Μοντέλα επικοινωνίας μεταξύ νημάτων και αρχέγονες δομές συγχρονισμού: Μια μελέτη περίπτωσης (Working title)

Νεσλεχανίδης Οδυσσέας 15537

June 7, 2019

Θα παρουσιαστούν κοινές αρχέγονες δομές συγχρονισμού (Semaphore, Spinlock, Barrier), και οι εφαρμογές τους στη λύση κλασικών προβλημάτων συγχρονισμού. Θα μελετηθούν πολυνηματικές υλοποιήσεις τους σε κώδικα C. Ακόμη, θα αναλυθούν οι παράμετροι που αφορούν την υλοποίηση μοντέλων επικοινωνίας μεταξύ νημάτων. Τέλος, θα γραφεί πρόγραμμα σε C με χρήση της βιβλιοθήκης pthreads, ως εφαρμογή των παραπάνω στο πρόβλημα του υπολογισμού της μεταφοράς θερμότητας.

Part I
Θεωρία παράλληλου προγραμματισμού συστημάτων

1 Εισαγωγή

Τα προγραμματιστικά εργαλεία που μας επιτρέπουν να εκμεταλλευόμαστε τις δυνατότητες παράλληλης επεξεργασίας των σύγχρονων πολυεπεξεργαστών και συστοιχιών (clusters) επεξεργαστών σχεδιάζονται βάσει μιας ομάδας μοντέλων παράλληλου προγραμματισμού. Ιστορικά, αυτά τα μοντέλα ανταποκρίνονταν πιο άμεσα στη δομή του παράλληλου υλικού. Σήμερα, αποτελούν κυρίως βολικές αφαιρέσεις για την κατηγοριοποίηση των αλγορίθμων που χρησιμοποιούνται στον παράλληλο προγραμματισμό και για τους όρους σύνθεσης και χρήσης τους εντός των προγραμμάτων.[1][2]

Την ίδια στιγμή, η επιλογή του μοντέλου έχει θεμελιώδη ρόλο στις αποφάσεις που λαμβάνονται κατά το σχεδιασμό παράλληλων υπολογιστικών συστημάτων μεγάλης κλίμακας, στον κλάδο της υπολογιστικής υψηλών αποδόσεων (high performance computing).[3]

Τα μοντέλα παράλληλου προγραμματισμού χωρίζονται σε άξονες βάσει του τρόπου (εάν υφίσταται) που επικοινωνούν μεταξύ τους οι παράλληλες διεργασίες, και βάσει της φύσης των προβλημάτων που επιδιώκεται να λυθούν με παράλληλη επεξεργασία.[1]

Η παρούσα εργασία κινείται στον πρώτο άξονα, παρουσιάζοντας τα βασικά μοντέλα επικοινωνίας μεταξύ διεργασιών, και ειδικότερα μελετώντας τα κλασικά προγραμματιστικά εργαλεία για την επίλυση προβλημάτων συγχρονισμού που ενυπάρχουν στον προγραμματισμό κοινόχρηστης μνήμης, τις λεγόμενες αρχέγονες δομές συγχρονισμού (synchronization primitives).

2 Επικοινωνία μέσω κοινόχρηστης μνήμης (shared memory)

Στην παράλληλη ή ταυτόχρονη (concurrent) υπολογιστική, ένας πολύ φυσικός και προγραμματιστικά βολικός τρόπος για την επίτευξη επικοινωνίας μεταξύ διεργασιών, καθώς και για την αποφυγή σπατάλης χώρου από πολλαπλά όμοια αντίγραφα δεδομένων, είναι η χρήση ενός κοινού χώρου μνήμης από πολλές διεργασίες. Στο παρελθόν, ήταν διαδεδομένα τα υπολογιστικά συστήματα πολυεπεξεργαστών που διέθεταν αληθινά κοινόχρηστη μνήμη, με την οποία επικοινωνούσαν μέσω κοινού διαύλου. Σήμερα, η πραγματικότητα είναι διαφορετική.[2]

2.1 Συστήματα μη ομοιογενούς πρόσβασης μνήμης (NUMA)

Με την αύξηση του πλήθους και της ταχύτητας των κεντρικών μονάδων επεξεργασίας (CPU) των πολυεπεξεργαστών, η ανάγκη διαχείρισης του προβλήματος συμφόρησης (bottleneck) von Neumann επέβαλε αλλαγή στο μοντέλο επικοινωνίας των κεντρικών μονάδων επεξεργασίας με τη μνήμη. Οι σημερινοί πολυεπεξεργαστές παρακάμπτουν σε ένα βαθμό το παραπάνω πρόβλημα έχοντας σχεδιαστεί σε πρότυπο μη-ομοιογενούς πρόσβασης μνήμης (Non-Uniform Memory Access: NUMA).[4]

Στις αρχιτεκτονικές αυτού του προτύπου, όπως μαρτυράει το όνομα του, χαρακτηριστική είναι η ανομοιογένεια στην ταχύτητα πρόσβασης (latency) ενός επεξεργαστή στα διάφορα στοιχεία μνήμης, ανάλογα με την απόσταση του στοιχείου μνήμης από τον εν λόγω επεξεργαστή.[5] Οι καθιερωμένες διεπαφές προγραμματισμού εφαρμογών (APIs) κοινόχρηστης μνήμης (pthread, OpenMP) είτε δεν παρέχουν εύχρηστα μέσα για τη διαχείριση, μέσω κώδικα, της παραπάνω ιδιότητας των σύγχρονων πολυεπεξεργαστών[6], είτε οι λύσεις που παρέχουν, παρεχόμενες ως επεκτάσεις, καταργούν τη φορητότητα του κώδικα[7]. Η διαχείριση της μνήμης συνήθως αφήνεται, κατά συνέπεια, στο λειτουργικό σύστημα.

2.2 Υβριδικά παράλληλα συστήματα

Τα αναφερθέντα χαρακτηριστικά των εργαλείων προγραμματισμού που στηρίζονται στην αφαίρεση μιας εικονικά ενιαίας κοινόχρηστης μνήμης, συντελούν στο αποτέλεσμα η χρήση τους να περιορίζεται σε υπολογιστικά συστήματα μικρής κλίμακας. Διαδεδομένη, ωστόσο, είναι η χρήση τους σε συνδυασμό με εργαλεία προγραμματισμού κατανεμημένης μνήμης, για την υλοποίηση παράλληλων προγραμμάτων που εκτελούν επί μέρους εργασίες σε μικρά σύνολα επεξεργαστών, εντός υπολογιστικών συστημάτων μεγαλύτερης κλίμακας. Σε αυτού του είδους τις υβριδικές προσεγγίσεις, η επικοινωνία των επι μέρους έργων μεταξύ τους, αλλά και ευρύτερα με το σύστημα, υλοποιείται σε όρους προγραμματισμού κατανεμημένης μνήμης (message passing).[8]

3 Επικοινωνία μεσω μηνυμάτων (message passing)

Ο κλάδος του παραλληλισμού κατανεμημένης μνήμης είναι ευρύτατος και εσχάτως ραγδαία εξελισσόμενος. Καθώς δεν αποτελεί κεντρικό θέμα αυτής της εργασίας, θα καλυφθεί σύντομα και αμιγώς θεωρητικά.

Παραδοσιακά, με τον όρο "παραλληλισμός κατανεμημένης μνήμης" περιγράφεται το μοντέλο όπου μια μονάδα επεξεργασίας εκτελεί ένα έργο σε έναν τοπικό χώρο μνήμης, το αποτέλεσμα του οποίου στη συνέχεια επικοινωνείται μέσω μηνυμάτων προς μια απομακρυσμένη μονάδα επεξεργασίας.[2] Αυτό προγραμματιστικά μεταφράζεται σε συναρτήσεις αποστολής (send) και λήψης (receive) μηνυμάτων, που συντονίζονται σε άμεση επικοινωνία βάσει κοινού πρωτοκόλλου (π.χ. master - slave)[9] ή, σε πιο σύνθετα και/ή ασύγχρονα συστήματα, με τη χρήση ενδιάμεσου υλικού ή λογισμικού διαχείρισης μηνυμάτων (message handlers, buffers).[10]

3.1 Το πρότυπο MPI

Στο προγραμματιστικό μοντέλο επικοινωνίας μέσω μηνυμάτων, ένα πρόγραμμα που εκτελείται σε ένα ζεύγος κεντρικής μονάδας επεξεργασίας (CPU) και μνήμης ονομάζεται επεξεργαστής (processor). Στην προγραμματιστική κοινότητα έχει καθιερωθεί αυτός ο όρος και για τις διεργασίες (processes) που ορίζονται από το πρότυπο της Διεπαφής Μεταβίβασης Μηνυμάτων (Message Passing Interface: MPI).[11]

Ο όρος διεργασία υιοθετήθηκε σε αυτό το πρότυπο για να υποδηλώσει ότι, σε υλοποιήσεις στηριγμένες σε αυτό, η αντιστοίχιση μιας μονάδας επεξεργασίας ανά μονάδα μνήμης παύει να είναι απόλυτη. Αυτό είναι ένα θεμελιώδες χαρακτηριστικό της MPI, που διευκολύνει τη φορητότητα του κώδικα ανάμεσα σε διαφορετικές πλατφόρμες (λειτουργικά συστήματα) και διαφορετικής κλίμακας υπολογιστικά συστήματα.[12, 13, 14] Δημιουργεί, εν τούτοις, μια σύγχυση με τις διεργασίες

(processes) των λειτουργικών συστημάτων. Γι'αυτό έχει εμφανιστεί και ο όρος proclets, για πιο ειδική αναφορά στις διεργασίες του προτύπου MPI. [15]

Το πρότυπο MPI υπήρξε για περισσότερο από μια δεκαετία, και παραμένει, το κυρίαρχο πρότυπο μεταβίβασης μηνυμάτων των προγραμμάτων που χρησιμοποιούνται στη βιομηχανία της υπολογιστικής υψηλών αποδόσεων.[14] Σήμερα, ωστόσο, η εμφάνιση νέων εργαλείων κατανεμημένης υπολογιστικής υψηλών αποδόσεων, όπως τα σχετικά εύκολα στην εγκατάσταση (deployment) frameworks Apache Hadoop και Spark, και γλωσσών κατασκευασμένων εξ αρχής για παράλληλη επεξεργασία κατανεμημένης μνήμης, όπως οι Charm++, Chapel, Julia, Erlang / Elixir, έχουν αρχίσει να περιορίζουν το εύρος εφαρμογών όπου το πρότυπο MPI, στις διάφορες υλοποιήσεις του, διατηρεί την πρωτοκαθεδρία.[16, 17, 18]

3.2 Υπολογιστική γενικού σκοπού σε GPU (GPGPU)

Οι μονάδες επεξεργασίας γραφικών (GPU) κατασκευάζονται με γνώμονα την επίλυση προβλημάτων γραφικής απεκόνισης και επεξεργασίας εικόνων. Η φύση αυτών των προβλημάτων έχει προσανατολίσει το σχεδιασμό των GPUs στην επίτευξη υψηλών αποδόσεων συνολικά σε υπολογισμούς που επωφελούνται από έντονο παραλληλισμό δεδομένων.[19]

Την ίδια στιγμή, οι GPUs έχουν πολύ περιορισμένες υπολογιστικές λειτουργίες και δυνατότητες προγραμματισμού, και η χρήση τους ενδείκνυται αποκλειστικά για επίλυση προβλημάτων που επωφελούνται από την επεξεργασία ροών (stream processing).[20]

Η τεχνολογία διαύλων υπολογιστικής γενικού σκοπού σε GPU (GPGPU pipelines) που αναπτύχθηκε ως λογισμικό με σκοπό τη βελτίωση των γραφικών σκίασης (shaders) στις αρχές της προηγούμενης δεκαετίας[21], σύντομα αξιοποιήθηκε και για πολύ διαφορετικούς σκοπούς, σε υπολογισμούς που επωφελούνται από υψηλές επιδόσεις στην παραλληλία δεδομένων (επιστημονική υπολογιστική, βιοπληροφορική, εξόρυξη κρυπτονομισμάτων).[22]
Τα προγραμματιστικά μοντέλα που αναπτύχθηκαν για τον προγραμματισμό συστημάτων GPGPU στηρίχθηκαν στην θεώρηση τους ως συστήματα κατανεμημένης μνήμης. Αυτή η αφαίρεση είχε μεγαλύτερη ανταπόκριση στην πραγματικότητα του τότε υλικού, όπου η επικοινωνία μεταξύ επεξεργαστή και κάρτας γραφικών γινόταν μέσω διαύλου PCI, που μεταφραζόταν σε μεγάλη καθυστέρηση (latency) στην επικοινωνία μεταξύ των δύο επεξεργαστικών μονάδων (CPU και GPU) με τις επιμέρους μονάδες μνήμης τους. Σήμερα, υπάρχουν προτάσεις για ανάπτυξη προτύπων προγραμματισμού GPGPU που να προσομοιάζουν περισσότερο το μοντέλο παραλληλισμού κοινόχρηστης μνήμης.[2]

Part II
Προβλήματα συγχρονισμού σε συστήματα κοινόχρηστης μνήμης

4 Αρχέγονες Δομές Συγχρονισμού

Κοινόχρηστη μνήμη = Συνθήκες ανταγωνισμού, προβλήματα συντονισμού τύπου bounded buffer -> Αμοιβαίος αποκλεισμός

4.1 Spinlock

4.2 Barrier

4.3 Semaphore, Mutex

5 Dining philosophers

Παρουσίαση του προβλήματος (εισαγωγή όρων deadlock, resource starvation), λύση με μέθοδο arbitrator για επίδειξη barriers. Επιλογή δυαδικού σηματοφορέα αντί mutex, για συνοχή.

6 Readers-writers

Παρουσίαση και λύση του προβλήματος, με προτίμηση στην απλούστερη εκδοχή όπου δίνεται αφορμή για αμοιβαίο αποκλεισμό με χρήση spinlock. Εάν αυτό αποδειχτεί δύσκολο, λύση της απλούστερης εκδοχής (producer-consumer equivalent) με χρήση monitor. Στην τελευταία περίπτωση, θα προηγηθεί κάλυψη της εν λόγω δομής στη θεωρία.

Part III
Μεταφορά θερμότητας σε δισδιάστατο χώρο: ένα πρόβλημα παράλληλου προγραμματισμού

Παρουσίαση του προβλήματος παράλληλου υπολογισμού της μεταφοράς θερμότητας σε δισδιάστατο χώρο. Μελέτη της παράλληλης λύσης με προγραμματισμό κατανεμημένης μνήμης (MPI), λύση με προγραμματισμό κοινόχρηστης μνήμης (pthreads).

Τι προκύπτει από τη σύγκριση των δυο λύσεων ως προς την απόδοση και την απλότητα υλοποίησης; Ποια γενικά κριτήρια για την επιλογή ανάμεσα στους δύο τύπους παράλληλου προγραμματισμού μπορούν να εξαχθούν από τη μελέτη των λύσεων του συγκεκριμένου προβλήματος;

Η εσωτερική δομή της ενότητας αυτής θα οριστεί αργότερα.

References

[1] https://en.wikipedia.org/wiki/Parallel_programming_model

[2] https://stackoverflow.com/questions/36642382/main-difference-between-shared-memory-and-distributed-memory#answer-36659895

[3] www.shodor.org/media/content/petascale/materials/UPModules/beginnersGuideHPC/moduleDocun

[4] https://en.wikipedia.org/wiki/Von_Neumann_architecture#Mitigations

[5] https://en.wikipedia.org/wiki/Non-uniform_memory_access

[6] https://stackoverflow.com/questions/11959906/openmp-and-numa-relation#answer-11975593

[7] http://man7.org/linux/man-pages/man3/pthread_setaffinity_np.3.html

[8] https://software.intel.com/en-us/articles/hybrid-parallelism-parallel-distributed-memory-and-shared-memory-computing

[9] https://en.wikipedia.org/wiki/Message_Passing_Interface#Point-to-point_basics

[10] https://en.wikipedia.org/wiki/Message_passing#Synchronous_versus_asynchronous_message_pass

[11] https://en.wikipedia.org/wiki/Message_Passing_Interface#Functionality

[12] http://www.netlib.org/utk/papers/mpi-book/node3.html

[13] www.netlib.org/utk/people/JackDongarra/PAPERS/cluster.pdf

[14] https://en.wikipedia.org/wiki/Message_Passing_Interface#Overview

[15] https://dl.acm.org/citation.cfm?id=1851507

[16] https://www.dursi.ca/post/hpc-is-dying-and-mpi-is-killing-it.html

[17] https://en.wikipedia.org/wiki/Apache_Hadoop#Prominent_use_cases

[18] https://en.wikipedia.org/wiki/Julia_(programming_language)#Notable_uses

[19] https://en.wikipedia.org/wiki/Graphics_processing_unit#Stream_processing_and_general_purpos

[20] https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units#Stream

[21] https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units#History

[22] https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units#Applica