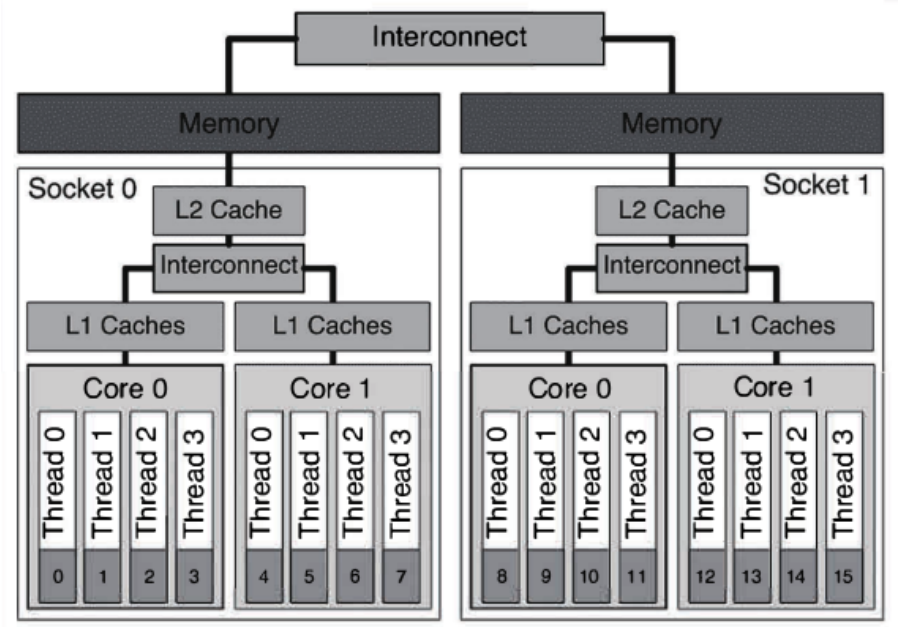
Thread Affinity

Thread Affinity είναι μια έννοια που περιλαμβάνει την βελτιστοποίηση του χρόνου εκτέλεσης ενός προγράμματος, μέσω βελτιστοποιήσεων στο εύρος ζώνης μνήμης, την αποφυγή καθυστέρησης μνήμης ή της καθυστέρησης χρήσης προσωρινής μνήμης.

Το *OpenMP 4.0* εισάγει ένα σύνολο οδηγιών για την υποστήριξη του *thread affinity*[1]. Η πλειοψηφία πλέον των μηχανημάτων βασίζονται στην *cc-NUMA* αρχιτεκτονική. Ο λόγος που αυτό το σύστημα μνήμης έγινε κυρίαρχο, είναι η συνεχής αύξηση του αριθμού των επεξεργαστών. Η μονολιθική διασύνδεση μνήμης με σταθερό εύρος ζώνης μνήμης θα αποτελούσε πρόβλημα στην ραγδαία αύξηση των επεξεργαστών.

Στη *cc-NUMA* αρχιτεκτονική κάθε υποδοχή συνδέεται με ένα υποσύνολο της συνολικής μνήμης του συστήματος. Μία διασύνδεση ενώνει τα υποσύνολα μεταξύ τους και δημιουργεί την εικόνα ενιαιας μνήμης στον χρήστη. Ενα τέτοιο σύστημα είναι ευκολότερο να επεκταθεί.

Το πλεονέκτημα της διασύνδεσης είναι ότι η εφαρμογή έχει πρόσβαση σε όλη την μνήμη του συστήματος, ανεξάρτητα από το που βρίσκονται τα δεδομένα. Ωστόσο, πλεον ο χρόνος πρόσβασης σε αυτά δεν ειναι ο σταθερός καθώς εξαρτάται από τη θέση τους στη μνήμη.



Σχήμα 1: Αρχιτεκτονική cc-NUMA[5]

Thread affinity στο OpenMP 4.0

Με το *thread affinity* μπορεί να αποτραπεί η μετάβαση μιας διαδικασίας *MPI* ή ενός νήματος του *OpenMP* σε απομακρυσμένο υλικό. Η μετάβαση αυτή θα μπορούσε να προκαλέσει μείωση της απόδοσης του κώδικα. Η έκδοση 4.0 του *OpenMP* εισήγαγε ρυθμίσεις για το χειρισμό του *affinity* μέσω των μεταβλητών περιβάλλοντος *OMP_PLACES* και *OMP_PROC_BIND*[4].

0.1.1.1 *Thread Binding*

Οι προαναφερθήσες μεταβλητές, μπορούν να καθορίσουν σε θέση στο υλικό θα ανατεθούν τα νήματα μια ομάδας, που δημιουργήθηκε για να εκτελέσει μια διεργασία.

Παράδειγμα: Αν υπάρχουν δύο *sockets* και ορισθεί

$$OMP\_PLACES = sockets$$

, τότε:

- το νήμα 0 θα πάει στο *socket 0*
- το νήμα 1 θα πάει στο *socket 1*
- το νήμα 2 θα πάει στο *socket 2* κοκ.

Επίσης, αν δυο *socket* έχουν συνολικά 16 πυρήνες και ο χρήστης ορίσει

$$OMP\_PLACES = cores$$

και

$$OMP\_PROC\_BIND = close$$

τότε:

- το νήμα 0 θα πάει στον πυρήνα 0 που βρίσκεται στο *socket* 0
- το νήμα 1 θα πάει στον πυρήνα 1 που βρίσκεται στο *socket* 0
- το νήμα 2 θα πάει στον πυρήνα 2 που βρίσκεται στο *socket* 0
- ...
- το νήμα 7 στον πυρήνα 7 του *socket* 0
- το νήμα 8 στον πυρήνα 8 του *socket* 1, κλπ

Το όρισμα *OMP_PROC_BIND* ορίζει τον τρόπο με τον οποίο τα νήματα ανατίθεται στους πόρους. Η επιλογή *OMP_PROC_BIND* = *close* σημαίνει ότι η ανάθεση περνά διαδοχικά στις διαθέσιμες θέσεις. Μια αλλη αποδεκτή τιμή για το *OMP_PROC_BIND* είναι η *spread*. Η λειτουργία της φαίνεται στο παρακάτω παράδειγμα:
Παράδειγμα, για:

$$OMP\_PLACES = cores$$

$$OMP\_PROC\_BIND = spread$$

- το νήμα 0 πάει στον πυρήνα 0, που βρίσκεται στο *socket* 0
- το νήμα 1 πάει στον πυρήνα 8, που βρίσκεται στο *socket* 1
- το νήμα 2 πάει στον πυρήνα 1, που βρίσκεται στο *socket* 0
- ...
- το νήμα 15 πάει στον πυρήνα 15, που βρίσκεται στο *socket* 1

Η επιλογή *OMP_PROC_BIND=master* αναθέται τα νήματα στο ίδιο σημείο που είναι και το κύριο νήμα της ομάδας. Αυτή η επιλογή χρησιμοποιείται όταν δημιουργούνται πολλές ομάδες αναδρομικά.

Εκτός από τις επιλογές *cores* και *sockets* για τη μεταβλητή *OMP_PLACES*, υπάρχει και η *threads* που χρησιμοποιείται σε ειδικές περιπτώσεις αρχιτεκτονικής, δηλαδή σε περιπτώσεις που οι επεξεργαστές περιέχουν νήματα[2].

Το *thread affinity* στην πράξη

Δυστυχώς, δεν υπάρχει καμία καλύτερη επιλογή για τη ρύθμιση της τοποθέτησης νήματος και της συγγένειας. Η επιλογή εξαρτάται από την εκάστοτε εφαρμογή που πρόκειται να εκτελεστεί. Ακόμη, ο χρήστης θα πρέπει να λαμβάνει υπόψη τις εξής παραμέτρους[3]:

- Οι χρόνοι εκτέλεσης μιας εργασίας μπορούν να επηρεαστούν από άλλες εργασίες που εκτελούνται στο ίδιο μηχάνημα εκείνη τη στιγμή και μοιράζονται πρόσβαση στο δίκτυο, τον δίαυλο μνήμης και στην προσωρινή μνήμη.
- Το λειτουργικό σύστημα ενός μηχανήματος, εκτελεί ταυτόχρονα τις δικές του διεργασίες

References

[1] *The Design of OpenMP Thread Affinity (Heidelberg, Berlin, June 2012)*, volume 7312 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2012. Springer.

[2] V. Eijkhout. Openmp topic: Affinity. https://pages.tacc.utexas.edu/~eijkhout/pcse/html/omp-affinity.html#OpenMPthreadaffinitycontrol. Accessed: 2020-06-20.

[3] N. Z. eScience Institute. Thread placement and thread affinity. https://support.nesi.org.nz/hc/en-gb/articles/360000995575-Thread-Placement-and-Thread-Affinity. Accessed: 2020-06-20.

[4] K. A. U. of Science and Technology. Thread affinity with openmp 4.0. https://www.hpc.kaust.edu.sa/tips/thread-affinity-openmp-40. Accessed: 2020-06-20.

[5] C. T. Ruud van der Pas, E. Stotzer. *Using OpenMP. The Next Step: affinity, accelerators, tasking, SIMD*, page 152. The MIT Press, 2017.