Εθνικο Μετσοβιο Πολυτεχνειο
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ
9ο εξάμηνο ΗΜΜΥ, ακαδημαϊκό έτος 2005-06

1 Γενικά

Σκοπός της εργαστηριακής άσκησης του μαθήματος είναι η σχεδίαση και υλοποίηση παράλληλου αλγορίθμου, που να επιλύει επαναληπτικά διακριτοποιημένη εκδοχή της ακόλουθης ασταθούς εξίσωσης διάχυσης θερμότητας (diffusion equation) με δεδομένες αρχικές και συνοριακές συνθήκες:

$$
\begin{aligned}
\frac{\partial U}{\partial t} &= \nabla^2 U \\
U(x,y,0) &= f(x,y) \\
U(x,y,t) &= g(x,y,t) \text{ στο } \partial\Omega
\end{aligned}
$$

όπου U η συνάρτηση θερμότητας σε διδιάστατη ορθογώνια επιφάνεια $X \times Y$, $f(x,y)$ η αρχική τιμή της συνάρτησης θερμότητας για $t=0$ και $g(x,y,t)$ η τιμή της θερμότητας στο σύνορο $\partial\Omega$ της επιφάνειας. Η διακριτοποίηση των χωρικών και χρονικών παραγώγων του παραπάνω αλγορίθμου μπορεί να οδηγήσει στον ακόλουθο αλγόριθμο επαναληπτικής φύσεως για τον υπολογισμό της θερμότητας:

for $x \leftarrow 2$ ***to*** $\frac{X}{\Delta x}$ **do**
  for $y \leftarrow 2$ ***to*** $\frac{Y}{\Delta y}$ **do**
    for $t \leftarrow 1$ ***to*** $\frac{T}{\Delta t}$ **do**
      $U[x,y,t] = U[x,y,t-1] + \frac{\Delta t}{\Delta x^2}\big(2U[x,y,t-1] - 2U[x-1,y,t-1] -$
      $2U[x,y-1,t-1] + U[x-2,y,t-1] + U[x,y-2,t-1]\big);$

Στον παραπάνω αλγόριθμο, $X \times Y$ είναι οι διαστάσεις της υπό εξέταση επιφάνειας, ενώ η παράμετρος T αντιστοιχεί στη χρονική διάρκεια εξέλιξης του φαινομένου. Περισσότερα τόσο για την εξίσωση διάχυσης, όσο και για τη διακριτοποίηση μερικών διαφορικών εξισώσεων γενικότερα, μπορούν να αναζητηθούν στο κεφάλαιο 5 του [7].

2 Ζητούμενα

Ζητούμενο της εργαστηριακής άσκησης είναι η υλοποίηση παράλληλου προγράμματος σε MPI, καθώς και προαιρετικά υβριδικού προγράμματος με χρήση MPI+OpenMP, που να υλοποιούν τον αλγόριθμο της ενότητας 1. Αφετηριακό σημείο για τη σχεδίαση ενός παράλληλου προγράμματος αποτελεί η υιοθέτηση στρατηγικής *κατανομής των υπολογισμών* (computation distribution) στις επιμέρους διεργασίες. Οι δυνατότητες εναλλακτικών σχημάτων κατανομής υπολογισμών προκύπτουν από τη μελέτη της φύσης του αλγορίθμου, ενώ η επιλογή ενός συγκεκριμένου σχήματος θα καθορίσει τελικά και τις αντίστοιχες απαιτήσεις *κατανομής των δεδομένων* (data distribution) μεταξύ των διεργασιών.

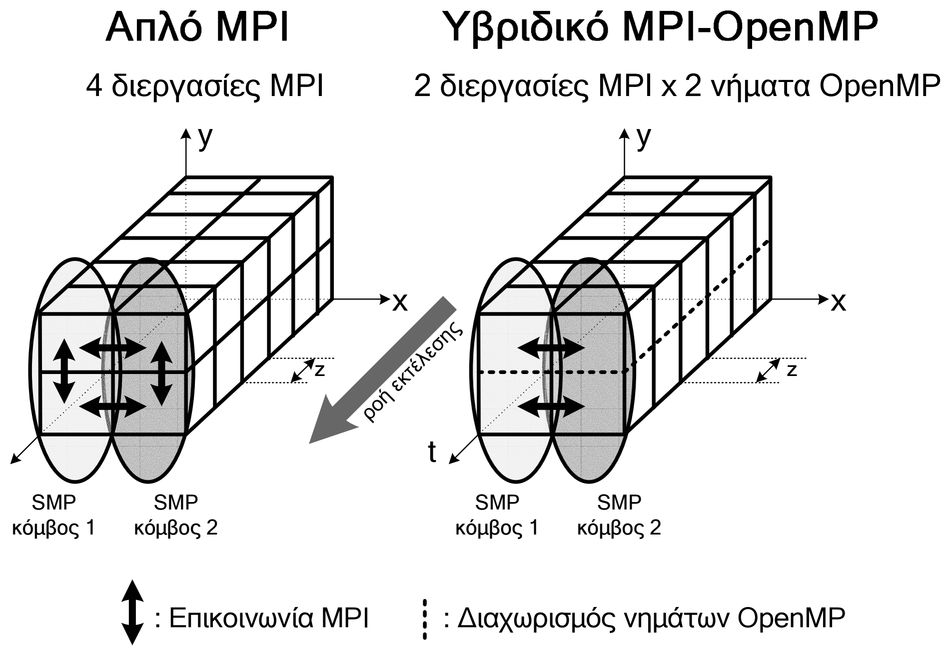
Στο σχήμα 1 προτείνεται μια δυνατή κατανομή των βασικών δεδομένων του προβλήματος. Ο τρισδιάστατος πίνακας θερμότητας U κατανέμεται ως προς τις χωρικές διαστάσεις x, y στους διαθέσιμους φορείς εκτέλεσης (διεργασίες/νήματα), κάθε ένας από τους οποίους αναλαμβάνει τη σειριακή εκτέλεση υπολογισμών, ακολουθιακά κατά την τρίτη χρονική διάσταση (t). Ο υπολογισμός των επιμέρους τιμών του πίνακα U εναλλάσσεται με φάσεις επικοινωνίας, που απαιτούνται για την ανταλλαγή συνοριακών δεδομένων μεταξύ των διεργασιών. Μάλιστα, προκειμένου να επιτευχθεί αφενός ικανοποιητικός βαθμός παραλληλισμού, και αφετέρου να ελαχιστοποιηθεί η επιβάρυνση της επικοινωνίας MPI, θα πρέπει η ανταλλαγή συνοριακών δεδομένων να γίνεται περιοδικά ανά *υπερκόμβο* (*tile*) υπολογισμού, σε αντιδιαστολή π.χ. με την επικοινωνία ανά στοιχείο ή επιφάνεια υπολογισμού. Το βέλτιστο ύψος z του υπερκόμβου (βλ. σχήμα 1), που ελαχιστοποιεί το συνολικό χρόνο εκτέλεσης του παράλληλου προγράμματος, αποτελεί συμβιβασμό μεταξύ της επιδίωξης υψηλού βαθμού παραλληλίας (μικρό ύψος υπερκόμβου, άρα μεγάλος αριθμός παράλληλων φάσεων εκτέλεσης) και της προσπάθειας ελαχιστοποίησης της επιβάρυνσης επικοινωνίας (μεγάλο ύψος υπερκόμβου, ή ισοδύναμα μικρός αριθμός ανταλλασσόμενων μηνυμάτων). Περισσότερα για την παραλληλοποίηση αλγορίθμων φωλιασμένων βρόχων, καθώς και το μετασχηματισμό υπερκόμβου, μπορούν να αναζητηθούν στα [6], [8], [4], [9], [5], [2], [3], [1].

Με βάση τα παραπάνω, ζητούνται από κάθε ομάδα τα εξής:

- Η **υλοποίηση παράλληλου προγράμματος** σε MPI ή προαιρετικά σε υβριδικό MPI+OpenMP, που να εφαρμόζει τον διακριτοποιημένο αλγόριθμο. Η υλοποίηση του υβριδικού προγράμματος είναι προαιρετική, και πριμοδοτείται με μία επιπλέον μονάδα. Για απλότητα, υποθέτουμε ότι μας ενδιαφέρουν οι τιμές της U μονάχα κατά την τελική χρονική στιγμή $t = T - 1$. Επίσης, για απλότητα θα υποθέσουμε ότι $\Delta x = \Delta y = \Delta t = 1$. Τέλος, θεωρείστε ότι

$$f(x,y) = g(x,y,t) = \begin{cases} 1 \quad , & x = X - 1 \\ 1 \quad , & y = Y - 1 \\ 0 \quad , & \text{οπουδήποτε αλλού} \end{cases}$$

  Σε κάθε περίπτωση όμως, η δήλωση και χρήση όλων των παραπάνω μεγεθών ($\Delta x, \Delta y, \Delta t, f, g$) στο πρόγραμμα να γίνεται με δομημένο τρόπο, για να είναι εύκολο να μεταβληθεί οποιοδήποτε από αυτά. Επίσης, το πρόγραμμα θα πρέπει να είναι παραμετρικό ως προς τον αριθμό των διεργασιών Pr, το πλήθος Th των



Σχήμα 1: Παράδειγμα παραλληλοποίησης του αλγορίθμου σε 4 διεργασίες και 2 διπλούς SMP κόμβους, με χρήση τόσο απλού MPI όσο και υβριδικού MPI-OpenMP

νημάτων (για υβριδικό), τις διαστάσεις X, Y, T του προβλήματος, καθώς και το ύψος z του υπερκόμβου, που καθορίζει τη συχνότητα επικοινωνίας. Κάθε διεργασία πρέπει να δεσμεύει μόνο το χώρο μνήμης που χρειάζεται για την επεξεργασία του δικού της, τοπικού κομματιού του πίνακα U, και σε καμία περίπτωση να μη δεσμεύει χώρο για όλο τον U. Μάλιστα, καθώς ενδιαφέρουν μόνο οι τελικές τιμές του U την $t = T - 1$, κάθε διεργασία χρειάζεται χώρο για την αποθήκευση μόνο της τρέχουσας και την προηγούμενης τιμής του U σε συγκεκριμένο σημείο (x, y), συνεπώς το μήκος της χρονικής διάστασης t του U θα πρέπει να είναι 2.

Η έξοδος του προγράμματος (τιμές $U(x, y, t)$ για $t = T - 1$) θα πρέπει να μπορεί να γράφεται σε **ένα** αρχείο κειμένου `output<Pr>_<X>x<Y>x<T>_<z>.txt` ή στην υβριδική περίπτωση `output<Pr>_<Th>_<X>x<Y>x<T>_<z>.txt`, που θα έχει την ακόλουθη μορφή:

$$\begin{array}{rcl} U[0][0][T-1] & = & < value_1 > \\ U[0][1][T-1] & = & < value_2 > \\ & \cdots & \\ U[X-1][Y-1][T-1] & = & < value_{XY} > \end{array}$$

Οι τιμές του U να είναι πραγματικοί αριθμοί διπλής ακριβείας τύπου `double`. Για ευκολία, η αρχικοποίηση του πίνακα U μέσω της f, καθώς και η χρήση των συνοριακών τιμών μέσω της g, μπορεί να γίνεται μέσα στο πρόγραμμα από κάθε διεργασία ξεχωριστά βάσει των f, g, δεν είναι υποχρεωτικό π.χ. να την κάνει η root διεργασία και να αποστείλει τις τιμές στις υπόλοιπες διεργασίες.

- Η **πραγματοποίηση μετρήσεων** για χώρους επαναλήψεων $X \times Y \times T = \{1k \times 32 \times 2k, 1k \times 64 \times 2k, 1k \times 128 \times 2k, 1k \times 256 \times 2k, 1k \times 512 \times 2k\}$, πλήθος διεργασιών $Pr = \{2, 4, 8, 16\}$ και ύψος υπερκόμβου $z = 50$, και η σύγκριση των αντίστοιχων χρόνων εκτέλεσης με εκείνους του *σειριακού* προγράμματος (προσοχή! όχι του παραλλήλου για μία διεργασία). Στην περίπτωση υβριδικής υλοποίησης να θεωρήσετε επιπλέον $Pr = \{2, 4, 8\}$ και $Th = 2$. Συγκεκριμένα, για κάθε χώρο επαναλήψεων, να παρασταθεί η *επιτάχυνση* που επιτυγχάνεται ως προς το σειριακό αλγόριθμο σαν συνάρτηση του πλήθους Pr των διεργασιών (ή και του γινομένου $Pr \times Th$, στην υβριδική περίπτωση). Για τη μέτρηση των χρόνων προτείνεται να χρησιμοποιηθεί η συνάρτηση βιβλιοθήκης `gettimeofday` του `sys/time.h`. Πιο συγκεκριμένα, να ακολουθηθεί η εξής διαδικασία:

  ```
  /* ορισμός μεταβλητών, δεσμεύσεις μνήμης,
     αρχικοποιήσεις πινάκων, αρχικοποιήσεις MPI */
  MPI_Barrier(MPI_COMM_WORLD);
  gettimeofday(start,(struct timezone*)NULL);
  /* παράλληλος αλγόριθμος εξίσωσης διάχυσης */
  ...
  MPI_Barrier(MPI_COMM_WORLD);
  gettimeofday(finish,(struct timezone*)NULL);
  duration=f(finish-start);
  /* εκτύπωση αποτελεσμάτων σε αρχείο */
  ```

  Παρατηρείστε ότι κατά την μέτρηση χρόνων ενδιαφέρει **μόνο** το υπολογιστικό κομμάτι του αλγορίθμου, και όχι η φάση αρχικοποίησης ή εκτύπωσης των αποτελεσμάτων. Ο συγχρονισμός των διεργασιών μέσω της `MPI_Barrier` πριν κάθε μέτρηση χρόνου θα οδηγήσει στον υπολογισμό του χρόνου από την (συγχρονισμένη) έναρξη όλων των διεργασιών μέχρι την περάτωση της τελευταίας χρονικά διεργασίας, που αποτελεί άλλωστε και τον πραγματικό χρόνο εκτέλεσης του παράλληλου προγράμματος. Επίσης, για το χώρο επαναλήψεων $1kx128x2k$ και $Pr = 16$ διεργασίες να πραγματοποιηθούν μετρήσεις για μεταβλητό ύψος υπερκόμβου, στο εύρος $z = 8 \ldots 256$ με βήμα 8, και να παρασταθεί γραφική η επιτάχυνση που επιτυγχάνεται σε συνάρτηση με το z. Επιπλέον, σε περίπτωση υβριδικής υλοποίησης να επαναληφθεί η διαδικασία αυτή για $Pr = 8$ διεργασίες και $Th = 2$ νήματα, και να παρασταθεί η επιτάχυνση σε *κοινό* διάγραμμα με το απλό MPI.

- **Σύντομη αναφορά** με δικαιολόγηση των κυριότερων σχεδιαστικών επιλογών του προγράμματος, επεξήγηση των βασικών δομών και την γενικής λειτουργίας αυτού, παρουσίαση μετρήσεων με τα διαγράμματα επιτάχυνσης, καθώς και σχολιασμό των πειραματικών αποτελεσμάτων.

3 Διευκρινίσεις

Η ανάπτυξη της άσκησης θα γίνει στις συστοιχίες υπολογιστών του εργαστηρίου, τα 16 kids (kid1-kid16) για απλό MPI και τα 8 twins (twin1-twin4, twin6, twin7, twin9, twin12) σε περίπτωση υβριδικού προγράμματος. Συγκεκριμένα, θα ήταν καλό όσοι υλοποιήσουν μόνο το απλό MPI να λάβουν τις τελικές μετρήσεις χρόνου στα kids, ενώ όσοι υλοποιήσουν επιπλέον και το υβριδικό να προτιμήσουν για το σύνολο των τελικών μετρήσεων τους τα twins. Για να συνδεθούμε στα μηχανήματα αυτά, ακολουθούμε τις οδηγίες που βρίσκονται στη σελίδα του μαθήματος (`http://www.cslab.ece.ntua.gr/courses/pps`).

Για ομοιομορφία των μετρήσεων, συνίσταται να χρησιμοποιηθεί ο Intel C++ compiler (icc) κατά τη μεταγλώττιση του σειριακού, καθώς και η εγκατάσταση MPI που βασίζεται στον icc για τη μεταγλώττιση των παραλλήλων προγραμμάτων. Το παραπάνω μπορεί να επιτευχθεί με χρήση του `/usr/local/mpich-intel/bin/mpicc` για μεταγλώττιση σε όλες τις περιπτώσεις (χρειάζεται `-openmp` για το υβριδικό), ενώ το script `/usr/local/mpich-intel/bin/mpirun` μπορεί να χρησιμοποιηθεί για την εκτέλεση των παράλληλων προγραμμάτων. Τα δύο παραπάνω scripts, για δική σας διευκόλυνση, έχουν γίνει aliased σε `mpicc`, `mpirun`, αντίστοιχα. Επίσης, για ταχύτερους χρόνους εκτέλεσης, προτείνεται η χρήση επιπέδου βελτιστοποίησης `-O3` κατά τη μεταγλώττιση.

Έτσι, π.χ. για την μεταγλώττιση και εκτέλεση ενός προγράμματος MPI mde.c, συνίσταται η εξής ακολουθία εντολών:

```
mpicc -o mde mde.c -O3
mpirun -np 16 mde
```

Σε περίπτωση υβριδικού προγράμματος hde.c, η πρώτη εντολή τροποποιείται ως εξής:

```
mpicc -o hde hde.c -O3 -openmp
```

Κατά την υβριδική προσέγγιση, επιθυμούμε σε κάθε SMP κόμβο της συστοιχίας των twins να εκκινείται μία μόνο διεργασία MPI, η οποία με τη σειρά της θα αξιοποιεί τους δύο επεξεργαστές του κόμβου με ισάριθμα νήματα OpenMP. Επειδή η εγκατάσταση του MPI θα εκκινεί εξ ορισμού δύο διεργασίες σε κάθε SMP κόμβο, απαιτείται η χρήση κατάλληλου machinefile. Έτσι, η εκτέλεση του παράλληλου προγράμματος στην περίπτωση αυτή θα πρέπει να γίνεται με

```
mpirun -np 8 -machinefile machines hde
```

όπου το αρχείο machines θα περιέχει τα εξής:

```
twin1
twin2
twin3
twin4
twin6
twin7
twin9
twin12
```

Περισσότερες πληροφορίες σχετικά με το MPICH και τις SMP συστοιχίες υπάρχουν στο `http://www-unix.mcs.anl.gov/mpi/mpich/docs/mpichman-chp4/node14.htm#Node19`.

Τέλος, είναι ιδιαίτερα σημαντικό να φροντίζει κάθε ομάδα για τον καθαρισμό zombie διεργασιών και τυχόν δεσμευμένων IPC segments, π.χ. κατά τον απότομο τερματισμό ενός MPI προγράμματος. Για το σκοπό αυτό χρησιμοποιείστε τα scripts `/kid/bin/mexec` και `/twin/bin/mexec`, ανάλογα με τη συστοιχία που εργάζεστε. Έτσι, π.χ. στα kids, μπορούμε με

```
/kid/bin/mexec killall mde
```

να σκοτώσουμε όλες τις zombie διεργασίες του προγράμματος mde. Επίσης, με

```
/twin/bin/mexec ipcs
```

μπορούμε στα twins να διερευνήσουμε κατά πόσο διατηρούμε δεσμευμένα IPC shared memory segments, και αντίστοιχα να τα απελευθερώσουμε με την εντολή `cleanipcs` ως εξής:

```
/twin/bin/mexec cleanipcs
```

Η τήρηση των παραπάνω θα επιτρέψει την ομαλή λειτουργικότητα του περιβάλλοντος των συστοιχιών, τόσο για τα δικά μας προγράμματα, όσο και για των υπολοίπων.

Καλή επιτυχία!

Βιβλιογραφία

[1] N. Drosinos and N. Koziris. Performance Comparison of Pure MPI vs Hybrid MPI-OpenMP Parallelization Models on SMP Clusters. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium 2004 (IPDPS 2004)*, page 15, Santa Fe, New Mexico, Apr 2004.

[2] G. Goumas, M. Athanasaki, and N. Koziris. An Efficient Code Generation Technique for Tiled Iteration Spaces. *IEEE Trans. on Parallel and Distributed Systems*, 14(10):1021–1034, Oct 2003.

[3] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS'01)*, San Francisco, USA, Apr 2001.

[4] E. Hodzic and W. Shang. On Supernode Transformation with Minimized Total Running Time. *IEEE Trans. on Parallel and Distributed Systems*, 9(5):417–428, May 1998.

[5] K. Högstedt, L. Carter, and J. Ferrante. On the Parallel Execution Time of Tiled Loops. *IEEE Trans. on Parallel and Distributed Systems*, 14(3):307–321, Mar 2003.

[6] F. Irigoin and R. Triolet. Supernode Partitioning. In *Proc. of the 15th Ann. ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages (POPL'85)*, pages 319–329, San Diego, California, USA, Jan 1988.

[7] G. Em. Karniadakis and R. M. Kirby. *Parallel Scientific Computing in C++ and MPI : A Seamless Approach to Parallel Algorithms and their Implementation*. Cambridge University Press, 2002.

[8] M. Wolf and M. Lam. A Loop Transformation Theory and an Algorithm to Maximize Parallelism. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):452–471, Oct 1991.

[9] J. Xue. Communication-Minimal Tiling of Uniform Dependence Loops. *Journal of Parallel and Distributed Computing*, 42(1):42–59, 1997.