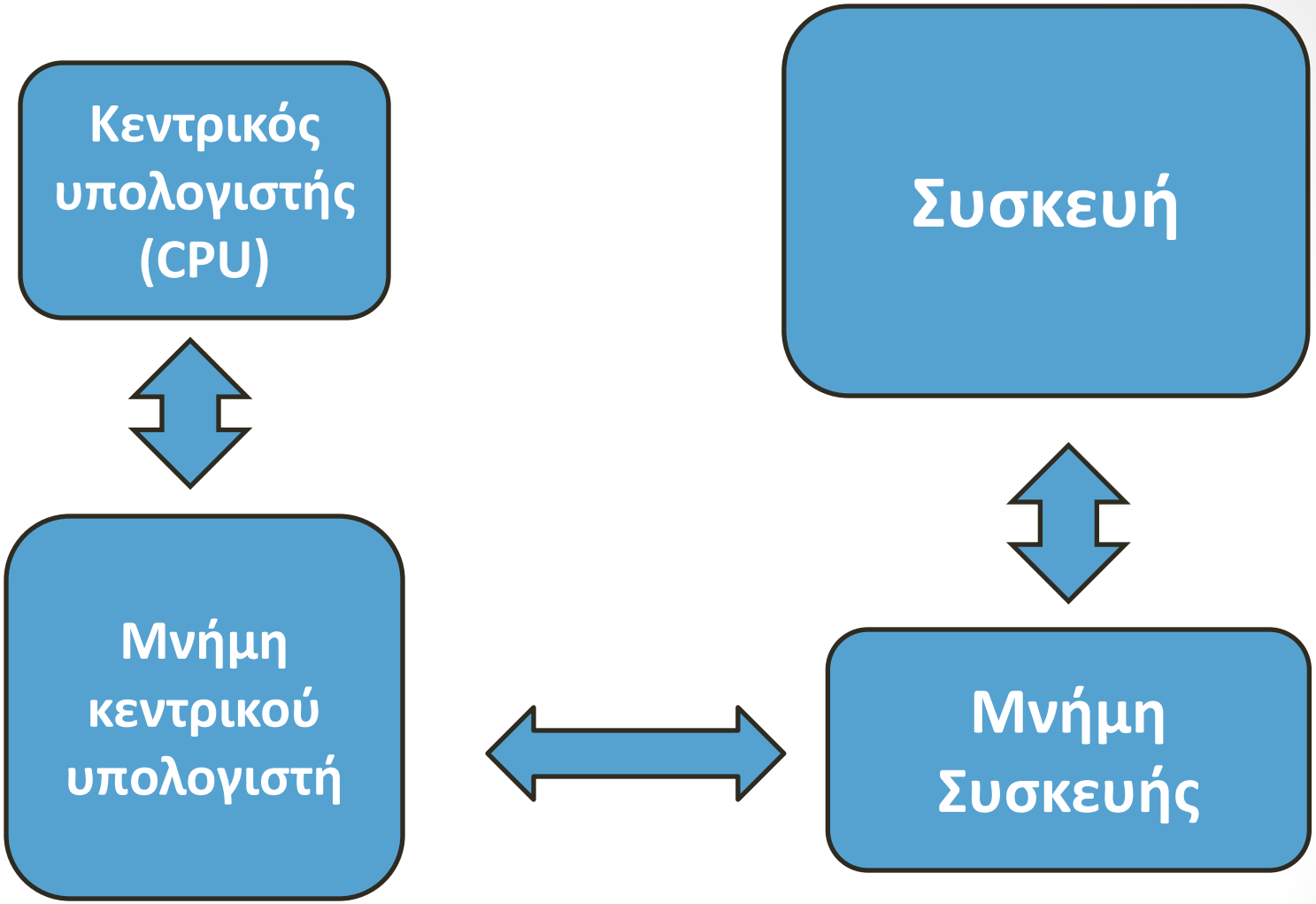
- Στην πραγματικότητα, αυτές οι συσκευές μπορεί να είναι ίδιες ή και διαφορετικές αρχιτεκτονικές, όπως στην περίπτωση όπου host είναι η CPU και επιταχυντής η GPU.

- Οι δύο συσκευές ενδέχεται επίσης να έχουν χωριστούς χώρους μνήμης ή έναν μόνο χώρο μνήμης.

Το μοντέλο του επιταχυντή GPU στο OpenACC

- Στην περίπτωση που οι δύο συσκευές έχουν διαφορετικές μνήμες, ο μεταγλωττιστής OpenACC και ο χρόνος εκτέλεσης θα αναλύσουν τον κώδικα και θα διαχειριστούν τη μνήμη του επιταχυντή και τη μεταφορά δεδομένων μεταξύ της μνήμης του κεντρικού υπολογιστή (CPU) και της συσκευής.

- Το παρακάτω σχήμα δείχνει ένα διάγραμμα υψηλού επιπέδου του αφηρημένου επιταχυντή OpenACC, αλλά θυμηθείτε ότι οι συσκευές και οι μνήμες μπορεί να είναι οι ίδιες σε ορισμένες αρχιτεκτονικές.



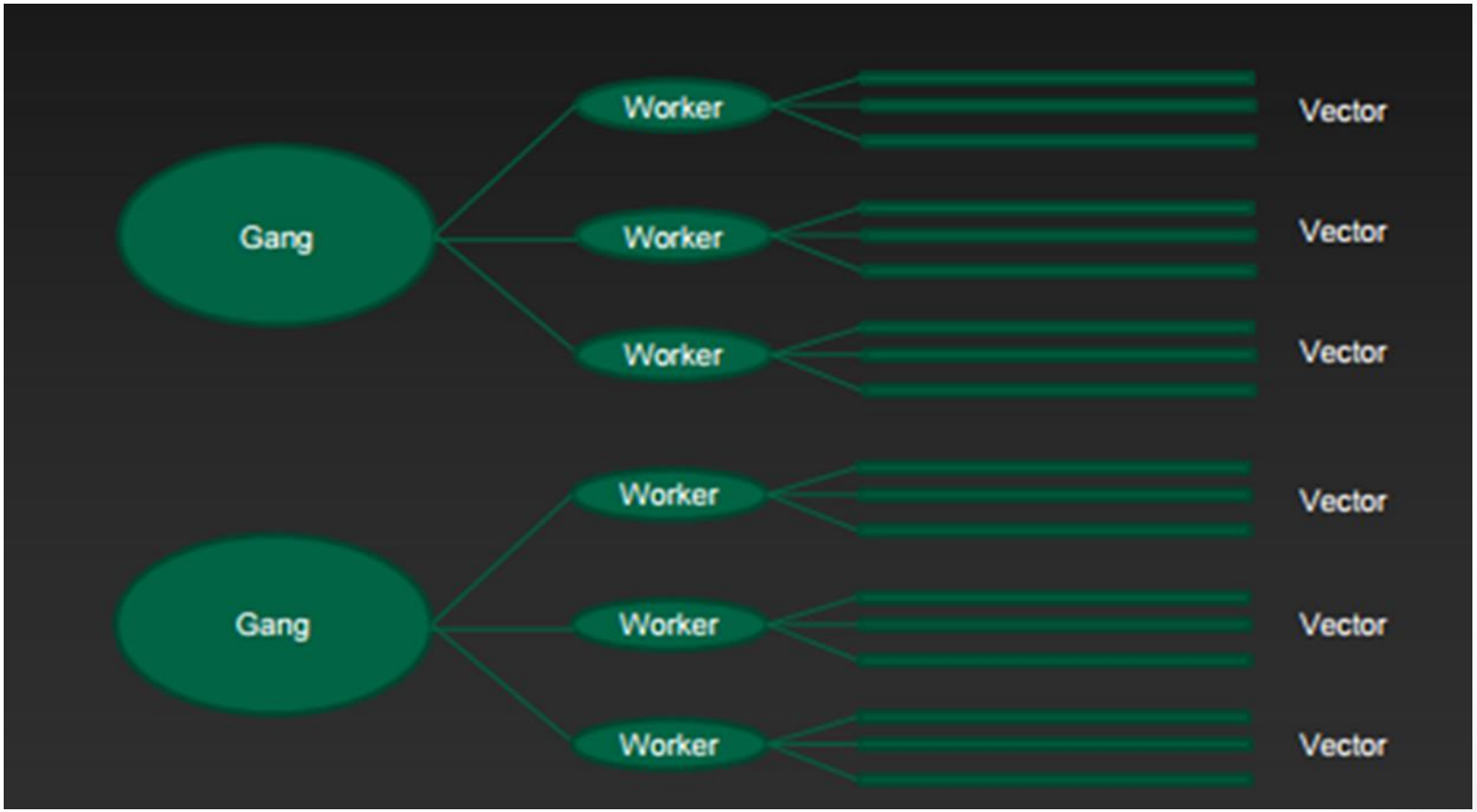
Μοντέλο αφηρημένου επιταχυντή του OpenACC

Υπολογισμός των εργασιών του OpenACC (1/3)

- Ο υπολογισμός της ποσότητας των εργασιών ή των υπολογισμών στο OpenACC έχει τρία επίπεδα: gang, worker και vector.

- Αυτό το μοντέλο υποτίθεται ότι αντιστοιχεί σε οποιαδήποτε αρχιτεκτονική που είναι συλλογή από στοιχεία επεξεργασίας Processing Elements (PEs), όπου κάθε στοιχείο επεξεργασίας (PE) υπόκειται υπερνήματωση και κάθε νήμα εκτελεί τις οδηγίες του vector.

Υπολογισμός των εργασιών του OpenACC (2/3)



Απεικόνιση του υπολογισμού των εργασιών του OpenACC

Υπολογισμός των εργασιών του OpenACC (3/3)

- Πιο συγκεκριμένα έχουμε τον ορισμό του κάθε επιπέδου:
- **gang**: Είναι ένα μπλοκ νημάτων (worker). Όλα τα νήματα σε έναν συνδιασμό (ή διαφορετικά gang) μπορούν να μοιράζονται πόρους, όπως κρυγή μνήμη ή επεξεργαστή.
- **worker:** Είναι αποτελεσματικά ένα warp (νήμα που τρέχει κατά μήκος διασυνδεδεμένων κόμβων που παρομοιάζονται σαν "ύφασμα"). Υπολογίζει ένα νήμα (vector).
- **vector**: Είναι ένα νήμα CUDA και εκτελεί τις επαναλήψεις στην SIMD (**Single instruction, multiple data**) σε ένα μόνο βήμα. Εάν υπάρχουν λιγότερα δεδομένα από το μήκος του διανύσματος, η λειτουργία εξακολουθεί να εκτελείται σε μηδενικές τιμές και το αποτέλεσμα απορρίπτεται.

❑ Σύνταξη για τη C

```
#pragma acc kernels loop gang(n) worker(m) vector(k)
#pragma acc parallel loop num_gangs(n) num_workers(m) vector_length(k)
```

- Kernel: Μια παράλληλη λειτουργία που τρέχει στη GPU.

Στοχεύοντας στην αρχιτεκτονική (1)

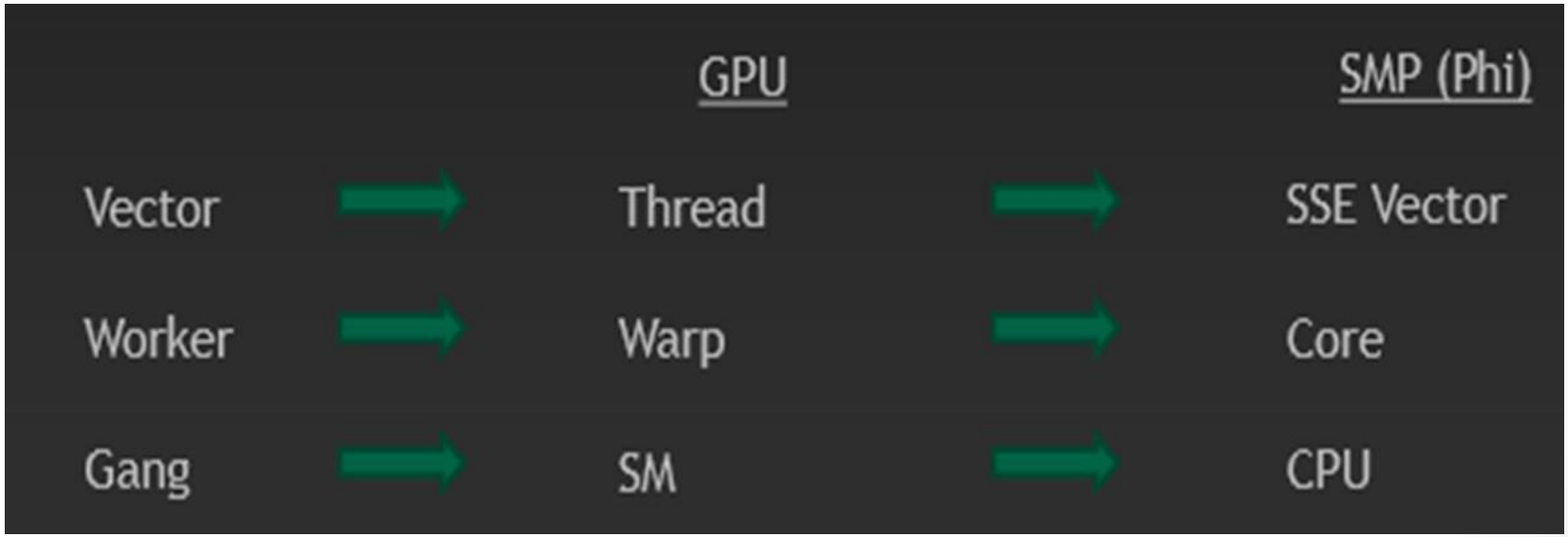
- Όπως προαναφέρθηκε, το OpenACC υποθέτει ότι μια συσκευή θα περιέχει πολλαπλά στοιχεία επεξεργασίας (PE) που εκτελούνται παράλληλα.
- Κάθε στοιχείο επεξεργασίας (PE) έχει επίσης την ικανότητα να εκτελεί αποτελεσματικά διανυσματικές λειτουργίες.

- Για τις GPU της NVIDIA , είναι λογικό να σκεφτόμαστε ένα στοχείο επεξεργασίας (PE) ως ένα πολυεπεξεργαστή ροής (Streaming Multiprocessor-SM).

- Τότε ένα OpenACC gang είναι ένα block νημάτων, ένα worker είναι ένα warp και ένα OpenACC vector (διάνυσμα) είναι ένα νήμα CUDA. Το Phi, ή παρόμοιες αρχιτεκτονικές Intel SMP επίσης χαρτογραφούν με λογικό, αλλά διαφορετικό τρόπο.

Στοχεύοντας στην αρχιτεκτονική (2)

Σχηματικά μπορούμε να δούμε αυτά που προαναφέρθηκαν στην προηγούμενη διαφάνεια:



Τι πρέπει να ξέρουμε για τα warps

- Συγκεριμένα εδώ αναφερόμαστε στα warps της GPU μικροαρχιτεκτονικής -**Kepler**- της NVIDIA.
- Τα block χωρίζονται σε 32 νηματικές ευρείες μονάδες που ονομάζονται warps.
  - Το μέγεθος των warps είναι συγκεκριμένο και μπορεί να αλλάξει μελλοντικά.
- Ο πολυεπεξεργαστής ροής (SM) δημιουργεί, διαχειρίζεται, προγραμματίζει και εκτελεί τα νήματα βάσει τον βαθμό ανάλυσης των warp.
  - Κάθε warp αποτελείται από 32 νήματα συνεχών νημάτων αναγνώρισης (theadIDs).
- Όλα τα νήματα σε ένα warp εκτελούν την ίδια εντολή
  - Εάν τα νήματα ενός warp αποκλίνουν, το warp εκτελεί σειριακά κάθε διαδρομή διακλάδωσης.
- Όταν ένα warp εκτελεί μια εντολή που αποκτά πρόσβαση στην παγκόσμια μνήμη συγχωνεύει τις προσβάσεις μνήμης των νημάτων εντός του warp με όσο το δυνατόν λιγότερες μεταφορές .

Όρια μεγέθους των επιπέδων vector, worker, gang

- Υπενθυμίζουμε ότι αναφερόμαστε πάντα στην μικροαρχιτεκτονική GPU -**Kepler**- της NVIDIA, όπου ισχύουν τα παρακάτω:
- Το μήκος του vector πρέπει να είναι πολλαπλάσιο του 32 (έως 1024)
- Το μέγεθος του gang δίνεται από τον αριθμό των worker μετρώντας το μέγεθος ενός vector. Ο αριθμός αυτός δεν μπορεί να είναι μεγαλύτερος από 1024.
- Ο συνολικός αριθμός των νημάτων σε ένα μπλοκ: μεταξύ 256 και 512 είναι συνήθως είναι ένας καλός αριθμός.
- Για παράδειγμα, το clause worker(32) του vector (32) θα δημιουργούσε 32 worker για να πραγματοποιήσει υπολογισμούς σε διανύσματα (vectors) μεγέθους 32.

Πώς αποφασίζουμε τον αριθμό των blocks (grid)

- Αναφερόμενοι στην μικροαρχιτεκτονική GPU -**Kepler**- της NVIDIA:
- Οι περισσότεροι αρχίζουν με το να έχουν κάθε νήμα να εκτελεί μια μονάδα εργασίας.
- Συνήθως είναι καλύτερα να έχετε λιγότερα νήματα, ώστε κάθε νήμα να εκτελεί πολλαπλά κομμάτια της εργασίας.
- Ποιο είναι το όριο για το πόσο μικρότερο μπορούμε να κάνουμε τον αριθμό των συνολικών μπλοκ;
  - Εξακολουθούμε να θέλουμε να έχουμε τουλάχιστον όσο το δυνατόν περισσότερα νήματα ώστε να γεμίσει η GPU πολλές φορές (π.χ 4 φορές). Αυτό σημαίνει ότι χρειαζόμαστε τουλάχιστον 2880 x 15 x 4 = ~ 173.000 νήματα
  - Πειραματιστείτε μειώνοντας τον αριθμό των νημάτων!

Αντιστοίχιση OpenACC με CUDA threads και blocks (1/2)

- Γνωρίζουμε ότι το OpenACC προορίζεται ως γλώσσα για τους γενικούς επιταχυντές, έτσι δεν υπάρχει άμεση αντιστοίχηση των νημάτων, των blocks και των warps του CUDA.
- Η έκδοση 2.0 του προτύπου OpenACC δηλώνει μόνο ότι gang πρέπει να είναι το εξωτερικό επίπεδο παραλληλισμού, ενώ ο vector πρέπει να είναι το εσωτερικό επίπεδο.
- Στην πραγματικότητα η αντιστοίχιση έχει αφεθεί στον μεταγλωττιστή. Ωστόσο, αν γνωρίζετε ότι μπορεί να υπάρξουν εξαιρέσεις, μπορείτε να σκεφτείτε την εφαρμογή της ακόλουθης αντιστοίχισης:
- OpenACC vector => CUDA threads
- OpenACC worker => CUDA warps
- OpenACC gang => CUDA thread blocks

Αντιστοίχιση OpenACC με CUDA threads και blocks (2/2)

```
#pragma acc kernels
for ( int i = 0; i < n; ++i )
        y[ i ] += a*x[i];
```

16 blocks, 256 νήματα στο καθένα

```
#pragma acc kernels loop gang(100) vector(128)
for ( int i = 0; i<n; ++i )
        y[ i ] += a*x[ i ];
```

100 blocks, στο καθένα 128 νήματα, σε κάθε νήμα εκτελείται μια επανάληψη στο βρόχο

```
#pragma acc parallel num_gangs(100) vector_length (128)
{
        #pragma acc loop gang vector
        for( int i = 0; i < n; ++i ) y[ i ] +=a*x[ i ];
}
```

100 blocks, στο καθένα 128 νήματα, σε κάθε νήμα εκτελείται μια επανάληψη στο βρόχο με χρήση parallel

Τα βασικά στοιχεία του OpenACC

Τι είναι οι οδηγίες (directives) του μεταγλωττιστή;

- Τα directives υποδεικνύουν στον μεταγλωττιστή ή στο χρόνο εκτέλεσης να…..

✓Δημιουργεί τον παράλληλο κώδικα για τη GPU.

✓Κατανέμει τη μνήμη της GPU και αντιγράφει τα δεδομένα εισόδου.

✓Αντιγράφει τα δεδομένα εξόδου στη CPU και απελευθερώνει τμήματα της μνήμης της GPU που είχε δεσμεύσει προηγουμένως.

Σύνταξη οδηγιών (directives)

- **Σε Fortran**

  !$acc directive [clause [,] clause] …]

➢Συνδυάζεται συχνά με μια αντίστοιχη τελική οδηγία που περιβάλλει ένα δομημένο block κώδικα.

  ! $acc end directive

- **Σε C**

  #pragma acc directive [clause [,] clause] …]

➢ Συχνά ακολουθείται από ένα δομημένο block κώδικα.

Σημαντικά directives 1/4

- #pragma acc parallel [clause…]

  {structured block}

  - Καθορίζει μια περιοχή όπου οι επαναλήψεις βρόχων μπορούν να εκτελούνται παράλληλα.
  - Ο μεταγλωττιστής έχει την ελευθερία να το αποσυνθέτει αυτό, ωστόσο θεωρείται επικερδές.
- #pragma acc kernels [clause…]

  {structured block}

  - Παρόμοιο με το parallel, αλλά οι βρόχοι μέσα στην περιοχή των πυρήνων θα είναι ανεξάρτητοι πυρήνες, παρά ένας μεγάλος πυρήνας.
  - Και τα δύο χρησιμοποίουνται για να καθορίσουν την παραλληλοποίηση των πυρήνων που εκτελούνται σε έναν επιταχυντή με τη χρήση ξεχωριστής σημασιολογίας.
  - Οι ανεξάρτητοι πυρήνες και οι συνδεδεμένες μεταφορές δεδομένων ενδέχεται να επικαλύπτονται από άλλους πυρήνες.

Σημαντικά directives 2/3

- #pragma acc data[clause…]

  {structured block}

  - Είναι το κύριο directive που καθορίζει και αντιγράφει τα δεδομένα προς και από τον επιταχυντή.
  - Χρήσιμο για τη μείωση των συνδέσεων PCIe (Peripheral Component Interconnect Express), ένα πρότυπο διαύλου της σειριακής επέκτασης για τη σύνδεση ενός υπολογιστή σε μία ή περισσότερες περιφερειακές συσκευές, δημιουργώντας προσωρινές συστοιχίες ή αφήνοντας δεδομένα στη συσκευή μέχρι να χρειαστεί.

- #pragma acc host_data
  - Καθορίζει μια περιοχή στην οποία θα χρησιμοποιούν συστοιχίες CPU, εκτός αν ορίζεται με use_device()
  - Το use_device αναφέρεται στο δείκτη συσκευής στη CPU.
  - Χρήσιμο για την αλληλεπικάλυψη με υπολογισμό της CPU ή για κλήση βιβλιοθηκών που αναμένουν τη μνήμη συσκευής.

Σημαντικά directives 3/4

- #pragma acc wait
  - Συγχρονίζεται με ασύγχρονες δραστηριότητες.
  - Μπορεί να δηλώσει συγκεκριμένες καταστάσεις ή να περιμένει όλα τα εκκρεμή αιτήματα.

- #pragma acc update [clause…]

  if (expression)

  async (expression)
  - Μετακινεί δεδομένα από τη GPU στον κεντρικό υπολογιστή, ή και αντιστρόφως.
  - Η κίνηση δεδομένων μπορεί να είναι υπό συνθήκη και ασύγχρονη.

Σημαντικά directives 4/4

- #pragma acc loop [clause…]

  {loop}

  - Βελτιστοποιεί τον τρόπο με τον οποίο ο μεταγλωττιστής αντιμετωπίζει συγκεκριμένους βρόχους.
  - Μπορεί να χρησιμοποιηθεί για να καθορίσει την αποσύνθεση της εργασίας.
  - Μπορεί να χρησιμοποιηθεί για την κατάρρευση εμφωλευμένου βρόχου για επιπλέον παραλληλοποίηση.
  - Μπορεί να χρησιμοποιηθεί για να δηλώσει τους πυρήνες (kernels) ως ανεξάρτητους ο ένας από τον άλλον.

Η οδηγία kernels

- Ο μεταγλωττιστής θα διαιρέσει τον κώδικα στην περιοχή πυρήνων σε μια ακολουθία πυρήνων επιταχυντή.

- Τυπικά, κάθε βρόχος θα είναι ένας ξεχωριστός πυρήνας. Όταν το πρόγραμμα εντοπίσει κώδικα με την οδηγία kernels, θα ξεκινήσει την σειριακή αλληλουχία των πυρήνων για τη συσκευή.

- Ο αριθμός και η διαμόρφωση των gangs, των workers και το vector length μπορεί να διαφέρουν για κάθε πυρήνα.

Δηλώσεις της οδηγίας kernels (1/3)

- async *[***(*int-expr*)***]***
  Είναι απαραίτητο για την εκτέλεση του τοπικού νήματος καθώς και τα δεδομένα μπορούν να υποβάλλονται με ασύγχρονη επεξεργασία.
- wait *[***(*int-expr-list*)***]***
  Το τοπικό νήμα περιμένει όλες τις σχετικές ασύγχρονες ουρές δραστηριότητας συσκευών.
- device_type**(** *device-type-list* **)**
  Το πρόγραμμα θα εκχωρήσει ή θα ανακατανομήσει προσωρινά τη μνήμη της συσκευής και θα μεταφέρει δεδομένα προς και / ή από τη συσκευή.
- if**(** *condition* **)**
  Το πρόγραμμα θα κατανέμει υπό συνθήκη τη μνήμη και θα μεταφέρει δεδομένα προς και / ή από τη συσκευή.
- copy**(** *var-list* **)**
- copyin**(** *var-list* **)**
  Χρησιμοποιείται για να δηλώσει ότι οι μεταβλητές, οι πίνακες, ή τα μπλοκ στη var-list έχουν τιμές στην τοπική μνήμη που πρέπει να αντιγραφούν στη μνήμη της συσκευής.
- copyout**(** *var-list* **)**
  Χρησιμοποιείται για να δηλώσει ότι οι μεταβλητές, οι πίνακες, ή τα μπλοκ στον var-list έχουν διατεθεί ή περιέχουν τιμές στη μνήμη της συσκευής που πρέπει να αντιγραφούν στην τοπική μνήμη για ένα επιταχυντή μη κοινόχρηστης μνήμης.

Δηλώσεις της οδηγίας kernels (2/3)

- create**(*var-list*)**

  Χρησιμοποιείται για να δηλώσει ότι οι μεταβλητές, οι πίνακες, ή τα μπλοκ στην var-list πρέπει να κατανεμηθούν (δημιουργηθούν) στη μνήμη της συσκευής για έναν επιταχυντή μη κοινόχρηστης μνήμης.

- present**(*var-list*)**

  Χρησιμοποιείται για να ενημερώσει την εφαρμογή σε μια συσκευή μη κοινόχρηστης μνήμης ότι οι μεταβλητές ή οι πίνακες της var-list παρουσιάζονται ήδη στη μνήμη της συσκευής.

- present_or_copy**(*var-list)***

  Εάν τα δεδομένα είναι ήδη διαθέσιμα, εκτελείται η δήλωση present. Εάν τα δεδομένα δεν υπάρχουν, εκτελείται η δήλωση copy.

- present_or_copyin**(*var-list*)**

  Εάν τα δεδομένα είναι ήδη διαθέσιμαεκτελείται η δήλωση present.Εάν τα δεδομένα δεν υπάρχουν εκτελείται η δήλωση copyin.

Δηλώσεις της οδηγίας kernels (3/3)

- present_or_copyout**(*var-list*)**

  Εάν τα δεδομένα είναι ήδη διαθέσιμα, εκτελείται η δήλωση present. Εάν τα δεδομένα δεν υπάρχουν, εκτελείται η δήλωση copyout.

- present_or_create**(*var-list*)**

  Εάν τα δεδομένα είναι ήδη διαθέσιμα, εκτελείται η δήλωση present. Εάν τα δεδομένα δεν υπάρχουν, εκτελείται η δήλωση create.

- deviceptr**(*var-list*)**

  Χρησιμοποιείται για να δηλώσει ότι οι δείκτες στην var-list είναι δείκτες συσκευής.

- default**(none)**

  Αποτρέπει τον μεταγλωττιστή να προσδιορίσει έμμεσα τα χαρακτηριστικά γνωρίσματα δεδομένων για τις μεταβλητές που αναφέρονται.

Περιορισμοί για την οδηγία kernels

- Ένα πρόγραμμα μπορεί να μην διακλαδώνεται εντός ή εκτός της δομής kernels.
- Το πρόγραμμα δεν πρέπει να εξαρτάται από τη σειρά αξιολόγησης των δηλώσεων ή από τυχόν παρενέργειες των αξιολογήσεων.
- Μόνο οι δηλώσεις async και wait μπορούν να ακολουθήσουν μια δήλωση device_type.
- Μπορεί να εμφανιστεί το πολύ μία δήλωση. Στη C ή C ++, η συνθήκη πρέπει να αξιολογηθεί σε μια ακέραι βαθμωτή τιμή.

Η εφαρμογή της οδηγίας kernels

```
#pragma acc data copyin(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;
#pragma acc kernels loop reduction( max:error ), gang(32), worker(16)
    for( int j = 1; j < n-1; j++) {
#pragma acc loop gang(16), worker(32)
        for(int i = 1; i < m-1; i++) {
            Anew[j][i] = 0.25f * (A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = max(error, abs(Anew[j][i] - A[j][i]);
    }}
#pragma acc kernels loop
    for( int j = 1; j < n-1; j++) {
#pragma acc loop gang(16), worker(32)
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
    }}
    iter++;  }
```

Συνδιαστικά 65% πιο γρήγορο

Στην περίπτωση gang(32), worker(8) -> 30% πιο γρήγορο

Η οδηγία parallel

- Όταν το πρόγραμμα συναντά παράλληλη δομή, δημιουργούνται μία ή περισσότερες gangs για να εκτελέσει την παράλληλη περιοχή του επιταχυντή.
- Ο αριθμός των gangs, ο αριθμός των workers ανά gang και ο αριθμός των vector lanes ανά worker παραμένουν σταθεροί κατά τη διάρκεια αυτής της παράλληλης περιοχής.
- Κάθε gang αρχίζει να εκτελεί τον κώδικα στο δομημένο μπλοκ σε κατάσταση πλεονασμού των gang.
- Αυτό σημαίνει ότι ο κώδικας μέσα στην παράλληλη περιοχή, αλλά έξω από έναν βρόχο με μια οδηγία loop και τον διαμοιρασμό εργασιών σε επίπεδο των gangs, θα εκτελεστεί με πλεονασμό από όλα τα gangs.

Δηλώσεις της οδηγίας parallel

• num_gangs (expression)	Ελέχγει πόσα gangs έχουν δημιουργηθεί(CUDA gridDim)
• num_workers (expression)	Ελέγχει πόσα workers έχουν δημιουργηθεί σε κάθε gang (CUDA blockDim)
• vector_length (list)	Ελέχγει το μήκος του vector του κάθε worker (SIMD εκτέλεση)
• private (list)	Κατανέμει ένα αντίγραφο της κάθε μεταβλητής σε λίστα για κάθε gang
• firstprivate (list)	Μεταβλητές private που αρχικοποιούνται από τον κεντρικό υπολογιστή
• reduction (operator: list)	Μεταβλητές private σε συνδυασμό με τα gangs

Περιλαμβάνονται και οι δηλώσεις που είδαμε προηγουμένως: async, wait, device_type, if, copy, copyin, copyout, create, present, present_or_copy, present_or_copyin, present_or_copyout, present_or_create, deviceptr και default

Περιορισμοί για την οδηγίας parallel

- Ένα πρόγραμμα μπορεί να μην διακλαδώνεται εντός ή εκτός της parallel δομής.
- Το πρόγραμμα δεν πρέπει να εξαρτάται από τη σειρά αξιολόγησης των δηλώσεων ή από τυχόν παρενέργειες των αξιολογήσεων.
- Μόνο οι δηλώσεις async, wait, num_gangs, num_workers και vector_length ενδέχεται να ακολουθούν μια δήλωση device_type.
- Μπορεί να εμφανιστεί το πολύ μία δήλωση. Στη C ή C ++, η συνθήκη πρέπει να αξιολογηθεί σε μια βαθμωτή ακέραια τιμή.

Περισσότερη παραλληλοποίηση

```
#pragma acc data copyin(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;
#pragma acc parallel reduction( max:error )
    for( int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {
            Anew[j][i] = 0.25f * (A[j][i+1] + A[j][i-1]
                                + A[j-1][i] + A[j+1][i]);
            error = max(error, abs(Anew[j][i] - A[j][i]);
        }
    }

#pragma acc parallel
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
    }}
    iter++;  }
```

Βρίσκει το μέγιστο σε όλες τις επαναλήψεις

Η οδηγία data

- Καθορίζει βαθμωτές μεταβλητές, πίνακες και υποπίνακες που θα διατεθούν στη μνήμη της συσκευής για τη διάρκεια της εκτέλεσης του μπλοκ κώδικα της οδηγίας data.

- Εάν, δηλαδή, τα δεδομένα πρέπει να αντιγραφούν από τον κεντρικό υπολογιστή στη μνήμη της συσκευής κατά την έναρξη του κώδικα της οδηγίας και να αντιγραφούν από τη συσκευή στη μνήμη υποδοχής κατά την έξοδο από το μπλόκ του κώδικα.

Δηλώσεις της οδηγίας data

- Οι δηλώσεις της οδηγίας data είναι οι παρακάτω:

- **copyin (list)** - Καταχωρεί μνήμη στη GPU και αντιγράφει δεδομένα από τον κεντρικό υπολογιστή σε GPU κατά την είσοδο στην περιοχή.
- **copyout (list)** - Καταχωρεί μνήμη στη GPU και αντιγράφει δεδομένα στον κεντρικό υπολογιστή όταν εξέρχεται από την περιοχή.
- **copy (list)** - Καταχωρεί τη μνήμη στη GPU και αντιγράφει δεδομένα από τον κεντρικό υπολογιστή στη GPU κατά την είσοδο στην περιοχή και αντιγράφει τα δεδομένα στον κεντρικό υπολογιστή όταν εξέρχεται από την περιοχή. (Μόνο δομημένα δεδομένα)
- **create (list)** - Καταχωρεί τη μνήμη στη GPU αλλά δεν αντιγράφει.
- **delete (list)** - Διαγραφή μνήμης στη GPU χωρίς αντιγραφή. (Μόνο αδόμητα δεδομένα)
- **present (list)** - Τα δεδομένα βρίσκονται ήδη στη GPU από άλλη περιοχή δεδομένων που τα περέχει.

Διαμόρφωση Πινάκων στη δήλωση της οδηγίας data

- Οι μεταγλωττιστές μερικές φορές δεν μπορούν να καθορίσουν το μέγεθος των πινάκων οπότε πρέπει να καθορίσουμε ρητά χρησιμοποιώντας δηλώσεις δεδομένων με ένα πίνακα “shape.
- Ο μεταγλωττιστής θα σας ενημερώσει εάν πρέπει να το κάνετε αυτό. Μερικές φορές, εσείς θα θέλετε για τους δικούς σας λόγους απόδοσης.
- Σε C και C ++, ένας υποπίνακας είναι ένα όνομα πίνακα ακολουθούμενο από τον προσδιορισμό της απόστασης σε αγκύλες, με αρχή και μήκος, όπως:

AA[2:n]

- Εάν λείπει το κατώτερο όριο, χρησιμοποιείται μηδέν.
- Εάν το μήκος λείπει και ο πίνακας έχει γνωστό μέγεθος, χρησιμοποιείται το μέγεθος του πίνακα, διαφορετικά το μήκος είναι απαραίτητο. Ο υποπίνακας AA[2:n] σημαίνει ότι αποτελείται από τα στοιχεία **AA[2]**, **AA[3]**, …, **AA[2+n-1]**.

Προδιαγραφές δεδομένων στη δήλωση της οδηγίας data (1/2)

- Στις C και C ++, ένας πίνακας δύο διαστάσεων μπορεί να δηλωθεί με τουλάχιστον τέσσερις τρόπους:

1. Πίνακας με στατικό μέγεθος : **float AA[100][200];**
2. Δείκτης σε σειρά στατικού μεγέθους: **typedef float row[200]; row* BB;**
3. Πίνακας δεικτών με στατικό μέγεθος: **float* CC[200];**
4. Δείκτης σε δείκτες: **float** DD;**

Προδιαγραφές δεδομένων στη δήλωση της οδηγίας data (1/2)

- Κάθε διάσταση μπορεί να έχει στατικό μέγεθος ή ένας δείκτης μπορεί να κατανέμεται δυναμικά στη μνήμη. Καθένα από αυτά μπορεί να συμπεριληφθεί σε μια δήλωση δεδομένων χρησιμοποιώντας υποσημείωση για να καθορισμό ενός ορθογώνιου πίνακα:
- **AA[2:n][0:200]**
- **BB[2:n][0:m]**
- **CC[2:n][0:m]**
- **DD[2:n][0:m]**
- Μπορούν να καθοριστούν πολυδιάστατα ορθογώνια υποσυστήματα σε C και C ++ για οποιαδήποτε διάταξη με οποιονδήποτε συνδυασμό διαστάσεων στατικού μεγέθους ή δυναμικά κατανεμημένων διαστάσεων.
- Για διαστάσεις στατικού μεγέθους, όλες οι διαστάσεις εκτός από την πρώτη πρέπει να καθορίζουν ολόκληρη τη διάσταση, ώστε να διατηρούνται οι περιορισμοί των συνεχόμενων δεδομένων που περιγράφονται παρακάτω.
- Για δυναμικά κατανεμημένες διαστάσεις, η υλοποίηση θα κατανείμει στη συσκευή τους δείκτες που αντιστοιχούν στους δείκτες του κεντρικού υπολογιστή και θα τους συμπληρώσει ανάλογα με την περίπτωση.

Περιορισμοί των δεδομένων στη δήλωση της οδηγίας data (2/2)

- Στις C και C ++, το μήκος για δυναμικά κατανεμημένες διαστάσεις ενός πίνακα πρέπει να καθορίζεται άμεσα.
- Σε C και C ++, κατά τη διάρκεια της ζωής των δεδομένων, είτε στον κεντρικό υπολογιστή είτε στη συσκευή η τροποποίηση των δεικτών σε πίνακες δεικτών μπορεί να οδηγήσει σε απροσδιόριστη συμπεριφορά.
- Εάν ένα υποσύνολο έχει οριστεί σε μια δήλωση δεδομένων, η υλοποίηση μπορεί να επιλέξει να διαθέσει μνήμη μόνο για εκείνον τον υποπίνακα στον επιταχυντή.
- Οποιοσδήποτε πίνακας ή υποπίνακας σε μια δήλωση δεδομένων, πρέπει να είναι ένα συνεχόμενο μπλοκ μνήμης, εκτός από τους δυναμικούς πολυδιάστατους πίνακες C.
- Όλα τα στοιχεία δεδομένων της δομής ή της κλάσης κατανέμονται και αντιγράφονται εάν καθορίζεται μια μεταβλητή ή πίνακας τύπου struct ή class, ανάλογα με την περίπτωση. Εάν ένα μέλος struct ή class είναι ένας δείκτης, τα δεδομένα που διευθυνσιοδοτούνται από αυτόν τον δείκτη δεν αντιγράφονται έμμεσα.

Βελτιωμένη διαδικασία εκτέλεσης της GPU στο OpenACC

```
#pragma acc data copyin(A), create(Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;
#pragma acc parallel
    for( int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {
            Anew[j][i] = 0.25f * (A[j][i+1] + A[j][i-1]
                                  + A[j-1][i] + A[j+1][i]);
            error = max(error, abs(Anew[j][i] - A[j][i]);
        }
    }

#pragma acc parallel
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
    }}
    iter++;  }
```

Η αντιγραφή του πίνακα A είναι πολύ πιο αποδοτική και η κίνηση δεδομένων για τον πίνακα Anew έχει εξαλειφθεί.

Η οδηγία loop

- Εφαρμόζεται σε έναν βρόχο ο οποίος πρέπει να ακολουθήσει αμέσως αυτή την οδηγία.

- Η οδηγία βρόχου μπορεί να περιγράψει τον τύπο παραλληλισμού που πρέπει να χρησιμοποιηθεί για την εκτέλεση του βρόχου και να δηλώσει τις ιδιωτικές μεταβλητές του βρόχου και τους πίνακες καθώς και τις λειτουργίες της δήλωσης reduction.

Δηλώσεις της οδηγίας loop (1/7)

- collapse (*n*)

  Χρησιμοποιείται για να καθορίσει πόσοι εμφωλευμένοι βρόχοι συνδέονται με τη δομή του βρόχου. Το όρισμα n πρέπει να είναι μια σταθερή θετική ακέραια έκφραση.

- seq

  Καθορίζει ότι ο συνδεδεμένος βρόχος ή βρόχοι πρέπει να εκτελούνται διαδοχικά από τον επιταχυντή.

- private (var- *list*)

  Δημιουργεί αντίγραφο για κάθε μεταβλητή στη var-list, για κάθε νήμα που εκτελεί μία ή πεισσότερες επανάληψεις του σχετικού βρόχου ή βρόχων.

- independent

  Καταδεικνύει ότι οι επαναλήψεις αυτού του βρόχου είναι ανεξάρτητες των δεδομένων. Αυτό επιτρέπει στην υλοποίηση να δημιουργήσει κώδικα για να εκτελέσει τις επαναλήψεις παράλληλα χωρίς συγχρονισμό. Σε μια δομή parallel η δήλωση independent εμπεριέχεται σε όλες τις οδηγίες βρόχων χωρίς μια δήλωση seq.

- device_type**(** *device-type-list* **)**

  Οι οδηγίες OpenACC μπορούν να καθορίσουν διαφορετικές δηλώσεις ή ορίσματα δηλώσεων για διαφορετικούς επιταχυντές χρησιμοποιώντας τη δήλωση device_type. Το όρισμα στη ρήτρα device_type είναι μια λίστα διαχωρισμένη με κόμμα ενός ή περισσότερων ονομαστικών αναγνωριστικών της αρχιτεκτονικής του επιταχυντή ή ένας αστερίσκος. Μία οδηγία μπορεί να έχει μία ή περισσότερες ρήτρες device_type.

Δηλώσεις της οδηγίας loop (2/7)

- gang ***[(*** *gang-arg-list* ***)]*****, *όπου*** *gang-arg-list χρησιμοποιείται ένα από τα παρακάτω:*
- ***[num:]*** *int-expr ,* **static:** *size-expr*

- Σε μια δομή parallel ορίζει ότι οι επαναλήψεις του σχετικού βρόχου ή βρόχων πρέπει να εκτελούνται παράλληλα με τη κατανομή των επαναλήψεων μεταξύ των gang που δημιουργούνται από τη δομή. Μια δομή loop με τη δήλωση gang μετατρέπει μια περιοχή υπολογισμών από κατάσταση πλεονασμού των gang σε κατάσταση διαμοιρασμού των gang.
- Σε μια δομή kernels, ορίζει ότι οι επαναλήψεις του σχετικού βρόχου ή βρόχων πρέπει να εκτελούνται παράλληλα σε όλα τα gang που δημιουργούνται για κάθε kernel που περιέχεται μέσα στον βρόχο ή τους βρόχους. Αν ένα όρισμα έχει καθοριστεί χωρίς λέξη-κλειδί ή παράμετρο μετά την λέξη num, τότε προσδιορίζει πόσα gang θα χρησιμοποιηθούν για να εκτελεστούν οι επαναλήψεις αυτού του βρόχου.

Δηλώσεις της οδηγίας loop (3/7)

- worker ***[([num:]*** *int-expr* ***)]***

- Σε μια parallel δομή ορίζει ότι οι επαναλήψεις του σχετικού βρόχου ή βρόχων πρέπει να εκτελούνται παράλληλα με τη διανομή των επαναλήψεων μεταξύ των πολλαπλών worker μέσα σε ένα ενιαίο gang. Μια δομή loop με μια δήλωση worker αναγκάζει μια gang να μεταβεί από την κατάσταση απλής λειτουργίας ενός worker σε κατάσταση διαμερισμού του worker.
- Σε αντίθεση με τη δήλωση gang, η δήλωση worker ενεργοποιεί πρώτα τον παραλληλισμό στο επιπέδο worker και στη συνέχεια διανέμει τις επαναλήψεις των βρόχων μεταξύ των workers. Δεν επιτρέπεται κανένα όρισμα. Οι επαναλήψεις βρόχου πρέπει να είναι ανεξάρτητες από δεδομένα, εκτός από τις μεταβλητές που καθορίζονται σε δήλωση reduction .
- Σε μια δομή kernels ορίζει ότι οι επαναλήψεις του συνδεδεμένου βρόχου ή βρόχων πρέπει να εκτελούνται παράλληλα στους workers μέσα σε ένα gang που δημιουργήθηκε για κάθε kernel που περιέχεται μέσα στον βρόχο ή τους βρόχους. Αν υπάρχει ένα όρισμα, καθορίζει τον αριθμό των worker ανά gang για να εκτελέσει τις επαναλήψεις αυτού του βρόχου.

Δηλώσεις της οδηγίας loop (4/7)

- vector ***[([*length:*] *int-expr*)]***

- Σε μια δομή parallel καθορίζει ότι οι επαναλήψεις του συνδεδεμένου βρόχου ή βρόχων πρέπει να εκτελούνται σε διανυσματική ή SIMD λειτουργία. Μια δομή loop με δήλωση vector προκαλεί τη μετάβαση ενός vector από κατάσταση απλής λειτουργίας σε κατάσταση διαμερισμού του vector.
- Παρόμοια με τη δήλωση worker, η δήλωση vector ενεργοποιεί πρώτα τον παραλληλισμό σε επιπέδο vector και στη συνέχεια διανέμει τις επαναλήψεις βρόχου μεταξύ αυτών των γραμμών (lanes) του vector. Οι πράξεις θα εκτελούνται χρησιμοποιώντας διανύσματα του καθορισμένου ή επιλεγμένου μήκους για την παράλληλη περιοχή. Η δομή loop με δήλωση vector δεν μπορεί να περιέχει βρόχο με τη δήλωση gang, worker ή vector εκτός αν βρίσκεται μέσα σε μια φωλιασμένη παράλληλη περιοχή ή δομή parallel.
- Σε μια δομή kernels, η δήλωση vector καθορίζει ότι οι επαναλήψεις του συνδεδεμένου βρόχου ή βρόχων πρέπει να εκτελούνται με vector ή SIMD.επεξεργασία. Αν καθορίζεται ένα όρισμα, οι επαναλήψεις θα υποστούν επεξεργασία σελουριδες διανύσματος αυτού του μήκους, διαφορετικά η υλοποίηση θα επιλέξει ένα κατάλληλο μήκος διανύσματος (vector length).
- Όλες οι γραμμές vector θα ολοκληρώσουν την εκτέλεση των ανατεθειμένων επαναλήψεών τους προτού αποχωρήσει οποιαδήποτε γραμμή vector πέρα από το τέλος του βρόχου.

Δηλώσεις της οδηγίας loop (5/7)

- auto

- Ορίζει ότι η υλοποίηση πρέπει να επιλέξει αν θα εφαρμόσει παραλληλισμό gang, worker ή vector σε αυτόν τον βρόχο. Υπάρχει περίπτωση η υλοποίηση να περιοριστεί με τους τύπους παραλληλισμού που μπορεί να εφαρμόσει λόγω των οδηγιών loop με τις δηλώσεις gang, worker ή vector για εξωτερικούς ή εσωτερικούς βρόχους.
- Αυτή η δήλωση δεν λέει από μόνη της στην εφαρμογή ότι οι επαναλήψεις του βρόχου είναι ανεξάρτητες των δεδομένων, με αποτέλεσμα η υλοποίηση να μην μπορεί να εφαρμόσει παραλληλισμό εκτός εάν ο βρόχος έχει τη δήλωση auto.
- Είναι έμμεσα ανεξάρτητη είτε επειδή είναι σε δομή parallel ή η υλοποίηση μπορεί να αναλύσει τον βρόχο και να προσδιορίσει ότι οι επαναλήψεις του βρόχου είναι ανεξάρτητες των δεδομένων.
- Σε μια δομή kernels, μια οδηγία loop χωρίς gang, worker, vector ή δήλωση seq αντιμετωπίζεται σαν να έχει την δήλωση auto.

Δηλώσεις της οδηγίας loop (6/7)

- reduction (operator: var-*list)*

- Καθορίζει έναν τελεστή μείωσης και μία ή περισσότερες βαθμωτές μεταβλητές. Για κάθε μεταβλητή μείωσης, δημιουργείται ένα ιδιωτικό αντίγραφο για κάθε νήμα που εκτελεί επαναλήψεις του σχετικού βρόχου ή βρόχων και αρχικοποιείται για αυτόν τον τελεστή. Στο τέλος του βρόχου, οι τιμές για κάθε νήμα συνδυάζονται χρησιμοποιώντας τον καθορισμένο τελεστή μείωσης και το αποτέλεσμα αποθηκεύεται στην αρχική μεταβλητή στο τέλος του μπλοκ κώδικα parallel ή kernels.
- Σε μια παράλληλη περιοχή, εάν η δήλωση reduction χρησιμοποιείται σε έναν βρόχο με τις δηλώσεις vector ή worker (και καμία δήλωση gang) και η βαθμωτή μεταβλητή εμφανίζεται επίσης σε μια δήλωση private στη δομή parallel, η αξία του ιδιωτικού αντιγράφου της βαθμωτής θα ενημερωθεί κατά την έξοδο του βρόχου. Εάν η βαθμωτή μεταβλητή δεν εμφανίζεται σε μια δήλωση private στην δομή parallel ή εάν η δήλωση reduction χρησιμοποιείται σε βρόχο με τη δήλωση gang, η τιμή της δεν θα ενημερωθεί μέχρι το τέλος της δομής parallel.

Δηλώσεις της οδηγίας loop (7/7)

- tile **(** *size-expr-list* **), όπου** size-expr-list χρησιμοποιείται ένα από τα παρακάτω: ***,** *int-expr*

- Ορίζει ότι η υλοποίηση θα πρέπει να διαιρεί κάθε βρόχο σε δύο βρόχους, με ένα εξωτερικό σύνολο βρόχων πλακιδίων και ένα εσωτερικό σύνολο στοιχείων βρόχων. Το όρισμα που λαμβάνει είναι ένας κατάλογος με ένα ή περισσότερα μεγέθη πλακιδίων, όπου κάθε μέγεθος πλακιδίων είναι μια σταθερή θετική ακέραιη έκφραση ή ένας αστερίσκος. Εάν υπάρχουν n μεγέθη πλακιδίων στη λίστα, η οδηγία loop πρέπει να ακολουθείται αμέσως από n φωλιασμένους βρόχους. Οι βρόχοι πλακιδίων θα αναδιαταχθούν ώστε να είναι έξω από όλους τους βρόχους στοιχείων και όλοι οι βρόχοι στοιχείων θα βρίσκονται μέσα στους βρόχους πλακιδίων.
- Εάν η δήλωση vector εμφανίζεται στην οδηγία loop, η δήλωση vector εφαρμόζεται στους βρόχους στοιχείων (element loops).
- Εάν η δήλωση gang εμφανίζεται στην οδηγία loop, εφαρμόζεται η δήλωση gang στους βρόχους πλακιδίων (tile loops).
- Εάν η δήλωση worker εμφανίζεται στην οδηγία loop, εφαρμόζεται η δήλωση worker στους βρόχους στοιχείου αν δεν υπάρχει δήλωση vector και στους βρόχους πλακιδίων σε διαφορετική περίπτωση.

Άλλες οδηγίες

- cache construct

  Δεδομένα προσωρινής μνήμης του λογισμικού διαχειρίζονται τα δεδομένα κρυφής μνήμης CUDA (κοινή μνήμη CUDA).

- host_data construct

  Διαθέτει στο host την διεύθυνση των δεδομένων της συσκευής.

- wait directive

  Περιμένει την ολοκλήρωση της ασύγχρονης δραστηριότητας της GPU.

- declare directive

  Καθορίζει ότι τα δεδομένα πρόκειται να κατανεμηθούν στην μνήμη της συσκευής για τη διάρκεια μιας περιοχής έμμεσων δεδομένων, που δημιουργήθηκε κατά την εκτέλεση ενός υποπρογράμματος.

Παραλληλοποίηση βρόχων

Παραλληλοποίηση βρόχων (1/3)

- Δεδομένου ότι οι εφαρμογές έχουν αναγνωριστεί, οι προγραμματιστές θα επιταχύνουν αυστηρά αυτά τα σημεία με την προσθήκη directives του OpenACC στους σημαντικούς βρόχους μέσα σε αυτές τις ρουτίνες.
- Δεν υπάρχει λόγος να σκεφτούμε την μεταφορά δεδομένων σε αυτό το σημείο της διαδικασίας, ο μεταγλωττιστής OpenACC θα αναλύσει τα δεδομένα που χρειάζονται στην αναγνωρισμένη περιοχή και θα διασφαλίσει αυτόματα ότι τα δεδομένα είναι διαθέσιμα στον επιταχυντή.

Παραλληλοποίηση βρόχων (2/3)

- Εστιάζοντας αποκλειστικά στην παραλληλοποίηση κατά τη διάρκεια αυτού του βήματος, ο προγραμματιστής μπορεί να μετακινήσει όσο το δυνατόν περισσότερους υπολογισμούς στη συσκευή και να διασφαλίσει ότι το πρόγραμμα εξακολουθεί να δίνει σωστά αποτελέσματα προτού βελτιστοποιήσει την κίνηση των δεδομένων στο επόμενο βήμα.
- Κατά τη διάρκεια αυτού του βήματος της διαδικασίας είναι σύνηθες να αυξάνεται ο συνολικός χρόνος εκτέλεσης του κώδικα, ακόμη και αν η εκτέλεση των επιμέρους βρόχων είναι ταχύτερη με τη χρήση του επιταχυντή.

Παραλληλοποίηση βρόχων (3/3)

- Αυτό οφείλεται στο γεγονός ότι ο μεταγλωττιστής πρέπει να προσεγγίσει προσεκτικά την κίνηση των δεδομένων, συχνά αντιγράφοντας περισσότερα δεδομένα προς και από τον επιταχυντή από ότι είναι πραγματικά απαραίτητο.
- Ακόμη και αν ο συνολικός χρόνος εκτέλεσης αυξάνεται κατά τη διάρκεια αυτού του βήματος, ο δημιουργός του έργου θα πρέπει να επικεντρωθεί στην παραλληλοποίηση στον κώδικα πριν προχωρήσει στο επόμενο βήμα και επωφεληθεί από τα directives.

Παραδείγματα της δομής των οδηγιών kernels, parallel και atomic

Η δομή της οδηγίας kernels (1/2)

- Το directive kernels αναγνωρίζει μια περιοχή κώδικα που μπορεί να περιέχει παραλληλοποίηση, αλλά στηρίζεται στις αυτόματες δυνατότητες παραλληλοποίησης του μεταγλωττιστή για να αναλύσει την περιοχή, να προσδιορίσει ποιοι βρόχοι είναι ασφαλείς για να παραλληλοποιηθούν και στη συνέχεια να επιταχυνθούν.
- Οι προγραμματιστές θα είναι σε θέση να δοκιμάσουν μια άλλη εμπειρία προγραμματισμού ή να εργάζονται σε λειτουργίες που περιέχουν εμφωλευμένους βρόχους που θα μπορούσαν να παραλληλοποιηθούν, αποτελεί μια καλή αρχή για την επιτάχυνση του OpenACC με τη χρήση του directive kernels.

Η δομή της οδηγίας kernels (2/3)

- Ο παρακάτω κώδικας καταδεικνύει τη χρήση του σε γλώσσα προγραμματισμού C:

```
#pragma acc kernels
{
        for (i=0; i<N; i++)
        {
                y[ i ] = 0.0f;
                x[ i ] = (float)(i+1);
        }
        for (i=0; i<N; i++)
        {
           y [ i ] = 2.0f  *  x[ i ] + y[ i ];
        }
}
```

- Σε αυτό το παράδειγμα ο κώδικας αρχικοποιεί δύο πίνακες και στη συνέχεια εκτελεί έναν απλό υπολογισμό με αυτούς.

Η δομή της οδηγίας kernels (3/3)

- Ο μεταγλωττιστής θα αναλύσει αυτούς τους βρόχους για την ανεξαρτησία των δεδομένων και θα παραλληλισθεί και στους δύο βρόχους, δημιουργώντας έναν επιταχυντή kernel για κάθε loop.
- Ο μεταγλωττιστής έχει πλήρη ελευθερία να καθορίσει τον καλύτερο τρόπο να χαρτογραφήσει την παραλληλοποίηση που διατίθεται σε αυτούς τους βρόχους , πράγμα που σημαίνει ότι θα μπορέσουμε να χρησιμοποιήσουμε τον ίδιο κώδικα ανεξάρτητα από τον επιταχυντή για τον οποίο κατασκευάζουμε.
- Ο μεταγλωττιστής θα χρησιμοποιήσει τις δικές του γνώσεις του επιταχυντή στόχου για να επιλέξει την καλύτερη διαδρομή επιτάχυνσης. Εάν ο μεταγλωττιστής δεν είναι σίγουρος ότι ένας βρόχος είναι ανεξάρτητος των δεδομένων, ο βρόχος δεν θα παραλληλοποιηθεί.

Η δομή της οδηγίας parallel (1/4)

- Η δομή του parallel αναγνωρίζει μια περιοχή κώδικα που θα παραλληλοποιηθεί σε όλα τα gangs του OpenACC.
- Από μόνη της μια παράλληλη περιοχή έχει περιορισμένη χρήση, αλλά όταν συνδυάζεται με την οδηγία loop ο μεταγλωττιστής θα παράγει μια παράλληλη έκδοση του βρόχου για τον επιταχυντή.
- Αυτά τα δύο directives μπορούν και πιο συχνά, να συνδυαστούν σε μια οδηγία parallel loop.
- Με την τοποθέτηση αυτής της οδηγίας σε βρόχο, ο προγραμματιστής βεβαιώνει ότι ο βρόχος που έχει προκύψει είναι ασφαλής για να παραλληλοποιηθεί και επιτρέπει στον μεταγλωττιστή να επιλέξει τον τρόπο προγραμματισμού των επαναλήψεων του βρόχου στον επιταχυντή στόχου.

Η δομή της οδηγίας parallel (2/4)

- Ο παρακάτω κώδικας καταδεικνύει τη χρήση του συνδυασμού των directives parallel loop σε C:

```
#pragma acc parallel loop
        for (i=0; i<N; i++)
        {
                y[ i ] = 0.0f;
                x[ i ] = (float)(i+1);
        }

#pragma acc parallel loop
        for (i=0; i<N; i++)
        {
                y[ i ]=2.0f * x[ i ] + y[ i ];

        }
```

Η δομή της οδηγίας parallel (3/4)

- Παρατηρήστε ότι, σε αντίθεση με το directive kernels, κάθε βρόχος πρέπει να είναι ρητά σχεδιασμένος με directives τύπου parallel loop .
- Αυτό οφείλεται στο γεγονός ότι η δομή parallel βασίζεται στον προγραμματιστή για να προσδιορίσει την παραλληλοποίηση στον κώδικα αντί στο μεταγλωττιστή για να εκτελέσει τη δική του ανάλυση των βρόχων.
- Στην περίπτωση αυτή, ο προγραμματιστής αναγνωρίζει μόνο τη διαθεσιμότητα της παραλληλοποίησης, αλλά αφήνει την απόφαση για το πώς να χαρτογραφήσει την παραλληλοποίηση με τον επιταχυντή βασιζόμενος στις γνώσεις του μεταγλωττιστή σχετικά με τη συσκευή.
- Αυτό είναι ένα βασικό χαρακτηριστικό που διαφοροποιεί το OpenACC από άλλα, παρόμοια μοντέλα προγραμματισμού.

Η δομή της οδηγίας parallel (4/4)

- Ο προγραμματιστής αναγνωρίζει την παραλληλοποίηση χωρίς να υπαγορεύει στον μεταγλωττιστή το πώς να την εκμεταλλευτεί.
- Αυτό σημαίνει ότι ο κώδικας OpenACC μπορεί να είναι φορητός σε συσκευές διαφορετικές από τη συσκευή στην οποία αναπτύσσεται ο κώδικας.
- Η δομή του directive loop δίνει στον μεταγλωττιστή πρόσθετες πληροφορίες για τον επόμενο βρόχο στον πηγαίο κώδικα.
- Το directive loop παρουσιάστηκε παραπάνω σε συνδυασμό με το directive parallel, και είναι επίσης έγκυρο με το directive kernels.

Η δομή της οδηγίας atomic (1/4)

#pragma acc atomic **[** *atomic-clause* **]** new-line

expression-stmt

ή

#pragma acc atomic capture *new-line*

structured-block

- *Όπου atomic-clause είναι read, write, update ή capture.*
- Η *expression-stmt* είναι μια δήλωση έκφρασης με μία από τις ακόλουθες μορφές:
- Εάν atomic-clause είναι **read**, το οποίο εξασφαλίζει ότι δύο επαναλήψεις βρόχουδεν θα διαβάζονται ταυτόχρονα από την περιοχή:

  v = x;
- Αν η atomic-clause είναι **write**, το οποίο θα διασφαλίσει ότι δεν υπάρχουν δύο επαναλήψεις με εγγραφή στην περιοχή ταυτόχρονα:

  x = expr;

Η δομή της οδηγίας atomic (2/4)

- Εάν η atomic-clause είναι **update** (συνδυασμός read και write που προαναφέρθηκαν) ή δεν εμφανίζεται:

x ++

x--;

++ x;

--x;

x binop = expr;

x = x binop expr;

x = expr binop x.

- Αν η atomic-clause είναι **capture**, το οποίο πραγματοποιεί ότι και το update, αλλά αποθηκεύει την τιμή που υπολογίζεται σε εκείνη την περιοχή για να χρησιμοποιηθεί στον κώδικα που ακολουθεί. Εάν δεν δίνεται κα΄μία δήλωση τότε συμβαίνει ό,τι και στο update.
- συμβούν. :

v = x ++

v = x--;

v = ++ x

v = - x;

v = x binop = expr;

v = x = x binop expr;

v = x = expr binopx

Η δομή της οδηγίας atomic (3/4)

- Το structured-block είναι ένα δομημενο μπλοκ με μία από τις ακόλουθες μορφές:

{v = x; x *binop*= *expr*;**}**
{x *binop*= *expr*; **v = x;}**
{v = x; x = x *binop expr*;**}**
{v = x; x = *expr binop* **x;}**
{x = x *binop expr*; **v = x;}**
{x = *expr binop* **x; v = x;}**
{v = x; x = *expr*;**}**
{v = x; x++;}
{v = x; ++x;}
{++x; v = x;}
{x++; v = x;}
{v = x; x--;}
{v = x; --x;}
{--x; v = x;}
{x--; v = x;}

Η δομή της οδηγίας atomic (4/4)

- Στις προηγούμενες εκφράσεις:
- Τα x και v (ανάλογα με την περίπτωση) είναι και οι δύο εκφράσεις της τιμής l με κλιμακωτό τύπο.
- Κατά την εκτέλεση μιας atomic περιοχής, πολλαπλές συντακτικές περιπτώσεις του x πρέπει να χαρακτηρίζουν την ίδια θέση αποθήκευσης.
- Κανένα από τα v και expr (κατά περίπτωση) δεν μπορεί να έχει πρόσβαση στη θέση αποθήκευσης που ορίζεται από το x.
- Κανένα από τους x και expr (κατά περίπτωση) μπορεί να έχει πρόσβαση στη θέση αποθήκευσης που καθορίζεται από v.
- Το expr είναι μια έκφραση τύπου scalar.
- Το binop είναι ένα από τα +, *, -, /, &, ^, |, << or >>.
- Το binop, binop =, ++ και - δεν είναι υπερφορτωμένοι τελεστές.
- Η έκφραση x binop expr πρέπει να είναι μαθηματικά ισοδύναμη με το x binop (expr). Αυτή η απαίτηση ικανοποιείται εάν οι χειριστές σε expr έχουν προτεραιότητα μεγαλύτερη από το binop, ή χρησιμοποιώντας παρενθέσεις γύρω από expr ή υποεκφράσεις του expr.
- Η έκφραση expr binop x πρέπει να είναι μαθηματικά ισοδύναμη με την (expr) binop x. Αυτή η απαίτηση ικανοποιείται εάν οι τελεστές στην expr έχουν προτεραιότητα ίση ή μεγαλύτερη από το binop ή χρησιμοποιώντας παρενθέσεις γύρω από expr ή υποεκφράσεις του expr.
- Για μορφές που επιτρέπουν πολλαπλές εμφανίσεις του x, ο αριθμός των επαναλήψεων x δεν είναι προσδιορισμένος.

Λειτουργίες της οδηγίας atomic μέσα σε κώδικα της parallel (1/3)

- Όταν μία ή περισσότερες επαναλήψεις βρόχου πρέπει να έχουν πρόσβαση σε ένα στοιχείο στη μνήμη ταυτόχρονα , μπορεί να προκύψουν *data races*.
- Για παράδειγμα, εάν μια επανάληψη βρόχου τροποποιεί την τιμή που περιέχεται σε μια μεταβλητή και μια άλλη προσπαθεί να διαβάσει από την ίδια μεταβλητή παράλληλα, μπορεί να προκύψουν διαφορετικά αποτελέσματα ανάλογα με την επανάληψη που εμφανίζεται πρώτη.
- Το directive atomic δέχεται ένα από τα τέσσερα clauses για να δηλώσει τον τύπο της λειτουργίας που περιέχεται στην περιοχή.

Λειτουργίες της οδηγίας atomic μέσα σε κώδικα της parallel (2/3)

- Το *read* εξασφαλίζει ότι δεν θα διαβάζονται ταυτόχρονα από την περιοχή δύο επαναλήψεις βρόχου.

- Το *write* θα εξασφαλίσει ότι δεν υπάρχουν δύο επαναλήψεις με εγγραφή στην περιοχή ταυτόχρονα.

- Το *update* είναι μια συνδυασμένη ανάγνωση και εγγραφή. Εάν δεν δίνεται καμία δήλωση τότε θα πραγματοποιηθεί μια διαδικασία τύπου update.
- Το *capture* αναγκάζει μια ατομική ενημέρωση της θέσης που υποδεικνύεται από το x χρησιμοποιώντας τον ορισμένο τελεστή ενώ ταυτόχρονα καταγράφει την αρχική ή την τελική τιμή της θέσης που ορίζεται από το x σε σχέση με την ατομική ενημέρωση.

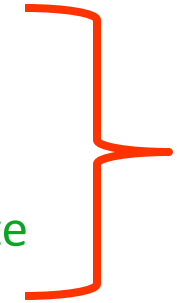
Παράδειγμα χρήσης της οδηγίας atomic

- Ένα ιστόγραμμα είναι μια συνηθισμένη τεχνική για την καταμέτρηση του πόσες φορές προκύπτουν οι τιμές από ένα σύνολο εισόδων ανάλογα με την αξία τους.
- Το σχήμα παρακάτω δείχνει ένα ιστόγραμμα που μετράει τον αριθμό των φορών που οι αριθμοί εμπίπτουν σε συγκεκριμένα εύρη.
- Ο παρακάτω κώδικας παραδείγματος βγαίνει μέσω μιας σειράς ακέραιων αριθμών γνωστού εύρους και μετρά τις εμφανίσεις κάθε αριθμού σε αυτό το εύρος.
- Δεδομένου ότι κάθε αριθμός στην περιοχή μπορεί να εμφανιστεί πολλές φορές, πρέπει να διασφαλίσουμε ότι κάθε στοιχείο του πίνακα του ιστογράμματος θα ενημερώνεται ατομικά.

Παράδειγμα χρήσης της οδηγίας atomic (1/2)

- Ο παρακάτω κώδικας δείχνει τη χρήση του atomic directive για τη δημιουργία ενός ιστογράμματος:

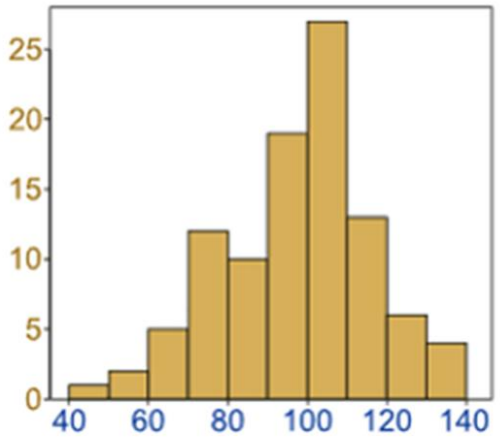
```
for (int it=0; i<ITERS; it++){
#pragma acc parallel loop
            for (int i=0; i<HN; i++)
            {
             h[ i ]=0;
            }
 #pragma acc parallel loop
           for (int i=0; i<N; i++)
            {
            #pragma acc atomic update
                        h[a[ i ]]+=1;
            }
}
```



Το h μπορεί να έχει πρόσβαση από ένα μόνο νήμα τη φορά

Παράδειγμα χρήσης της οδηγίας atomic (2/2)

- Παρατηρήστε ότι οι ενημερώσεις του πίνακα h του ιστογράμματος εκτελούνται ατομικά.
- Επειδή αυξάνουμε την τιμή του στοιχείου πίνακα, χρησιμοποιείται μια λειτουργία update για να διαβάσει την τιμή, να την τροποποιήσει και στη συνέχεια να την γράψει πίσω.



Η χρήση του OpenACC

Το πρώτο πρόγραμμα σε OpenACC: SAXPY

- Παράδειγμα: Υπολογίστε ***a*x+y***, όπου x και y είναι διανύσματα, και a είναι πραγματικός αριθμός.

```
int main( int argc, char **argv){
        int N=1000;
        float a=3.0f;
        float x[N], y[N];
for (int i=0; i<N; ++i){
        x[ i ]=2.0f;
        y[ i ]=1.0f;
}
#pragma acc kernels
for (int i=0; i<N; ++i){
        y[ i ]= a* x[i + y[ i ];
}

}
```

Pointer aliasing στη C (1)

- Μια ακατάλληλη εκδοχή του κώδικα SAXPY (με τη χρήση δεικτών)

```
int N=1000;
float a = 3.0f;
float * x = (float*)malloc(N * sizeof(float));
float * y = (float*)malloc(N * sizeof(float));

for (int i = 0; i < N; ++i) {
  x[i] = 2.0f;
  y[i] = 1.0f;
}

#pragma acc kernels
for (int i = 0; i < N; ++i) {
  y[i] = a * x[i] + y[i];
}
```

- ✓ Διαφορετικοί δείκτες επιτρέπουν πρόσβαση στο ίδιο αντικείμενο. Αυτό ίσως προκαλέσει εξάρτηση των έμμεσων δεδομένων σε ένα βρόχο.
- ✓ Σε αυτή τη περίπτωση, είναι πιθανό οι δείκτες x και y να επιτρέπουν τη πρόσβαση στο ίδιο αντικείμενο. Ενδεχομένως, να υπάρχει εξάρτηση δεδομένων στο βρόχο.

Pointer aliasing στη C (2)

- Ο μεταγλωττιστής αρνείται να παραλληλοποιήσει το βρόχο που περιέχει χρήση δεικτών.
- Η έξοδος του ακατάλληλου κώδικα SAXPY, μετά την αποσφαλμάτωση φαίνεται παρακάτω:

```
.....
20, Loop carried dependence of y-> prevents parallelization
    Complex loop carried dependence of x-> prevents parallelization
    Loop carried backward dependence of y-> prevents vectorization
    Accelerator scalar kernel generated
```

Χρήση restrict για την αποφυγή του pointer aliasing

- Μια κατάλληλη εκδοχή του κώδικα SAXPY (με τη χρήση δεικτών):

```
int N=1000;
float a = 3.0f;
float *x = (float*)malloc(N * sizeof(float));
float * restrict  y = (float*)malloc(N * sizeof(float));

for (int i = 0; i < N; ++i) {
   x[i] = 2.0f;
   y[i] = 1.0f;
}

#pragma acc kernels
for (int i = 0; i < N; ++i) {
   y[i] = a * x[i] + y[i];
}
```

- ✓ Για την αποφυγή του φαινομένου pointer aliasing, γίνεται χρήση του restrict.
- ✓ Σημασία του restrict: Για τη διάρκεια ζωής του δείκτη ptr, μόνο ο ίδιος ή μια τιμή που προέρχεται από αυτόν (όπως ptr+1) θα χρησιμοποιηθεί για την πρόσβαση του αντικειμένου στο οποίο δείχνει.

Διαφορές μεταξύ των directives kernels και parallel

Ειδικά για το directive parallel (1)

- Μια ακατάλληλη εκδοχή του κώδικα SAXPY (με τη χρήση της οδηγίας parallel) :

```
#pragma acc parallel
for (int i = 0; i < N; ++i) {
    y[i] = a * x[i] + y[i];
}
```

- ✓ Η οδηγία parallel υποδεικνύει στον μεταγλωττιστή να δημιουργήσει μια παράλληλη δομή. Διαφορετικά από τη δομή kernels, ο κώδικας στο βρόχο parallel εκτελείται άσκοπα (από όλα τα gangs). Δεν υπάρχει λόγος η εργασία να μοιραστεί ανάμεσα στα gangs.

Ειδικά για το directive parallel (2)

- Μια κατάλληλη εκδοχή του κώδικα SAXPY (με τη χρήση της οδηγίας parallel loop):

```
#pragma acc parallel loop
for (int i = 0; i < N; ++i) {
    y[i] = a * x[i] + y[i];
}
```

- ✓ Είναι απαραίτητο να προστεθεί η λέξη-κλειδί loop για να μοιραστεί η εργασία μεταξύ των gangs.
- ✓ Στην C, η λέξη κλειδί loop μπορεί να αντικατασταθεί από το for.

Διαφορές μεταξύ των directives kernels και parallel (1)

- kernels
  - Πιο έμμεσο
  - Δίνει ελευθερία στον μεταγλωττιστή να βρεί την παραλληλοποίηση
  - Ο μεταγλωττιστής εφαρμόζει ανάλυση της παραλληλοποίησης και παραλληλοποιεί ο,τι θεωρεί ασφαλές

- parallel
  - Πιο άμεσο
  - Απαιτεί ανάλυση από τον προγραμματιστή για να εξασφαλίσει ασφαλή παραλληλοποίηση
  - Προερχόμενο από το OpenMP

Διαφορές μεταξύ των directives kernels και parallel (1)

❑Παραλληλοποίηση μπλοκ κώδικα με δύο βρόχους:

- Kernels

```
#pragma acc kernels
{
   for (i=0; i<n; i++)
      a[i] = 3.0f*(float)(i+1);
   for (i=0; i<n; i++)
      b[i] = 2.0f*a[i];
}
```

✓ Δημιουργία δύο πυρήνων

✓ Υπάρχει ένα έμμεσο φράγμα (implicit barrier) μεταξύ των δύο βρόχων: το δεύτερο loop θα ξεκινήσει αφού τελειώσει το πρώτο.

- Parallel

```
#pragma acc parallel
{
   #pragma acc loop
   for (i=0; i<n; i++) a[i] = 3.0f*(float)(i+1);
   #pragma acc loop
   for (i=0; i<n; i++)  b[i] = 2.0f*a[i];
}
```

✓ Δημιουργία ενός πυρήνα

✓ Δεν υπάρχει κανένα εμπόδιο μεταξύ των δύο loops: το δεύτερο loop μπορεί να ξεκινήσει πριν τελειώσει το πρώτο. (Διαφορετικό από το OpenMP)

Οδηγία parallel vs. Οδηγία kernels (Ποια είναι η καλύτερη;)

- Αρχικά για να απλοποίησουμε αυτό το ζήτημα, οι kernels αφήνουν τη λήψη αποφάσεων στο μεταγλωττιστή, κάτι το οποίο δεν είναι απαραίτητα κακό με την εμπιστοσύνη στο μεταγλωττιστή (“καλό θα ήταν να επαληθεύσετε”) κι αυτό ίσως είναι ένα σημείο για να σας βοηθήσει να ξεκινήσετε.

- Αποκτώντας περισσότερη εμπειρία θα καταλάβετε ότι η δομή parallel σας επιτρέπει μια πιο άμεση και σαφή κατανόηση. Από την άλλη πλευρά, καθώς οι μεταγλωττιστές “ωριμάζουν“ θα έχουν τη δυνατότητα να κάνουν ακριβώς αυτό που πρέπει.

Παράδειγμα: Επίλυση Εξίσωσης Laplace στο BU Shared Computing Cluster (SCC)

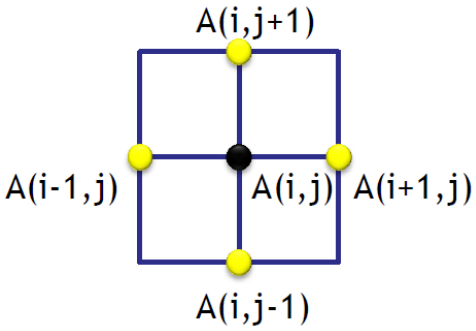
Επίλυση Laplace (1)

- Διδιάστατη εξίσωση Laplace $\nabla^2 f(x,y) = 0$

- Διακρίνουμε τη Λαπλασιανή με τη διαφορική μέθοδο πρώτης τάξης και εκφράζουμε τη λύση ως:

$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

- Η λύση σε ένα σημείο εξαρτάται από τα τέσσερα σημεία των γειτονικών



Επίλυση Laplace (2)

- Χρήση του αλγορίθμου *επανάληψης Jacobi* για να καταλήξουμε σε συγκλίνουσα λύση:

- Αλγόριθμος επανάληψης Jacobi

1. Δώστε μια δοκιμαστική λύση όπου το A να εξαρτάται από την αρχική κατάσταση που δίνεται
2. Υπολογίστε μια νέα λύση, την A_new(i,j) , βασισμένη σε παλιές τιμές από τα τέσσερα γειτονικά σημεία
3. Ενημερώστε τη λύση δηλαδή, A=A_new,
4. Επαναλάβετε τα βήματα 2 και 3 μέχρι να συγκλίνουν, δηλαδή , max(| A_new(i,j)-A(i,j)|)<tolerance.
5. Τελικά η συγκλίνουσα λύση αποθηκεύεται στο A.

Επίλυση εξίσωσης Laplace (σειριακά σε C)

```
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++) {
      A_new[i][j] = 0.25 * (A[i+1][j] + A[i-1][j] +A[i][j+1] + A[i][j-1]);
  }
  dt = 0.0;
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(A_new[i][j]-A[i][j]), dt);
      A[i][j] = A_new[i][j];
  }
  iteration++;
}
```

Ένας βρόχος για τις επαναλήψεις Jacobi

Βρόχοι για τον υπολογισμό νέας λύσης

Βρόχοι για την ενημέρωση της λύσης και την εύρεση του μεγίστου σφάλματος

Επίλυση εξίσωσης Laplace στο OpenACC

- Προϋποθέτοντας ότι υπάρχει ένας σειριακός κώδικας (σε C) για την επίλυση της διδιάστατης εξίσωσης Laplace , παραλληλίστε τη με τη χρήση οδηγιών του OpenACC. Στη συνέχεια συγκρίνετε την απόδοση μεταξύ του σειριακού κώδικα και του παραλληλοποιημένου με OpenACC.

- Χρήσιμες συμβουλές

1. Βρείτε τα “hot spots”, τα σημεία που απαιτούν περισσότερο χρόνο μέσα στον κώδικα.
2. Αναλύστε την παραλληλοποίηση. Ποιοί βρόχοι μπορούν να παραλληλοποιηθούν;
3. Ποιες οδηγίες (directives) πρέπει να χρησιμοποιηθούν; Πού πρέπει να τοποθετηθούν;

Επίλυση εξίσωσης Laplace στο OpenACC (1η εκδοχή)

```
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++) {
      A_new[i][j] = 0.25 * (A[i+1][j] + A[i-1][j] +A[i][j+1] + A[i][j-1]);
  }
  dt = 0.0;
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(A_new[i][j]-A[i][j]), dt);
      A[i][j] = A_new[i][j];
  }
  iteration++;
}
```

Ο βρόχος δεν μπορεί να παραλληλοποιηθεί λόγω της ανεξαρτησίας των δεδομένων

Αυτοί οι βρόχοι είναι παραλληλοποιήσιμοι. Δημιουργεί μια περιοχή kernel και ρωτά τον μεταγλωττιστή να καθορίσει την παραλληλοποίηση και τη μεταφορά δεδομένων

Αυτοί οι βρόχοι είναι παραλληλοποιήσιμοι. Δημιουργεί μια περιοχή kernel και ρωτά τον μεταγλωττιστή να καθορίσει την παραλληλοποίηση και τη μεταφορά δεδομένων

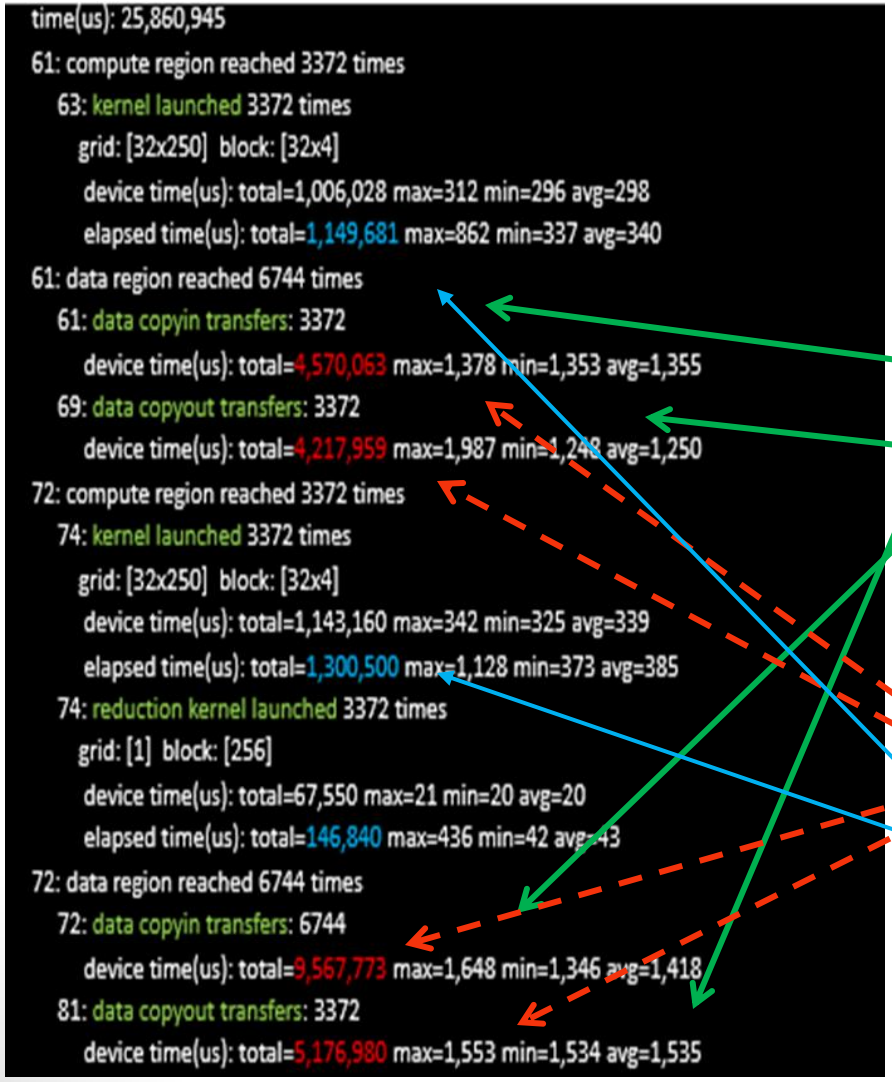
Ανάλυση της εκτέλεσης (1ης εκδοχής)

- Συγκρίνοντας τον χρόνο υπολογισμού (1000*1000 grid):
- Σειριακός κώδικας: 17.610445 seconds.
- Κώδικας OpenACC (1η εκδοχή): 48.796347 seconds.

✓Ο κώδικας του OpenACC είναι πολύ πιο αργός από ότι ο σειριακός. Τι πήγε λάθος;

✓Πρέπει να αναλύσουμε περαιτέρω την παραλληλοποίηση και τη μεταφορά δεδομένων.

Δυναμική ανάλυση του προγράμματος (1η εκδοχή)



- Ο μεταγλωττιστής PGI παρέχει αυτόματα όργανα όταν PGI_ACC_TIME = 1 κατά το χρόνο εκτέλεσης για να ενεργοποιηθεί το profiling και εκτελέστε ξανά.

- Υπάρχει μεταφορά δεδομένων 4 φορές μεταξύ του host (CPU) και της μνήμης GPU σε κάθε επανάληψη του εξωτερικού βρόχου while.

- Ο συνολικός χρόνος για τη μεταφορά δεδομένων είναι περίπου 23.6 seconds, που είναι μακράν μεγαλύτερος από τον εκτιμώμενο χρόνο 2.5 seconds.

Ανάλυση της μεταφοράς δεδομένων (1η εκδοχή)

- Αυτή η μεταφορά δεδομένων γίνεται σε κάθε επανάληψη του εξωτερικού βρόχου while!

```
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++) {
      A_new[i][j] = 0.25 * (A[i+1][j] + A[i-1][j] +A[i][j+1] + A[i][j-1]);
  }
  dt = 0.0;
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(A_new[i][j]-A[i][j]), dt);
      A[i][j] = A_new[i][j];
  }
  iteration++;
}
```

Εσωτερική αντιγραφή: A και A_new αντιγράφονται από τη CPU στη GPU

Εξωτερική αντιγραφή: A και A_new αντιγράφονται από τη GPU στη CPU

Εσωτερική αντιγραφή: A και A_new αντιγράφονται από τη CPU στη GPU

Εξψτερική αντιγραφή: A και A_new αντιγράφονται από τη GPU στη CPU

Επίλυση εξίσωσης Laplace στο OpenACC (2η εκδοχή)

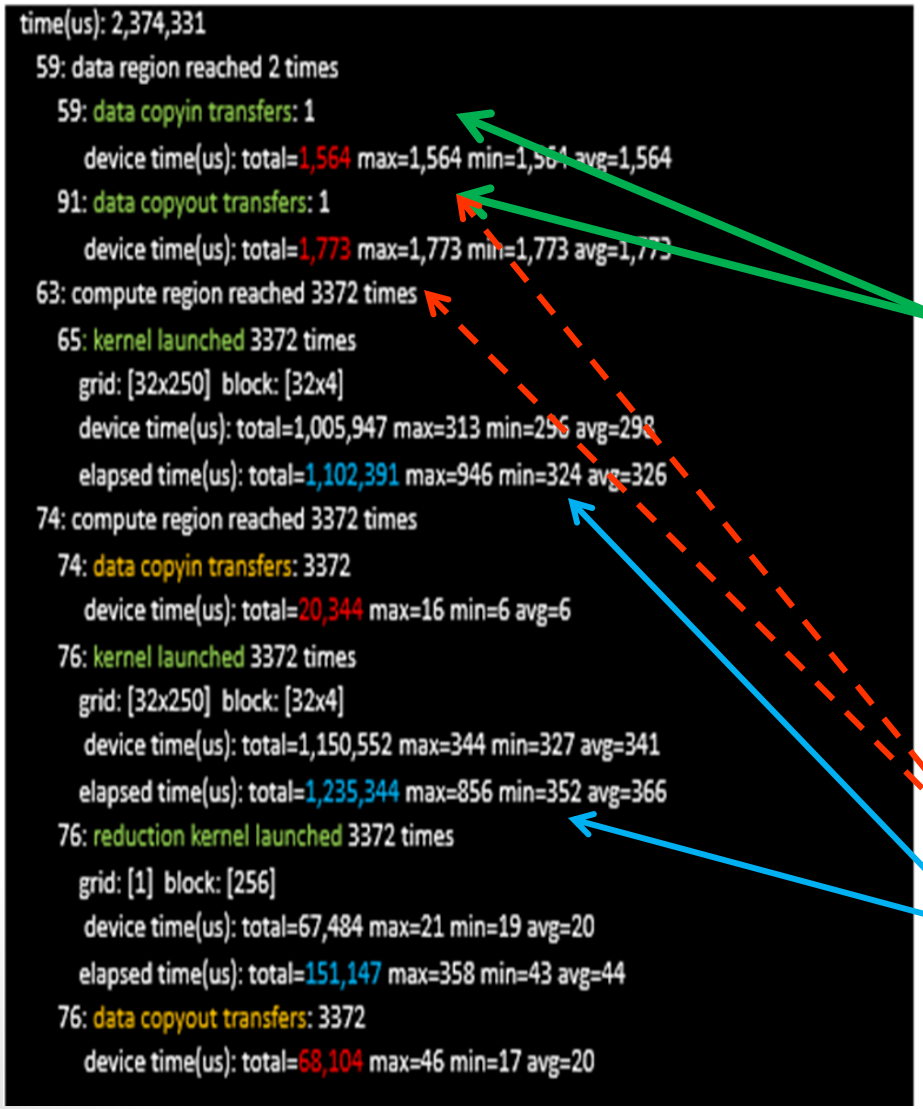
```
#pragma acc data copy(A), create(A_new)
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
    #pragma acc kernels
    for(i = 1; i <= ROWS; i++)
      for(j = 1; j <= COLUMNS; j++) {
        A_new[i][j] = 0.25 * (A[i+1][j] + A[i-1][j] + A[i][j+1] + A[i][j-1]);
    }
    dt = 0.0;
    #pragma acc kernels
    for(i = 1; i <= ROWS; i++)
      for(j = 1; j <= COLUMNS; j++){
        dt = fmax( fabs(A_new[i][j]-A[i][j]), dt);
        A[i][j] = A_new[i][j];
    }
    iteration++;
}
```

Εδώ δημιουργείται μια περιοχή data. Το A αντιγράφεται εσωτερικά πριν την έναρξη του βρόχου while και αντιγράφεται εξωτερικά μετά το τέλος του. Το A_new εντοπίζεται απευθείας στην μνήμη της GPU και είναι περιττή η αντιγραφή του στην CPU

Δημιουργεί μία περιοχή kernel για να παραλληλοποίησει τους βρόχους for, αλλά δεν υπάρχει μεταφορά δεδομένων

Δημιουργεί μία περιοχή kernel για να παραλληλοποίησει τους βρόχους for, αλλά δεν υπάρχει μεταφορά δεδομένων

Δυναμική ανάλυση του προγράμματος (2η εκδοχή)



- Ο μεταγλωττιστής PGI παρέχει αυτόματα όργανα όταν PGI_ACC_TIME = 1 κατά το χρόνο εκτέλεσης για να ενεργοποιηθεί το profiling και εκτελέστε ξανά.
- Υπάρχει κίνηση δεδομένων (των πινάκων) μόνο 2 φορές συνολικά.
- Υπάρχει κίνηση δεδομένων για τη μεταβλητή dt, αλλά επειδή δεν είναι πίνακας γιαυτό και η διαδικασία μεταφοράς χρειάζεται πολύ λιγότερο χρόνο.
- Ο συνολικός χρόνος για τη μεταφορά δεδομένων είναι περίπου 0.09 seconds, που είναι μικρότερος από τον εκτιμώμενο χρόνο (περίπου 2.5 seconds).

Ανάλυση της εκτέλεσης (2ης εκδοχής)

- Συγκρίνοντας τον χρόνο υπολογισμού (για 1000*1000 grid):

- Σειριακός κώδικας: 17.610445 seconds.
- Κώδικας OpenACC (1η εκδοχή): 48.796347 seconds.
- Κώδικας OpenACC (2η εκδοχή): 2.592581 seconds.

✓ Ο κώδικας του OpenACC (2η εκδοχή) είναι περίπου 6.8 φορές γρηγορότερος από το σειριακό κωδικό!

Ανάλυση της εκτέλεσης (2ης εκδοχής)

- Το speed-up θα ήταν μεγαλύτερο εάν το μέγεθος του προβλήματος αυξανόταν.

- Το μέγιστο μέγεθος της μνήμης GPU (τυπικά 6GB ή 12 GB) είναι πολύ μικρότερο από την κανονική CPU μνήμη (δηλαδή 128 GB στο BU SCC).

Δημιουργία reduction kernel (1)

- Όπως μπορούμε να δούμε από τη δυναμική ανάλυση του προγράμματος , δημιουργείται ένα reduction kernel από το μεταγλωττιστή.
  - Τι είναι το reduction kernel και γιατί είναι απαραίτητο;

- Στο προηγούμενο παράδειγμα, η μεταβλητή *dt* μπορεί να τροποποιηθεί από πολλαπλούς workers (warps) ταυτόχρονα. Αυτό ονομάζεται κατάσταση data race. Εάν συμβεί αυτό, θα επιστραφεί λανθασμένο αποτέλεσμα.

Δημιουργία reduction kernel (2)

- Για την αποφυγή της κατάστασης data race, απαιτείται ένα δήλωση reduction για να προστατευτεί η σχετική μεταβλητή.
- Ευτυχώς, ο compiler είναι αρκετά έξυπνος για να δημιουρήσει ένα reduction kernel και να αποφευχθεί το data race αυτόματα!

```
dt = 0.0;
#pragma acc kernels
 for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(A_new[i][j]-A[i][j]), dt);
      A[i][j] = A_new[i][j];
 }
```

Επίλυση εξίσωσης Laplace στο OpenACC (3η εκδοχή)

```
#pragma acc data copy(A), create(A_new)
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
    #pragma acc parallel loops
    for(i = 1; i <= ROWS; i++)
      for(j = 1; j <= COLUMNS; j++) {
        A_new[i][j] = 0.25 * (A[i+1][j] + A[i-1][j] + A[i][j+1] + A[i][j-1]);
    }
    dt = 0.0;
    #pragma parallel loops reduction(max:dt)
    for(i = 1; i <= ROWS; i++)
      for(j = 1; j <= COLUMNS; j++){
        dt = fmax( fabs(A_new[i][j]-A[i][j]), dt);
        A[i][j] = A_new[i][j];
    }
    iteration++;
}
```

Εδώ δημιουργείται μία δομή για την οδηγία data

Δημιουργεί μια περιοχή για την οδηγία parallel και παραλληλοποιεί τους βρόχους for

Δημιουργεί μια περιοχή για την οδηγία parallel και παραλληλοποιεί τους βρόχους for. Καθορίζει ρητά τον τελεστή και τη μεταβλητή μείωσης

Ανάλυση της εκτέλεσης (3ης εκδοχής)

- Συγκρίνοντας τον χρόνο υπολογισμού (για 1000*1000 grid):
- Σειριακός κώδικας: 17.610445 seconds.
- Κώδικας OpenACC (1η εκδοχή): 48.796347 seconds.
- Κώδικας OpenACC(2η εκδοχή): 2.592581 seconds.
- Κώδικας OpenACC(3η εκδοχή): 2.259797 seconds

✓ Χρησιμοποιώντας το directive parallel ο κώδικας είναι ελαφρώς γρηγορότερος από ότι χρησιμοποιώντας το directive kernel σε αυτή την περίπτωση, κυρίως λόγω της μέτρησης των εργασιών ή του υπολογισμού των εν λόγω εργασιών – task granularities.

✓ Είναι καλό να καθορίζονται άμεσα οι μεταβλητές και τελεστές τύπου reduction.

NVIDIA GPU (CUDA) Task Granularity (1/2)

- Συσκευή GPU - Πλέγματα CUDA:
- Τα πλέγματα αντιστοιχίζονται σε μια συσκευή.
- • Πολυεπεξεργαστές ροής (Streaming Multiprocessor-SM) – block νημάτων CUDA :
- Τα block εκχωρούνται έναν πολεπεξεργαστή ροής.
- • CUDA πυρήνες – CUDA νήματα:
- Τα νήματα εκχωρούνται σε έναν πυρήνα.

- Warp: μια μονάδα που αποτελείται από 32 νήματα.
- Τα μπλοκ χωρίζονται σε warps.
- Ο πολυεπεξεργαστής ροής εκτελεί τα νήματα σε warp κοκκιότητα.
- Το μέγεθος του warp μπορεί να αλλάξει στο μέλλον.

NVIDIA GPU (CUDA) Task Granularity (2/2)

Kepler GK110 full chip block diagram

Μεταγλώττιση OpenACC με τον BU SCC

- Μεταγλώττιση ενός OpenACC πηγαίου κώδικα:

% **pgcc -acc –Minfo=accel *name.c* –o exe*name***

% **pgf90 -acc –Minfo=accel *name.f90* –o exe*name***

• Σημείωση: η επιλογή **–Minfo=accel** χρησιμοποιείται για να εκτυπώνει πληροφορίες σχετικά με το στόχο της περιοχής του επιταχυντή.

Μεταγλώττιση OpenACC με τον GCC

- **-fopenacc**

Ενεργοποιεί το OpenACC στον μεταγλωττιστή, ενεργοποιεί τις οδηγίες στον κώδικα και παρέχει το σύμβολο προεπεξεργαστή _OPENACC.

- **-foffload-force**

Καθορίζει τη μετακίνηση στόχων και επιλογών για αυτούς. Η πιο συνηθισμένη μορφή χρήσης για αυτή την επιλογή θα είναι πιθανότατα να συνδεθεί στη βιβλιοθήκη μαθηματικών στο στόχο επιταχυντή, δηλαδή -foffload= -lm.

- **-fopt-info-note-omp**

Αυτή η σημαία καθοδηγεί τον μεταγλωττιστή να εκτυπώσει τυχόν παραλληλισμούς που ανίχνευσε.

- **-fopenacc-dim=geom**

Χρήση της διαμόρφωσης geom για τα νήματα.

Εύρεση παραλληλοποίησης μέσα στον κώδικα σας

- Είναι καλύτερα για την παραλληλοποίηση, να είναι φωλιασμένη στους βρόχους.
- Οι επαναλήψεις των βρόχων πρέπει να είναι ανεξάρτητες η μία από την άλλη.
- Ο μεταγλωττιστής θα πρέπει να μπορεί να υπολογίσει το μεγέθος των περιοχών δομένων
  - Μπορεί να χρησιμοποίησει τις οδηγίες για να ελέγξει άμεσα το μεγέθος.

Εύρεση παραλληλοποίησης μέσα στον κώδικα σας

- Εάν είναι δυνατόν, η αριθμητική δεικτών καλό θα είναι να αποφεύγεται.
- Οι κλήσεις των συναρτήσεων μέσα στην περιοχή επιτάχυνσης πρέπει να είναι ευδιάκριτες.

Βελτιστοποίηση κίνησης δεδομένων

- Οι μεταγλωττιστές θα είναι προσεκτικοί με την κίνηση των δεδομένων και ενδέχεται να μεταφέρουν περισσότερα δεδομένα που είναι απαραίτητα.
  - Εάν είναι αριστερά από το '=', πιθανότατα θα αντιγραφεί από τη συσκευή.
  - Εάν είναι δεξιά από το '=', πιθανόν να αντιγραφεί στη συσκευή.

- Ο Profiler CUDA μπορεί να χρησιμοποιηθεί για τη μέτρηση της κίνησης των δεδομένων.
- Ο Cray Compiler διαθέτει επίσης τη μεταβλητή περιβάλλοντος εκτέλεσης CRAY_ACC_DEBUG, η οποία θα εκτυπώσει χρήσιμες πληροφορίες.
- Για λεπτομέρειες δείτε man intro_openacc.

Βελτιστοποίηση κίνησης δεδομένων

- *Βήμα 1ο*, τοποθετήστε τον κώδικα της οδηγίας data στον βρόχο προσομοίωσης
  - Χρησιμοποιήστε αυτό το για να δηλώσετε τα δεδομένα που πρέπει να γίνουν copy in, copy out ή δημιουργία μέσα στη συσκευή.
  - Χρησιμοποιήστε το παρόν clause για να δηλώσετε τα μέρη όπου ο μεταγλωττιστής μπορεί να μην συνειδητοποιήσει ότι τα δεδομένα υπάρχουν ήδη στη συσκευή (για παράδειγμα στις κλήσεις συναρτήσεων)
- *Βήμα 2ο*, χρησιμοποιήστε ένα directive update για να αντιγράψετε δεδομένα μεταξύ της GPU και της CPU εντός της περιοχής data, όπως απαιτείται.

Βελτιστοποίηση kernels

- Ο μεταγλωττιστής έχει την ελευθερία να προγραμματίζει loops και kernels όπως θεωρεί ο ίδιος καλύτερα, αλλά ο προγραμματιστής μπορεί να το παραλείψει αυτό.

- Αρχικά πρέπει να γνωστοποιηθεί πως η εργασία αποσυνθέθηκε.

  - Feedback από τον μεταγλωττιστή κατά το χρόνο κατασκευής
  - Feedback από το εκτελέσιμο κατά το χρόνο εκτέλεσης
  - Cuda Profiler

Προσαρμόζοντας την αποσύνθεση

- Ρυθμίστε τον αριθμό των gangs, των workers ή και το μήκος του vector στην περιοχή του parallel ή στην περιοχή του kernels
  - num_gangs, num_workers, vector_length
- Προσθέστε το directive loop σε μεμονωμένους βρόχους δηλώνοντας τους ως gang, worker ή παραλληλοποίηση του vector.

Περαιτέρω βελτιστοποίηση kernels

- Χρησιμοποιείστε το loop collapse() για να συγνωνεύσετε τους βρόχους και να αυξήσετε την παραλληλοποίηση σε συγκεκριμένα επίπεδα
- Χρησιμοποιήστε τις υπάρχοντα directive του μεταγλωττιστή σχετικά με τις βελτιστοποιήσεις του βρόχου
  - Αποδέσμευση του βρόχου
  - Ένωση/ διάσπαση βρόχου
  - Αποκλεισμός βρόχου
- Εξασφαλίστε κατάλληλα πρότυπα πρόσβασης δεδομένων
  - Η συγχώνευση μνήμης, οι συγκρούσεις των banks της μνήμης και ο κλονισμός είναι εξίσου σημαντικά με το OpenACC όπως το CUDA / OpenCL
  - Αυτό επίσης πιθανόν να βοηθήσει όταν χρησιμοποιείτε την CPU.

Διαλειτουργικότητα

- Το OpenACC λειτουργεί ορθά και σε άλλες πλατφόρμες, CUDA C, CUDA Fortran, Βιβλιοθήκες
- Προσθέτοντας το OpenACC σε έναν υπάρχοντα κώδικα CUDA, το data clause deviceptr επιτρέπει τη χρήση των υπάρχοντων δομών δεδομένων.
- Προσθέτοντας το CUDA ή τη κλήση βιβλιοθήκης σε έναν κώδικα OpenACC, χρησιμοποιήστε host_data και use_device για να δηλώσετε τη χρήση μνήμης στη CPU ή στη GPU.

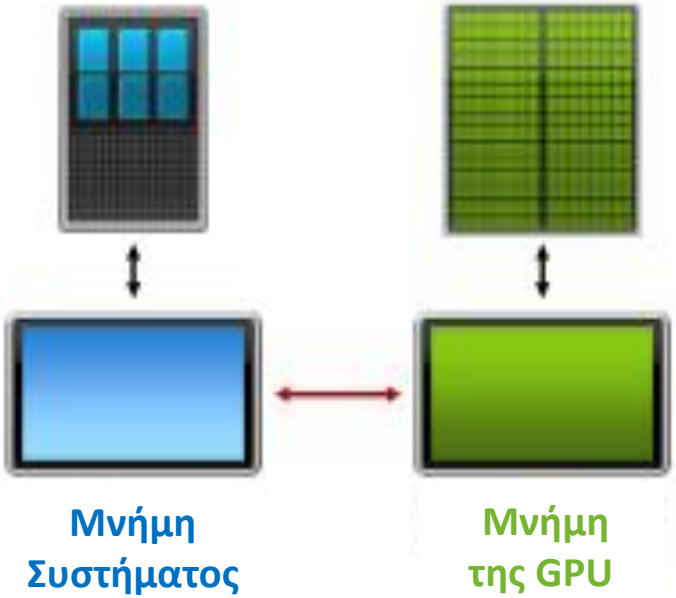
Συμβουλές διαλειτουργικότητας

- Το OpenACC παρέχει έναν πολύ απλό τρόπο διαχείρισης των δομών δεδομένων χωρίς να χρειάζεστε 2 δείκτες (host & device), γι 'αυτό χρησιμοποιήστε το στο υψηλότερο επίπεδο.
- Το CUDA παρέχει close-to-the-metal έλεγχο , ώστε να μπορεί να χρησιμοποιηθεί για πυρήνες πολύ υψηλής πυκνότητας που μπορούν να καλούνται από το OpenACC
- Οι μεταγλωττιστές κάνουν πολύπλοκες εργασίες όπως μειώσεις.

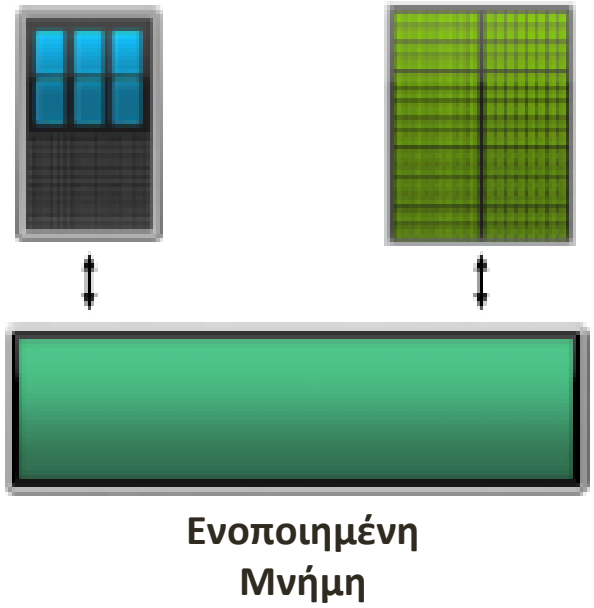
Επιτάχυνση με ενοποιημένη μνήμη

Η ενοποιημένη μνήμη βελτιώνει την παραγωγικότητα

Προβολή προγραμματιστή πριν



Προβολή με ενοποιημένη μνήμη



Το OpenACC και η ενοποιημένη μνήμη

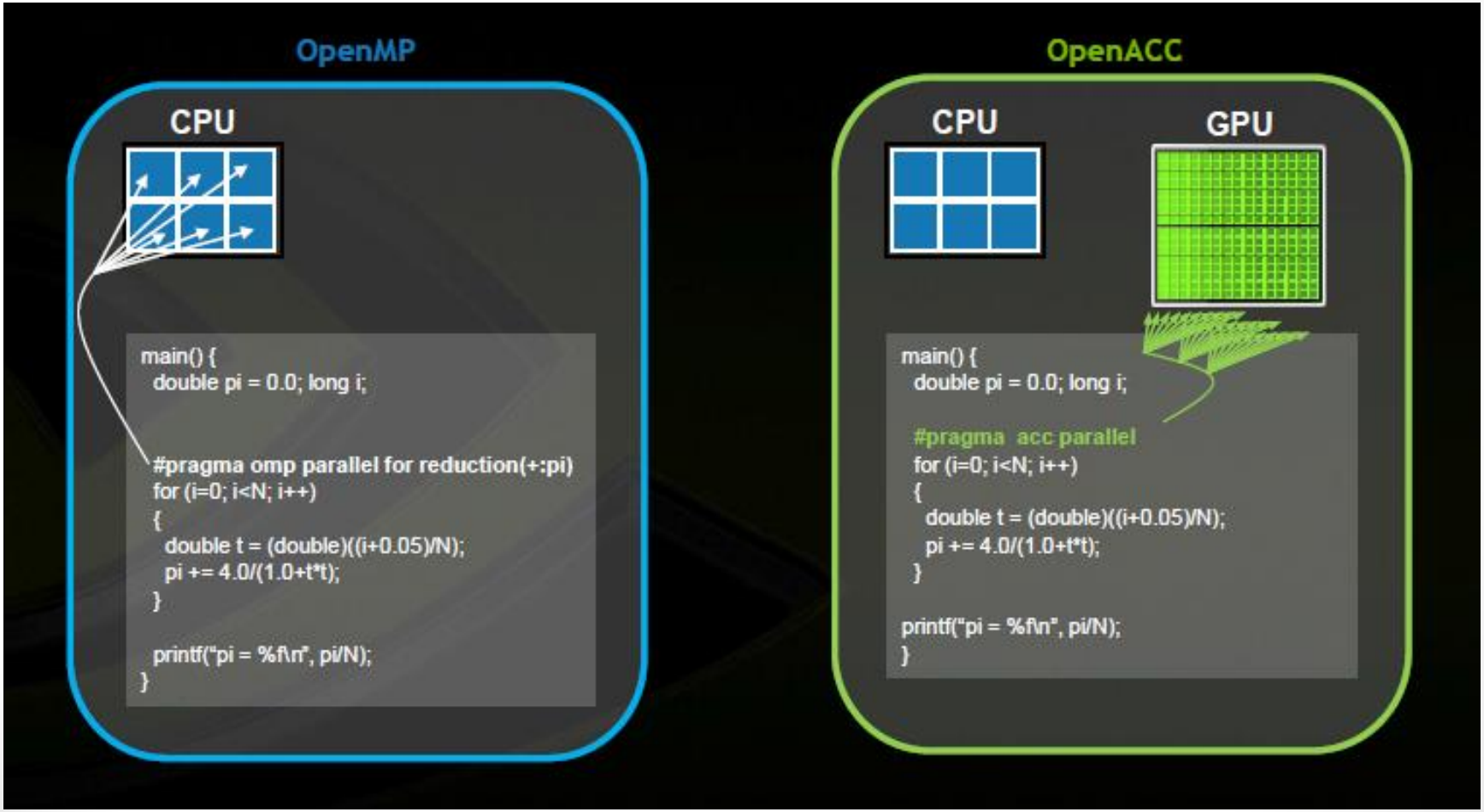
Τα πλεονεκτήματα

- Δεν χρειάζεται καμία δήλωση της οδηγίας data
- Δεν χρειάζεται η πλήρης κατανόηση της ροής των εφαρμογών των δεδομένων και της λογικής κατανομής
- Είναι δυνατή η σταδιακή επιτάχυνση του καθοδηγούμενου profiler
- Προοπτική προγραμματισμού της GPU σε Pascal

Ποιά η διαφορά μεταξύ OpenMP και OpenACC

- Το OpenMP είναι η κυρίαρχη μέθοδος για τον προγραμματισμό συστημάτων πολλαπλών πυρήνων και πολλαπλών επεξεργαστών κοινόχρηστης μνήμης στον τομέα της τεχνικής πληροφορικής.
- Ο στόχος του είναι ομοιογενείς πυρήνες με ομοιόμορφη κοινόχρηστη μνήμη κάτι που το χειρίζεται αρκετά καλά.
- Το OpenACC έχει μοντελοποιηθεί παρόμοια με το OpenMP, αν και υπάρχουν σημαντικές διαφορές μεταξύ των στόχων.
- Το OpenACC στοχεύει σε ένα σύστημα υποδοχής και επιταχυντή, όπου ο επιταχυντής μπορεί να έχει τη δική του μνήμη συσκευής.
- Επομένως, το OpenACC πρέπει να διαχειρίζεται την κίνηση δεδομένων καθώς και τον παραλληλοποίηση.
- Επιπλέον, οι στόχοι του OpenACC έχουν συνήθως πολλαπλά επίπεδα παραλληλοποίησης και το πρόγραμμα πρέπει να το διαχειριστεί και αυτό.

Διαφορές μεταξύ OpenMP και OpenACC



Ποια η διαφορά μεταξύ του OpenACC, OpenMP και OpenMPI;

- Το MPI είναι το πρότυπο τεχνικής πληροφορικής για την κατασκευή προγραμμάτων σε μια συστοιχία (cluster) ή σε ένα δίκτυο υπερυπολογιστών από κόμβους.
- Τα προγράμματα είτε χρησιμοποιούν άμεσα το MPI είτε, σε πολλές περιπτώσεις, χρησιμοποιούν μια πλατφόρμα εφαρμογών που αλληλεπιδρά με το MPI, ανακουφίζοντας τον προγραμματιστή από τις λεπτομέρειες των κλήσεων MPI.
- Το μοντέλο MPI είναι πολλά αντίγραφα του ίδιου προγράμματος που εκτελούνται παράλληλα σε μια συστοιχία ή σε έναν υπερυπολογιστή, αλληλεπιδρώντας και συνεργάζοντας για ένα μόνο πρόβλημα. Η γλώσσα προγραμματισμού δεν γνωρίζει τίποτα για τον παραλληλισμό, ωστόσο, ο παραλληλισμός MPI είναι κρυμμένος σε κλήσεις βιβλιοθήκης.
- Το OpenMP είναι το πρότυπο τεχνικής πληροφορικής για την κατασκευή προγραμμάτων πολλαπλών πυρήνων ή πολλαπλών επεξεργαστών κοινόχρηστης μνήμης, όπως προαναφέραμε. Πολλά προγράμματα χρησιμοποιούν MPI + OpenMP, με MPI μεταξύ κόμβων της συστποχίας και OpenMP σε κάθε κόμβο πολλαπλών πυρήνων κοινόχρηστης μνήμης.
- Το OpenMP υλοποιείται με οδηγίες και μερικές ρουτίνες της βιβλιοθήκης API και διατίθεται ευρέως από ουσιαστικά κάθε προμηθευτή στον τεχνικό υπολογιστικό χώρο. Το OpenACC στοχεύει σε ένα σύστημα υποδοχής και επιταχυντή, όπου ο επιταχυντής μπορεί να έχει τη δική του μνήμη συσκευής, όπως αναφέρθηκε παραπάνω.

Ποιά η διαφορά μεταξύ OpenCL και OpenACC;

- Το OpenCL (Open Computing Language) είναι ένα πλαίσιο για την εγγραφή προγραμμάτων που εκτελούνται σε ετερογενείς πλατφόρμες αποτελούμενες από κεντρικές μονάδες επεξεργασίας (CPU), μονάδες επεξεργασίας γραφικών (GPU), επεξεργαστές ψηφιακού σήματος (DSPs), πίνακες στο πεδίο-προγραμματισμού πύλης (FPGAs) και άλλους επεξεργαστές ή επιταχυντές υλικού.
- Η γλώσσα προγραμματισμού που χρησιμοποιείται για την εγγραφή πυρήνων υπολογιστών ονομάζεται OpenCL C και βασίζεται στο C99, κάτι που δεν υπάρχει στο OpenACC.
- Οι μεγαλύτερες διαφορές απόδοσης εμφανίζονται όταν εσείς, ο έξυπνος προγραμματιστής στο CUDA ή OpenCL, εκμεταλλευτείτε τις πληροφορίες που γνωρίζετε, αλλά ότι ο μεταγλωττιστής δεν μπορεί στην περίπτωση χρήσης OpenACC.
- Στο OpenACC με ελάχιστες πρόσθετες γραμμές κώδικα μπορούμε να εκμεταλλευτούμε τις τεράστιες δυνατότητες των καρτών γραφικών. Σε αντίθεση με άλλα παράλληλα προγραμματιστικά μοντέλα, όπως η CUDA και OpenCL, που απαιτείται πιο χρονοβόρα η διαδικασία συγγραφής των πολλών γραμμών κώδικα.

Βιβλιοθήκες χρόνου εκτέλεσης

Καθορισμός του χρόνου εκτέλεσης βιβλιοθηκών

- Στη C και C++, περιγράφονται τα πρωτότυπα για τις ρουτίνες της βιβλιοθήκης εκτέλεσης που παρέχονται σε ένα αρχείο κεφαλίδας με το όνομα *openacc.h*. Όλες οι ρουτίνες βιβλιοθήκης είναι εξωτερικές λειτουργίες με σύνδεση "C". Αυτό το αρχείο ορίζει:
- Τα πρωτότυπα όλων των ρουτινών.
- Οποιουσδήποτε τύπους δεδομένων χρησιμοποιούνται σε αυτά τα πρωτότυπα, συμπεριλαμβανομένου ενός τύπου απαρίθμησης για την περιγραφή τύπων επιταχυντών.
- Οι τιμές των **acc_async_noval** και **acc_async_sync**.
- Πολλές από τις ρουτίνες δέχονται ή επιστρέφουν μια τιμή που αντιστοιχεί στον τύπο της συσκευής επιταχυντή. Στε C και C++, ο τύπος δεδομένων που χρησιμοποιείται για τις τιμές τύπου συσκευής είναι acc_device_t.

Ρουτίνες εκτέλεσης βιβλιοθηκών

Fortran

use openacc or #include "openacc_lib.h"

acc_get_num_devices
acc_set_device_type
acc_get_device_type
acc_set_device_num
acc_get_device_num
acc_async_test
acc_async_test_all

C

#include "openacc.h"

acc_async_wait
acc_async_wait_all
acc_shutdown
acc_on_device
acc_malloc
acc_free
acc_init

Ρουτίνες του χρόνου εκτέλεσης βιβλιοθηκών

- Για τη C και C ++, οι δείκτες δηλώνονται h_void * ή d_void * για τον προσδιορισμό μιας διεύθυνσης κεντρικού υπολογιστή ή μιας διεύθυνσης συσκευής, και να συμπεριλαμβάνονται οι ακόλουθοι ορισμοί:

  #define h_void void

  #define d_void void

- Εκτός του **acc_on_device**, αυτές οι ρουτίνες είναι διαθέσιμες για τον υπολογιστή υποδοχής.

- Κάποιες από αυτές τις ρουτίνες εκτέλεσης χρόνου βιβλιοθηκής παρουσιάζονται αναλυτικά παρακάτω:

Ρουτίνες εκτέλεσης χρόνου βιβλιοθηκών (1/6)

- **acc_get_num_devices**

Επιστρέφει τον αριθμό των συσκευών του συγκεκριμένου τύπου που είναι συνδεδεμένες με τον κεντρικό υπολογιστή. Το όρισμα αναφέρει το είδος της συσκευής που πρέπει να μετρήσει.

Σύνταξη (C ή C++):

- **int acc_get_num_devices(acc_device_t);**

Περιοριμοί:

- Αυτή η ρουτίνα μπορεί να μην καλείται σε block κώδικα parallel ή kernels.

Ρουτίνες εκτέλεσης χρόνου βιβλιοθηκών (2/6)

- **acc_wait**

Περιμένει για την ολοκλήρωση όλων των συναφών ασύγχρονων λειτουργιών.

Σύνταξη (C ή C ++):

- **void acc_wait (int);**
- Το όρισμα πρέπει να είναι ένα όρισμα async όπως έχει προειπωθεί στη δήλωση async.
- Εάν η τιμή αυτή εμφανίστηκε σε μία ή περισσότερες δηλώσεις ασύγχρονης σύνδεσης, η ρουτίνα acc_wait δεν θα επιστρέψει μέχρι να ολοκληρωθεί η τελευταία τέτοια ασύγχρονη λειτουργία.
- Εάν δύο ή περισσότερα νήματα μοιράζονται τον ίδιο επιταχυντή, η ρουτίνα acc_wait θα επιστρέψει μόνο αν έχουν ολοκληρωθεί όλες οι ασύγχρονες επεμβάσεις που έχουν ξεκινήσει από αυτό το νήμα.
- Δεν υπάρχει εγγύηση ότι όλες οι ασύγχρονες λειτουργίες αντιστοίχισης που ξεκινούν από άλλα θέματα έχουν ολοκληρωθεί. Για συμβατότητα με την έκδοση 1.0, αυτή η ρουτίνα μπορεί επίσης να γραφτεί acc_async_wait.

Ρουτίνες εκτέλεσης χρόνου βιβλιοθηκών (3/6)

- **acc_init**

Αναφέρει το χρόνο εκτέλεσης για την αρχικοποίηση του χρόνου εκτέλεσης για αυτόν τον τύπο συσκευής. Αυτό μπορεί να χρησιμοποιηθεί για την απομόνωση οποιουδήποτε κόστους αρχικοποίησης από το υπολογιστικό κόστος, κατά τη συλλογή στατιστικών απόδοσης. Η συνάρτηση acc_init επίσης καλεί έμμεσα. την acc_set_device_type

- **Σύνταξη (C ή C ++):**

void acc_init (acc_device_t);

- **Περιορισμοί**
- Αυτή η ρουτίνα μπορεί να μην καλείται σε block κώδικα parallel ή kernels.
- Εάν ο συγκεκριμένος τύπος συσκευής δεν είναι διαθέσιμος, η συμπεριφορά ορίζεται από την εφαρμογή. Ειδικότερα, το πρόγραμμα μπορεί να τερματιστεί.
- Εάν η ρουτίνα καλείται περισσότερες από μία φορές χωρίς παρεμβαλλόμενη κλήση acc_shutdown, με διαφορετική τιμή για το όρισμα τύπου συσκευής, η συμπεριφορά καθορίζεται από την εφαρμογή.
- Εάν ορισμένα block κώδικα του επιταχυντή έχουν συνταχθεί για να χρησιμοποιούν μόνο έναν τύπο συσκευής, η κλήση αυτής της ρουτίνας με διαφορετικό τύπο συσκευής μπορεί να προκαλέσει απροσδιόριστη συμπεριφορά.

Ρουτίνες εκτέλεσης χρόνου βιβλιοθηκών (4/6)

- **acc_shutdown**

Αναφέρει το χρόνο εκτέλεσης για να τερματίσει τη σύνδεση με τη δεδομένη συσκευή επιτάχυνσης και να ελευθερώσει τους πόρους του χρόνου εκτέλεσης.

- **Σύνταξη (C ή C ++):**

void acc_shutdown (acc_device_t);

- Η ρουτίνα acc_shutdown αποσυνδέει το πρόγραμμα από τη συσκευή επιτάχυνσης.
- **Περιορισμοί**
- Αυτή η ρουτίνα μπορεί να μην καλείται κατά την εκτέλεση block κώδικα του επιταχυντή.

Ρουτίνες εκτέλεσης χρόνου βιβλιοθηκών (5/6)

- **acc_malloc**

Εκχωρεί μνήμη στη συσκευή επιτάχυνσης.

Σύνταξη (C ή C ++):

d_void * acc_malloc (size_t);

- Η ρουτίνα acc_malloc μπορεί να χρησιμοποιηθεί για την εκχώρηση μνήμης στη συσκευή επιτάχυνσης. Οι δείκτες που εκχωρούνται από αυτή τη συνάρτηση μπορούν να χρησιμοποιηθούν στις δηλώσεις deviceptr για να ενημερώσουν τον μεταγλωττιστή ότι στόχος του δείκτη είναι να παραμείνει στον επιταχυντή.

Ρουτίνες εκτέλεσης χρόνου βιβλιοθηκών (6/6)

- **acc_free**

Απελευθερώνει τη μνήμη στη συσκευή επιτάχυνσης.

- **Σύνταξη (C ή C ++):**

void acc_free (d_void *);

- Η ρουτίνα acc_free θα ελευθερώσει τη μνήμη που είχε προηγουμένως διατεθεί στη συσκευή επιταχυντή. Το όρισμα θα πρέπει να είναι μια τιμή δείκτη που επιστράφηκε από μια κλήση της acc_malloc

Προχωρημένη Δυναμική ανάλυση του κώδικα του OpenACC- με χρήση του εργαλείου PGPROF

Τι είναι το εργαλείο PGPROF; (1/2)

- Το PGPROF είναι ένα εργαλείο προχωρημένης ανάλυσης κώδικα που διανέμεται με την ακολουθία εργασίας του PGI Workstation. Μπορεί να χρησιμοποιηθεί για την ανάλυση της απόδοσης των προγραμμάτων και υποστηρίζει τα προγράμματα σε OpenMPI και MPI.
- Πρόκειται για ένα εργαλείο που αναλύει τα δεδομένα που δημιουργούνται κατά την εκτέλεση του κώδικα σε Fortran, C και C ++ που έχει μεταγλωττιστεί με επιλογές για προχωρημένη ανάλυση.

Τι είναι το εργαλείο PGPROF; (1/2)

- Εμφανίζει πληροφορίες σχετικά με τις εκτελέσιμες ρουτίνες και γραμμές, πόσο συχνά εκτελέστηκαν και πόσο χρόνο κατανάλωσαν από το συνολικό χρόνο. Οι κώδικες θα πρέπει να μεταγλωττίζονται με όλες τις σημαίες βελτιστοποίησης που θέλετε να χρησιμοποιήσετε, εκτός από τις ειδικές επιλογές προχωρημένης ανάλυσης.
- Είναι συμβατό με σύστημα 64-bit Linux.
- **Συνήθεις χρησιμοποιούμενες επιλογές για προχωρημένη ανάλυση:**
- **-Mprof = func** Παράγει ένα αρχείο pgprof.out με δεδομένα σχεδίασης επιπέδου λειτουργίας.
- **-Mprof = lines** Παράγει ένα αρχείο pgprof.out με δεδομένα γραμμής.
- **-Mprof = time** Παράγει ένα αρχείο pgprof.out με στατιστική δειγματοληψία βασισμένη σε χρόνο.
- **-pg** Παράγει ένα αρχείο gmon.out με δεδομένα ρουτίνας, γραμμής και οδηγίες.

PGPROF: OpenACC CPU και GPU Profiler

- Για τα συστήματα πολλαπλών επεξεργαστών 64 bit με ή χωρίς επιταχυντές:

• Υποστηρίζει OpenMP profiling σε επίπεδο νήματος.

• Υποστηρίζει profiling για OpenACC και CUDA Fortran κωδίκων σε επιταχυντές GPU της NVIDIA με υποστήριξη CUDA.

PGPROF: OpenACC CPU και GPU Profiler

- Διεπαφές χρήστη γραφικών και γραμμής εντολών.

- Επίπεδο λειτουργίας (ρουτίνα) και profiling ανά επίπεδο γραμμής του πηγαίου κώδικα.

- Πλήρως ενσωματωμένες εγκαταστάσεις βοήθειας.

Λειτουργία του PGPROF

- Υπάρχουν δύο τρόποι δημιουργίας προχωρημένης δυναμικής ανάλυσης:
  - *Σε γραμμή εντολών και Γραφική αναπαράσταση δυναμικής ανάλυσης*.
- Μπορείτε να εισάγετε τη λειτουργία γραμμής εντολών ανοίγοντας ένα κέλυφος και εισάγοντας τις εντολές που ακολουθούν παρακάτω.
- Μπορείτε να εισάγετε τη γραφική παράσταση πληκτρολογώντας pgprof σε ένα κέλυφος εντολών.
- Η χρήση του PGI Profiler αποτελείται από δύο βασικά βήματα: Δυναμική Ανάλυση της εφαρμογής σας και στη συνέχεια ανάλυση του προφίλ. Και τα δύο βήματα μπορούν να πραγματοποιηθούν σε κάθε λειτουργία.
- Στη λειτουργία γραμμής εντολών χρησιμοποιούνται δύο διακριτές εντολές για την δυναμική ανάλυση και την ανάλυση του προφίλ.
- Σε γραφική παράσταση, τόσο η δημιουργία προφίλ όσο και η ανάλυση μπορούν να πραγματοποιηθούν στην ίδια συνεδρία.
- Επιπλέον, είναι δυνατό να αποθηκεύσετε ένα προφίλ είτε στη γραμμή εντολών είτε στο γραφικό τρόπο. Το αποθηκευμένο προφίλ μπορεί να αναλυθεί στο γραφικό τρόπο χρησιμοποιώντας το File | Import option.

Η λειτουργία της γραμμής εντολών

- **Πόσο χρόνο χρειάζεται η εφαρμογή μου να τρέξει;**

- Δυναμική ανάλυση του a.out και αποθηκεύστε τα αποτελέσματα απόδοσης στο αρχείο a.prof με την εντολή:

  $ pgprof -o a.prof a.out

- Η επιλογή του pgprof -cpu-profile-mode top-down στρέφει το δέντρο κλήσεων για να εμφανίσει την main στην κορυφή και τις λειτουργίες που καλεί παρακάτω με την εντολή:

  $ pgprof --cpu-profiling-mode top-down -i a.prof

Η λειτουργία της γραμμής εντολών

- **Πώς μπορώ να απεικονίσω τόσο τα δεδομένα απόδοσης της CPU και GPU;**
- Δυναμική ανάλυση του a.out και αποθηκεύστε τα αποτελέσματα απόδοσης στο αρχείο a.prof με την εντολή:

$ pgprof -o a.prof a.out

- Στη συνέχεια, εμφανίστε τα περιεχόμενα του αρχείου εξόδου με την εντολή:

$ pgprof -i a.prof

- Τα αποτελέσματα χωρίζονται σε τέσσερα τμήματα:

1. Προφίλ εκτέλεσης πυρήνα GPU.
2. Προφίλ εκτέλεσης API CUDA.
3. Προφίλ εκτέλεσης OpenACC.
4. Προφίλ εκτέλεσης CPU.

Η λειτουργία της γραμμής εντολών

1
```
====== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 38.14%  1.41393s        20  70.696ms  70.666ms  70.731ms  calc2_198_gpu
 31.11%  1.15312s        18  64.062ms  64.039ms  64.083ms  calc3_273_gpu
 23.35%  865.68ms        20  43.284ms  43.244ms  43.325ms  calc1_142_gpu
  5.17%  191.78ms       141  1.3602ms  1.3120us  1.6409ms  [CUDA memcpy HtoD]
...
```

2
```
======== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 92.65%  3.49314s        62  56.341ms  1.8850us  70.771ms  cuStreamSynchronize
  3.78%  142.36ms         1  142.36ms  142.36ms  142.36ms  cuDevicePrimaryCtxRetain
...
```

3
```
======== OpenACC (excl):
Time(%)      Time     Calls       Avg       Min       Max  Name
 36.27%  1.41470s        20  70.735ms  70.704ms  70.773ms  acc_wait@swim-acc-data.f:223
 29.60%  1.15449s        18  64.138ms  64.114ms  64.159ms  acc_wait@swim-acc-data.f:302
 22.22%  866.66ms        20  43.333ms  43.294ms  43.376ms  acc_wait@swim-acc-data.f:169
  9.06%  353.49ms         1  353.49ms  353.49ms  353.49ms  acc_update@swim-acc-data.f:402
...
```

4
```
======== CPU profiling result (bottom up):
Time(%)      Time  Name
 59.09%  8.55785s  cudbgGetAPIVersion
 59.09%  8.55785s  | start_thread
 59.09%  8.55785s  |   clone
 25.75%  3.73007s  cuStreamSynchronize
 25.75%  3.73007s  | __pgi_uacc_cuda_wait
 25.75%  3.73007s  |   __pgi_uacc_computedone
 10.38%  1.50269s  |     swim_mod_calc2_
 10.38%  1.50269s  |     | MAIN_
 10.38%  1.50269s  |     |   main
  8.54%  1.23625s  |     swim_mod_calc3_
  8.54%  1.23625s  |     | MAIN_
  8.54%  1.23625s  |     |   main
  6.48%  937.85ms  |     swim_mod_calc1_
  6.48%  937.85ms  |     | MAIN_
  6.48%  937.85ms  |     |   main
  0.37%  53.287ms  |     swim_mod_calc3z_
  0.37%  53.287ms  |       MAIN_
  0.37%  53.287ms  |         main
  6.03%   873.9ms  swim_mod_inital_
...
```

1. Προφίλ εκτέλεσης πυρήνα GPU.
2. Προφίλ εκτέλεσης API CUDA.
3. Προφίλ εκτέλεσης OpenACC.
4. Προφίλ εκτέλεσης CPU.

Η λειτουργία της γραμμής εντολών

- **Ποιες πληροφορίες μπορεί να πει ο μεταγλωττιστής σχετικά με τον τρόπο δομής της εφαρμογής μου για απόδοση;**
- Ο Profiler PGI μπορεί να σας δείξει πληροφορίες σχετικά με τον τρόπο σύνταξης του προγράμματός σας. Πρώτα προσθέστε την ακόλουθη επιλογή κατά τη σύνταξη και τη σύνδεση:

  -Minfo=ccff

- Δυναμική ανάλυση του a.out και αποθηκεύστε τα αποτελέσματα απόδοσης στο αρχείο a.prof με την εντολή:

  $ pgprof -o a.prof a.out

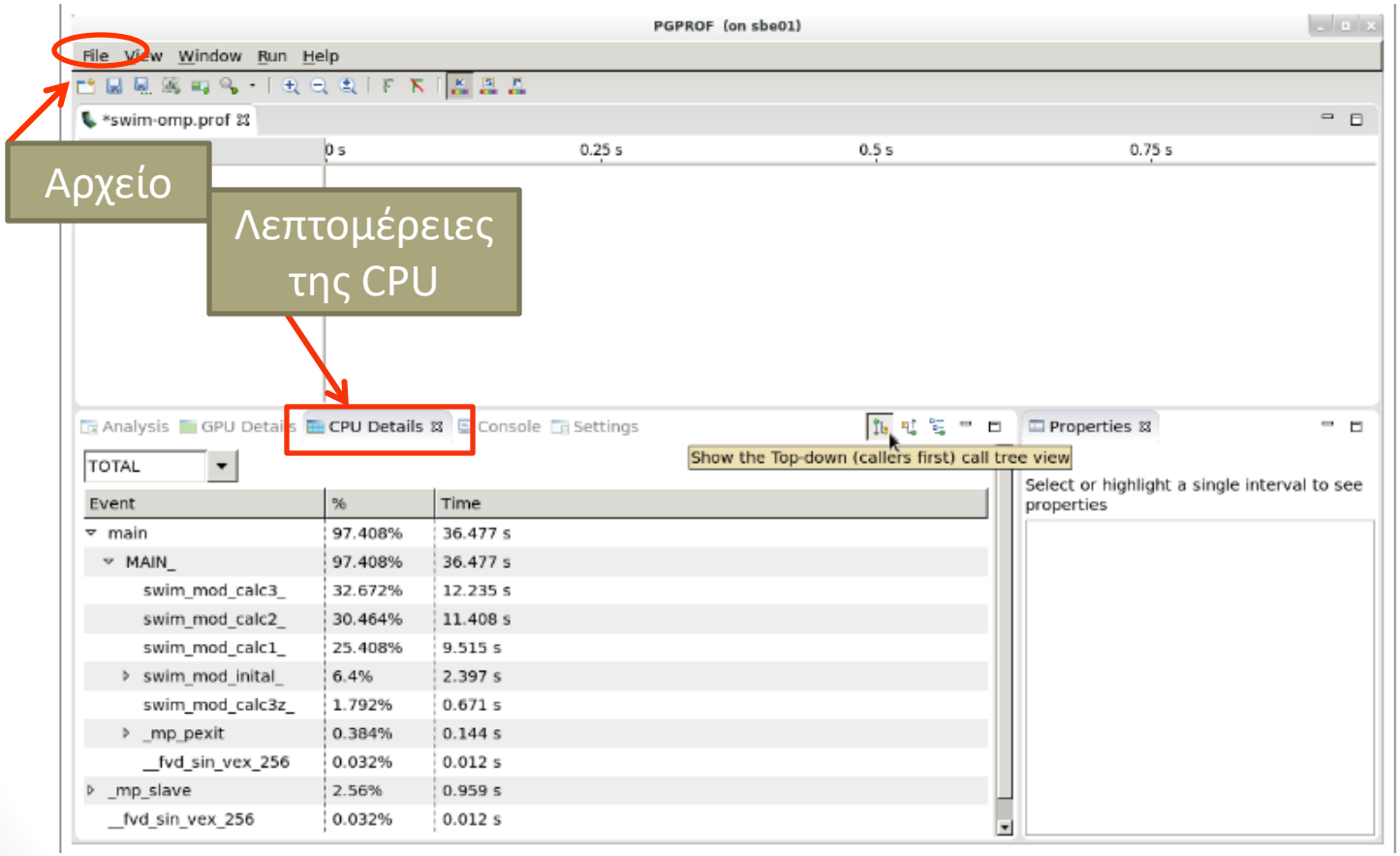
- Τώρα εμφανίστε το προφίλ με τις πληροφορίες CCFF και καταγράψτε τα αποτελέσματα απόδοσης με βάση τη γραμμή με την εντολή:

  $ pgprof --cpu-profiling-show-ccff on -i a.prof

Η λειτουργία γραφικής αναπαράστασης

- **Πόσο χρόνο χρειάζεται η εφαρμογή μου να τρέξει;**
- Για να ξεκινήσετε μια νέα περίοδο δημιουργίας δυναμικής ανάλυσης, μετά την εκκίνηση του Profiler PGI, ανοίξτε το μενού File και επιλέξτε New Session.
- Στο παράθυρο διαλόγου, μεταβείτε στο εκτελέσιμο αρχείο που θέλετε να αναλύσετε. Στη συνέχεια, προσθέστε οποιαδήποτε ορίσματα γραμμής εντολών για να ξεκινήσετε.
- Κάντε κλικ στο κουμπί Next και μετά στο Finish.
- Στην καρτέλα “CPU Details", κάντε κλικ στο κουμπί “Show the top-down", όπως φαίνεται στο εικόνα παρακάτω.

Η λειτουργία γραφικής αναπαράστασης

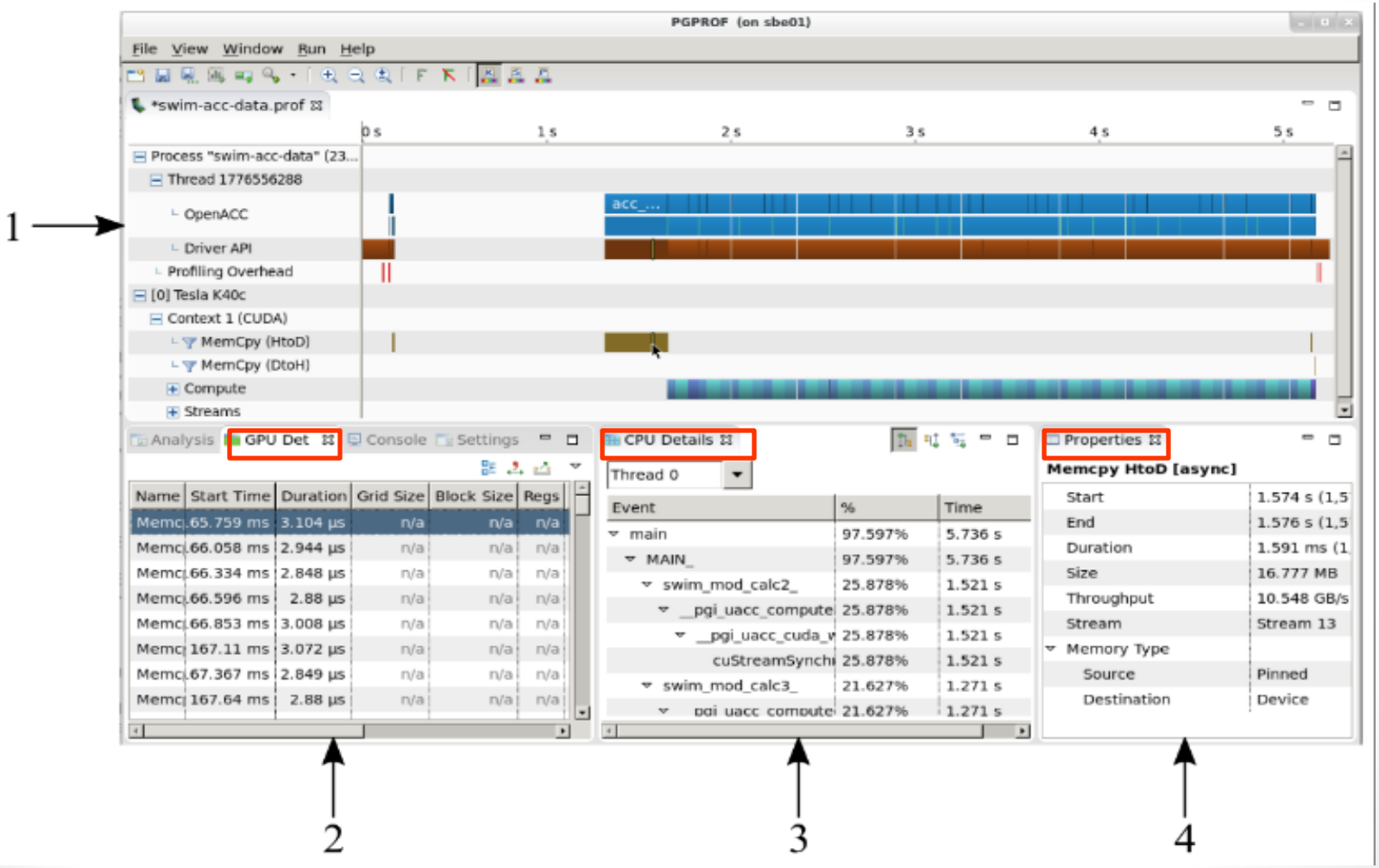


Η λειτουργία γραφικής αναπαράστασης

- **Πώς μπορώ να απεικονίσω τόσο τα δεδομένα απόδοσης CPU και GPU;**
- Για να ξεκινήσετε μια νέα περίοδο δημιουργίας δυναμικής ανάλυσης, μετά την εκκίνηση του Profiler PGI, ανοίξτε το μενού File και επιλέξτε New Session.
- Στο παράθυρο διαλόγου, μεταβείτε στο εκτελέσιμο αρχείο που θέλετε να προφίλ. Στη συνέχεια, προσθέστε οποιαδήποτε επιχειρήματα γραμμής εντολών για να το ξεκινήσετε.
- Κάντε κλικ στο κουμπί Next και μετά στο Finish.

1. Η προβολή “Timeline" θα εμφανίσει τα προγραμματισμένα συμβάντα από τη στιγμή που συνέβησαν.
2. Στην καρτέλα “GPU Details" παρατίθενται οι λεπτομέρειες της απόδοσης για κάθε πυρήνα GPU.
3. Στην καρτέλα “CPU Details” εμφανίζεται η δέντρο κλήσεων της CPU.
4. Η καρτέλα “Properties” εμφανίζει τις λεπτομέρειες των συμβάντων που επιλέχθηκαν στο χρονικό πλαίσιο.

Η λειτουργία γραφικής αναπαράστασης



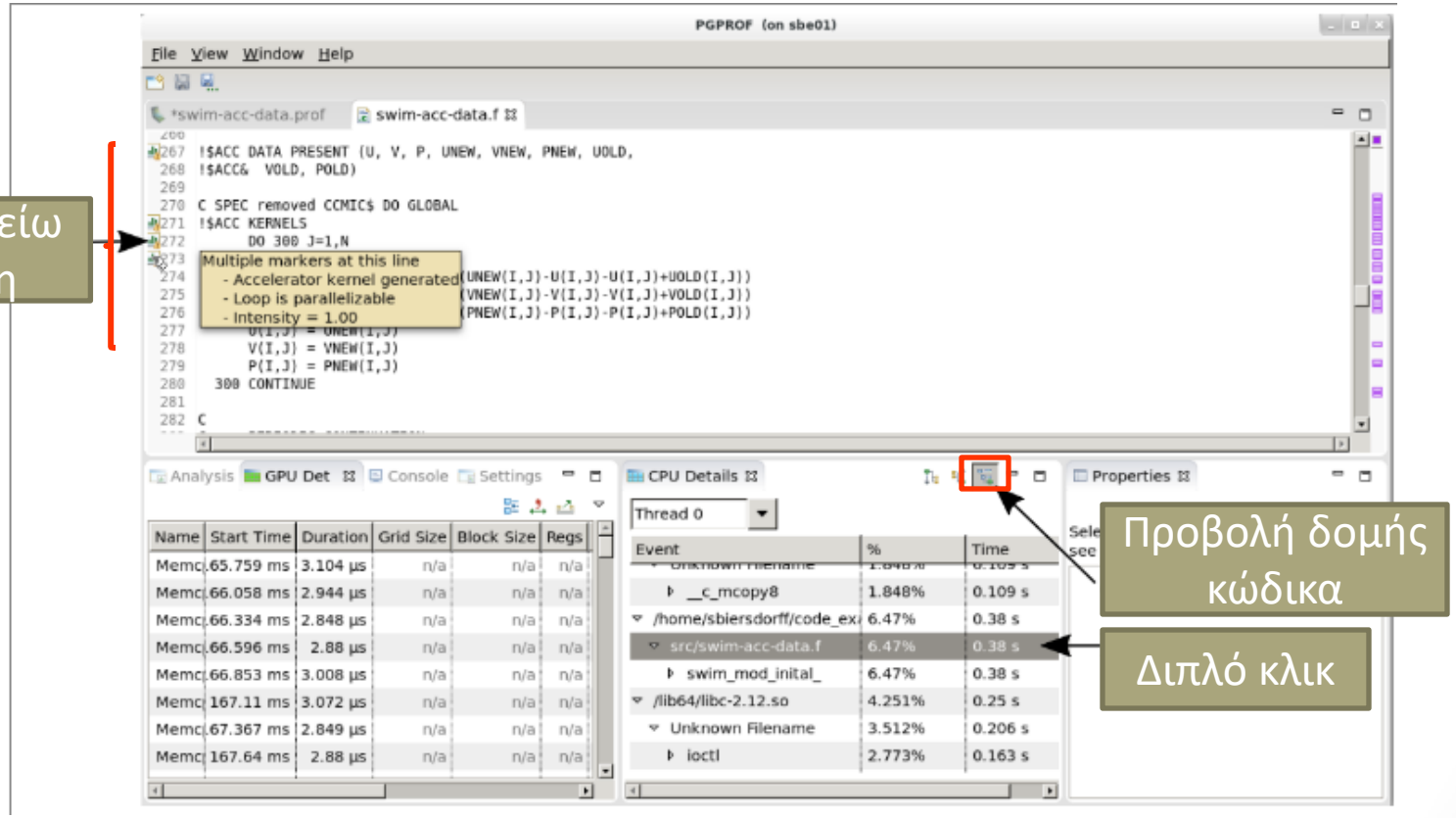
Η λειτουργία γραφικής αναπαράστασης

- **Ποιές πληροφορίες μπορεί να πει ο μεταγλωττιστής σχετικά με τον τρόπο δομής της εφαρμογής μου για απόδοση;**
- Ο Profiler PGI μπορεί να σας παρέχει πληροφορίες σχετικά με τον τρόπο σύνταξης του προγράμματός σας. Πρώτα προσθέστε την ακόλουθη επιλογή κατά τη σύνταξη και τη σύνδεση:

  -Minfo=ccff

- Στη συνέχεια, αναλύστε δυναμικά.
- Για να ξεκινήσετε μια νέα περίοδο δημιουργίας προφίλ, μετά την εκκίνηση του Profiler PGI, ανοίξτε το μενού File και επιλέξτε New Session.
- Στο παράθυρο διαλόγου, μεταβείτε στο εκτελέσιμο αρχείο που θέλετε να προφίλ. Στη συνέχεια, προσθέστε οποιαδήποτε ορίσματα στη γραμμή εντολών για να ξεκινήσετε.
- Κάντε κλικ στο κουμπί Next και μετά στο Finish.
- Στην καρτέλα Λεπτομέρειες επεξεργαστή (CPU), κάντε κλικ στο κουμπί "Εμφάνιση της προβολής δομής κώδικα".
- Κάντε διπλό κλικ σε μια καταχώρηση αρχείου, το όνομα της λειτουργίας ή τον αριθμό γραμμής για να εμφανιστεί αυτό το αρχείο ή συγκεκριμένη γραμμή στην προβολή του πηγαίου κώδικα.
- Στα αριστερά των αριθμών γραμμών στην προβολή πηγαίου κώδικα θα δείτε μια σειρά σημειώσεων μεταγλωττιστή. Τοποθετήστε το ποντίκι πάνω από μια σημείωση για να εμφανιστεί η ανατροφοδότηση που παρέχεται από τον μεταγλωττιστή.

Η λειτουργία γραφικής αναπαράστασης



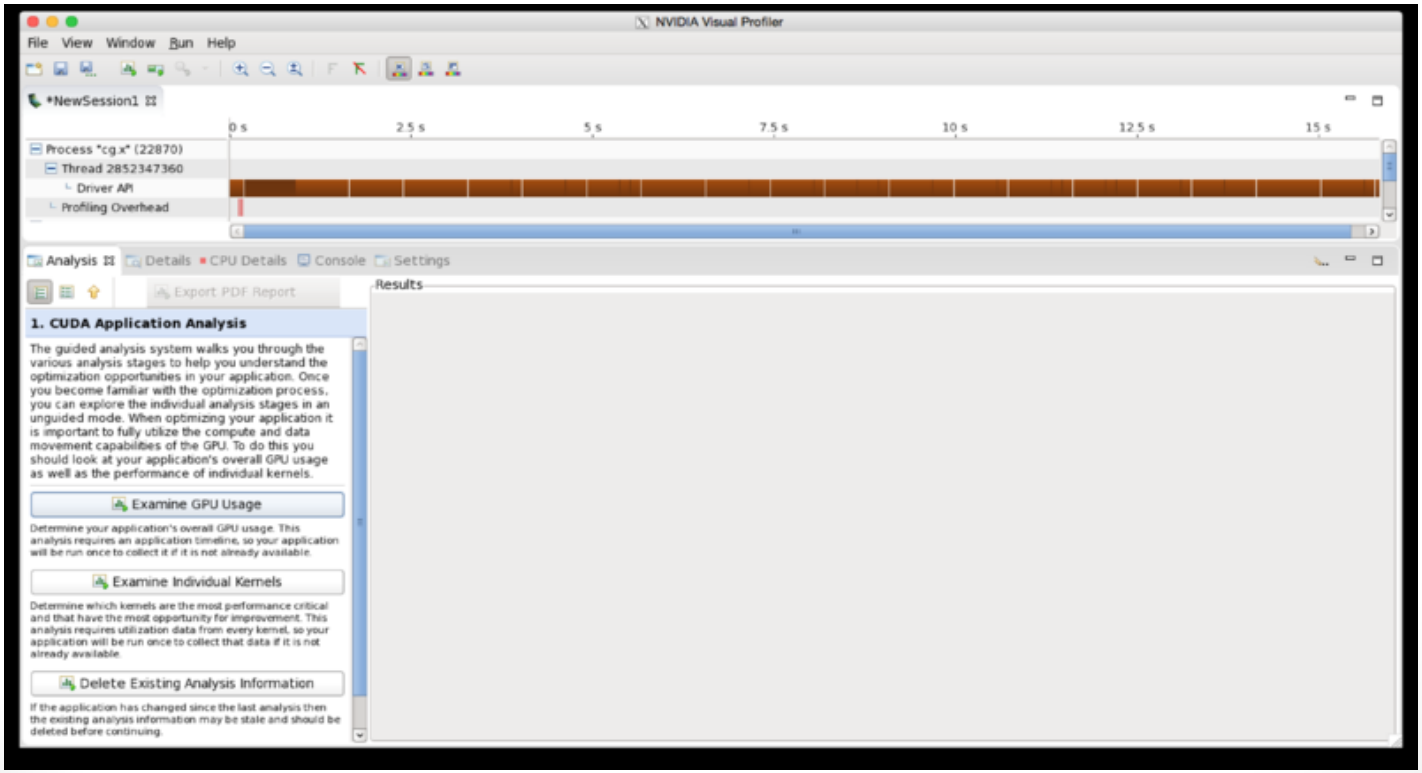
Προχωρημένη Δυναμική ανάλυση του κώδικα του OpenACC- με χρήση του εργαλείου Visual Profiler της NVIDIA

Τι είναι το Visual Profiler;

- Το Visual Profiler είναι ένα εργαλείο γραφικής προχωρημένης ανάλυσης το οποίο εμφανίζει μια χρονική γραμμή της δραστηριότητας της CPU και της GPU της εφαρμογής σας και περιλαμβάνει μια αυτοματοποιημένη μηχανή ανάλυσης για τον εντοπισμό ευκαιριών βελτιστοποίησης.
- Το εργαλείο nvprof σας επιτρέπει να συλλέγετε και να προβάλλετε δεδομένα δημιουργίας προφίλ από την γραμμή εντολών. Το NVIDIA Visual Profiler είναι ένα εργαλείο ανάλυσης πολλαπλών επιδόσεων που παρέχει στους προγραμματιστές ζωτικής σημασίας ανατροφοδότηση για τη βελτιστοποίηση των εφαρμογών CUDA C / C ++.
- Το Visual Profiler που κυκλοφόρησε για πρώτη φορά το 2008 υποστηρίζει όλα τα 350 εκατομμύρια + GPUs NVIDIA με δυνατότητα CUDA που έχουν αποσταλεί από το 2006 σε Linux, Mac OS X και Windows και είναι είναι διαθέσιμο ως μέρος του CUDA Toolkit.

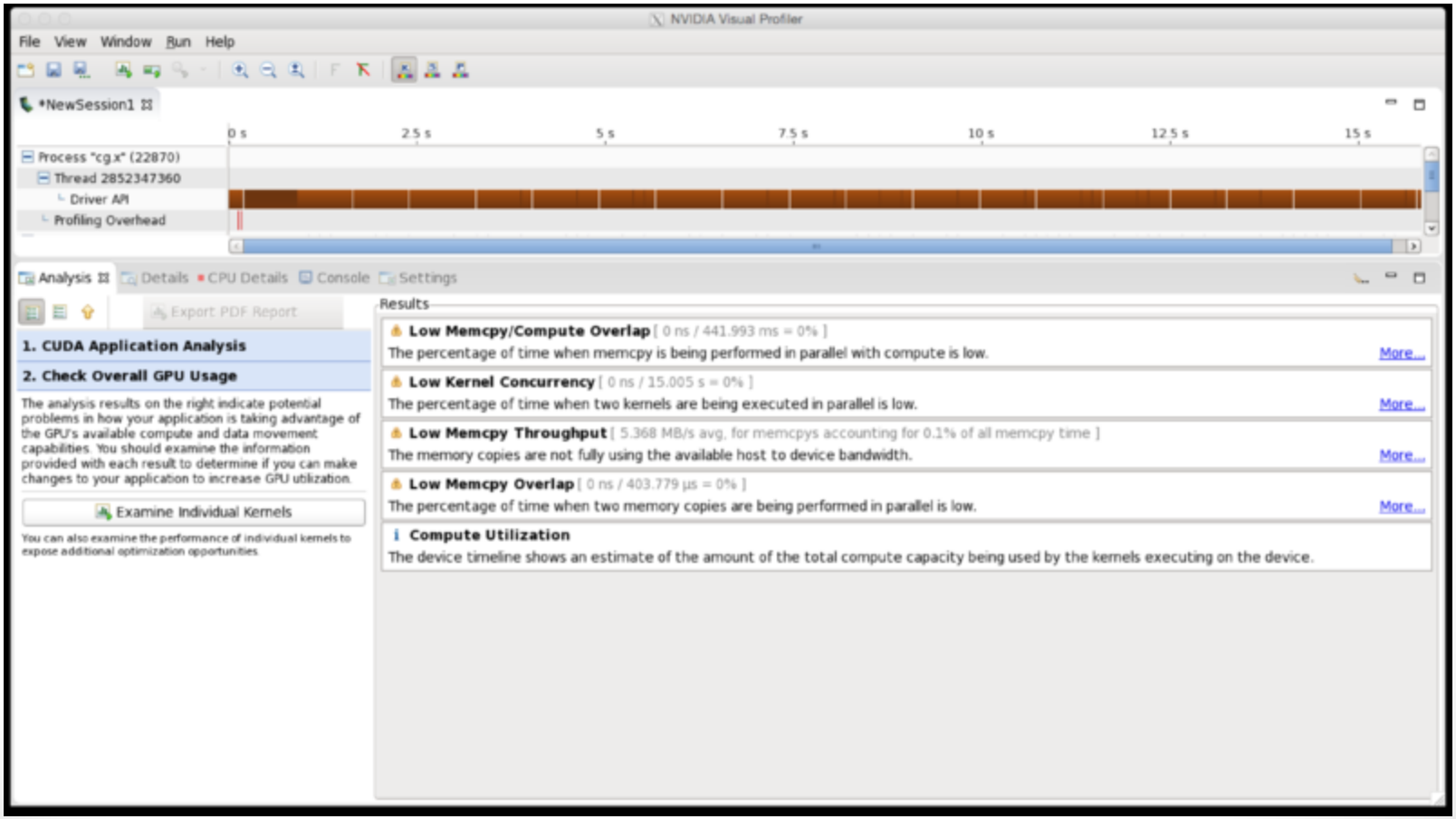
Η χρήση του Visual Profiler σε βήματα (1/4)

1. Μεταβείτε στην καρτέλα “Analysis" και κάντε κλικ στο “Examine GPU Usage". Μόλις εκτελεστεί η ανάλυση, ο profiler σας δίνει μια σειρά προειδοποιήσεων. Αυτό σας δίνει ενδείξεις σχετικά με το τι μπορεί να βελτιωθεί.



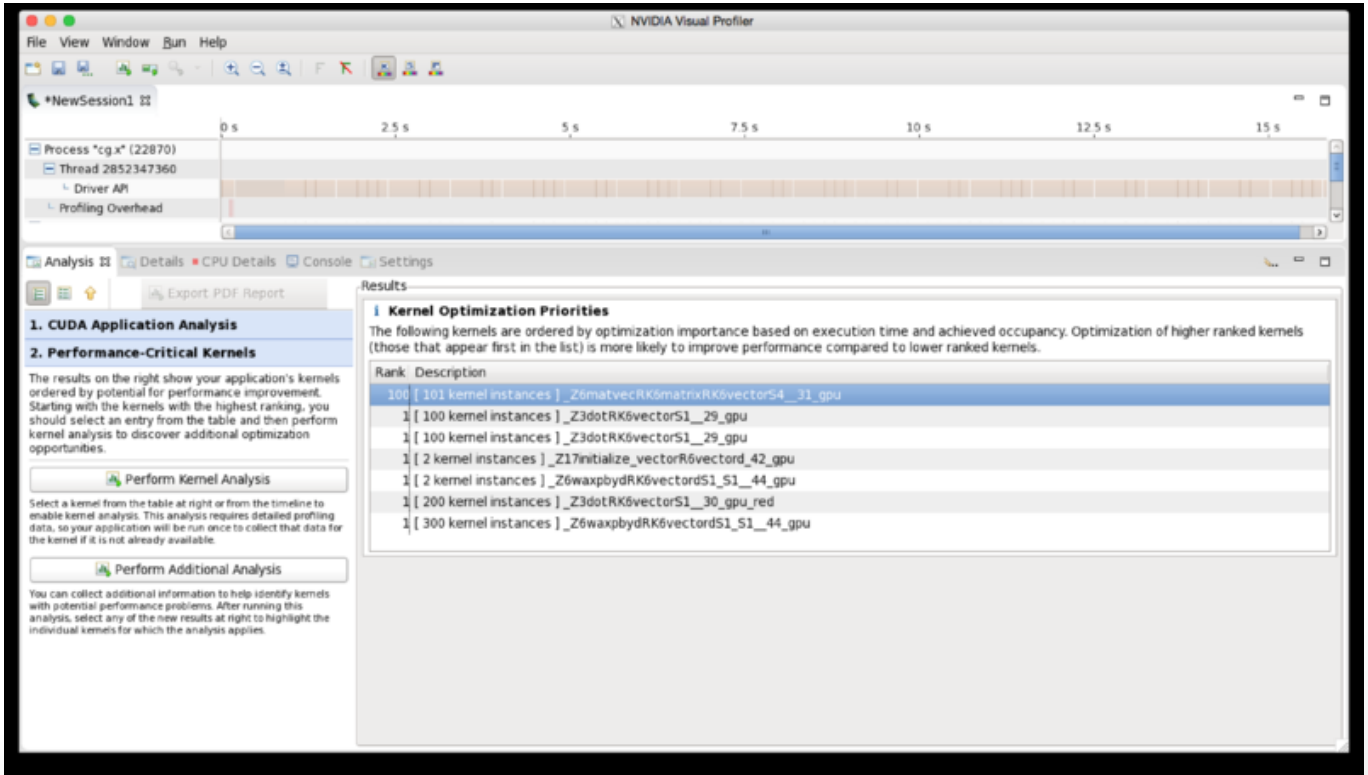
Η χρήση του Visual Profiler σε βήματα (2/4)

2. Στη συνέχεια, κάντε κλικ στο “Examine Individual Kernels". Αυτό θα σας δείξει μια λίστα πυρήνων.



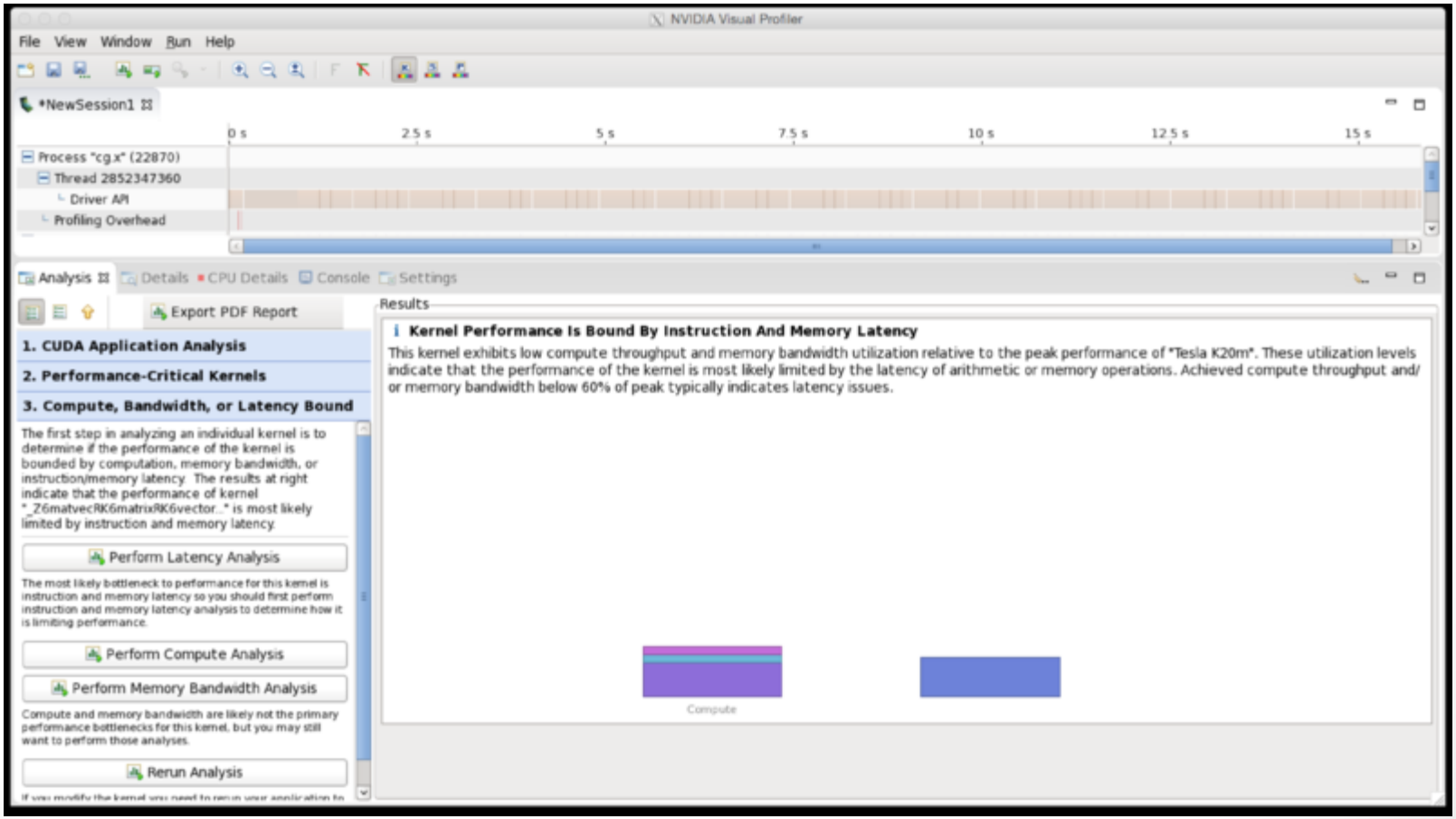
Η χρήση του Visual Profiler σε βήματα (3/4)

3. Επιλέξτε το πρώτο και κάντε κλικ στην επιλογή “Perform Kernel Analysis". Ο profiler θα σας δείξει μια λεπτομερέστερη ανάλυση αυτού του συγκεκριμένου πυρήνα, υπογραμμίζοντας το πιο πιθανό σημείο συμφόρησης. Στην περίπτωση αυτή, η απόδοση περιορίζεται από την καθυστέρηση μνήμης.

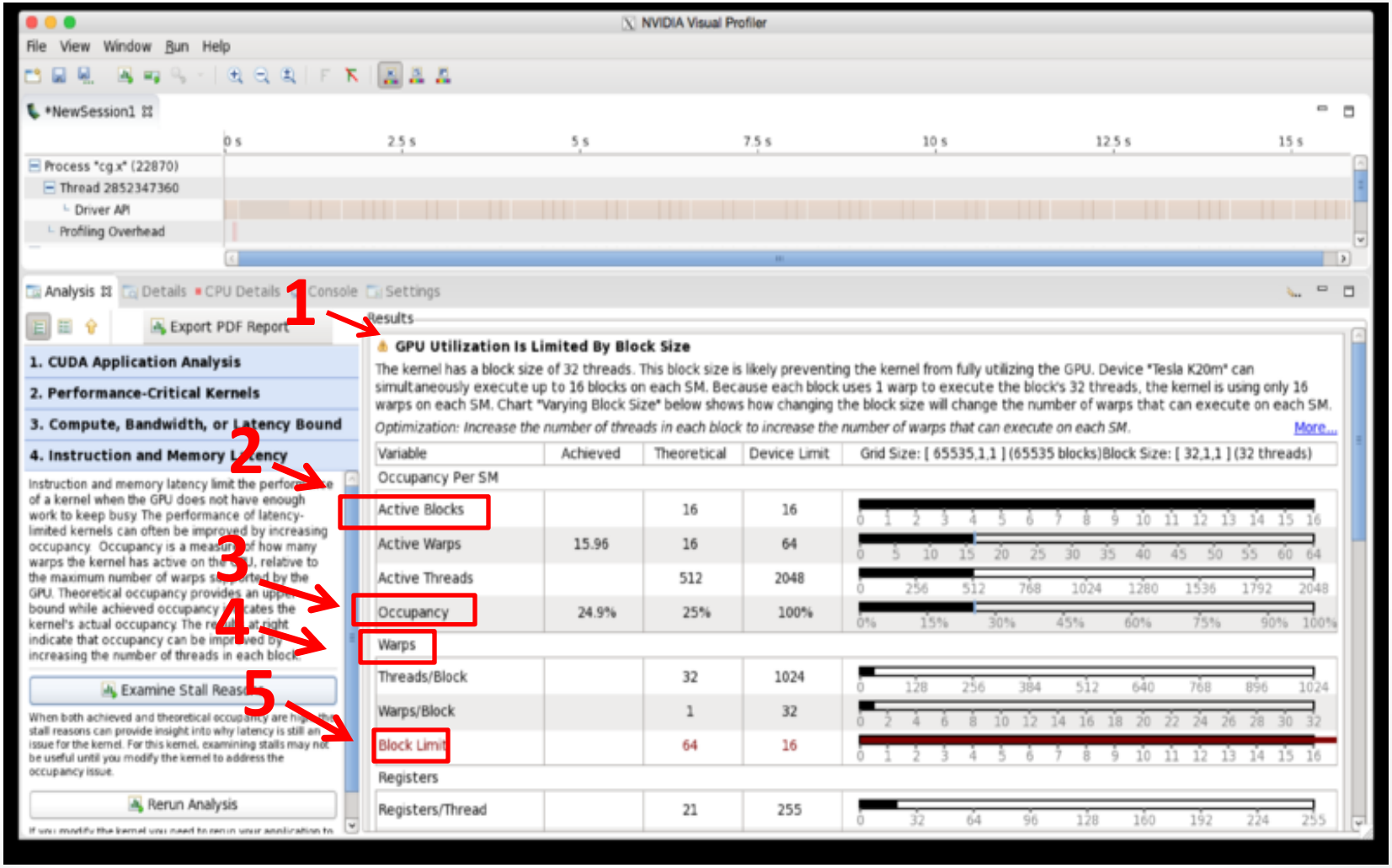


Η χρήση του Visual Profiler σε βήματα (4/4)

4. Κάντε κλικ στο “Perform Latency Analysis”.



Η μορφή των αποτελεσμάτων



Η επεξήγηση των αποτελεσμάτων γίνεται σε επόμενη διαφάνεια.

Επεξήγηση των αποτελεσμάτων (1/2)

1. Το κείμενο μας λέει σαφώς ότι η απόδοση περιορίζεται από το μέγεθος των μπλοκ, το οποίο στο OpenACC αντιστοιχεί στο μέγεθος των gang.
2. Η γραμμή “Active Threads" μας λέει ότι η GPU τρέχει 512 σπειρώματα, ενώ θα μπορούσε να τρέξει το 2048.
3. Η γραμμή πληρότητας δηλώνει αντίστοιχα ότι η GPU χρησιμοποιείται μόνο στο 25% της χωρητικότητάς της. Η κατοχή είναι η αναλογία πόση GPU χρησιμοποιείται για το πόσο θα μπορούσε να χρησιμοποιηθεί η GPU. Σημειώστε ότι η πληρότητα 100% δεν αποφέρει απαραίτητα την καλύτερη απόδοση. Ωστόσο, το 25% είναι αρκετά χαμηλό.

Επεξήγηση των αποτελεσμάτων (2/2)

4. Οι πιο σημαντικές απαντήσεις προέρχονται από τον πίνακα "Warps". Αυτός ο πίνακας μας λέει ότι η GPU τρέχει 32 νήματα ανά μπλοκ (OpenACC: διανυσματικά νήματα -vector threads- ανά gang) ενώ μπορεί να τρέχει 1024. Επίσης, μας λέει ότι τρέχει 1 warp ανά μπλοκ (OpenACC: worker ανά gang), ενώ θα μπορούσε να τρέξει 32.

5. Τέλος, η τελευταία γραμμή μας λέει ότι για να γεμίσουμε τη συσκευή θα έπρεπε να τρέξουμε 64 gangs, αλλά η συσκευή μπορεί να κρατήσει μόνο 16. Το συμπέρασμα είναι ότι χρειαζόμαστε μεγαλύτερα gangs. Μπορούμε να το κάνουμε αυτό με την προσθήκη περισσότερων workers διατηρώντας παράλληλα το μέγεθος του vector σε 32.

Συμπεράσματα Ενότητας

- Το OpenACC είναι ένας εύκολος και γρήγορος τρόπος για να χρησιμοποιήσετε τις GPU για να εκτελέσετε τον κώδικα πιο γρήγορα.
- Η εξάρτηση δεδομένων και η κίνηση δεδομένων είναι δύο σημαντικές προκλήσεις για να επιτύχει μια καλή επιτάχυνση του κώδικα που βασίζεται σε GPU.
- Η ομοιότητα του με το OpenMP ως προς τον τρόπο σύνταξής του το καθιστά εύκολο στην εκμάθηση και την γραφή του κώδικα.
- Οδηγίες υψηλότερου επιπέδου επιτρέπουν τη στόχευση ίδιου κώδικα σε διαφορετικές αρχιτεκτονικές.
- Ιδιαίτερα ευεργετικό για τους χρήστες που δεν είναι ειδικοί στο GPGPU.

Βιβλιογραφία

- *The OpenACC™ Application Programming Interface, Version 2.0 June, 2013,Corrected, August, 2013*
- *OPENACC® DIRECTIVES FOR ACCELERATORS ,NVIDIA*
- *Advanced OpenACC,John Urbanic, Parallel Computing Scientist, Pittsburgh Supercomputing Center*
- *OpenACC Programming and Best Practices Guide, June 2015*
- *Introduction to OpenACC, Shaohao Chen, Research Computing Services, Information Services and Technology, Boston University*
- *Introduction to OpenACC, Jeff Larkin*
- *ADVANCED OPENACC COURSE, Lecture 1: Advanced Profiling, May 19, 2016*
- *OpenACC, From Wikipedia , the free encyclopedia*
- *OpenACC, gcc.gnu.org/wiki/*

Βιβλιογραφία

- *docs.computecanada.ca/wiki/OpenACC_Tutorial_-_Optimizing_loops*
- *www.pgroup.com*
- *developer.nvidia.com*
- *docs.nvidia.com*
- */www.westgrid.ca*
- *http://www.fz-juelich.de*

Τέλος ενότητας