Στρατιγικές Διαμέρισης (Partitioning) και Διαίρει και Βασίλευε (Divide-and-Conquer)

• Στη τεχνική της διαμέρισης, το αρχικό πρόβλημα διαιρείται σε μικρότερα προβλήματα. Συνήθως γίνεται διαμέριση των δεδομένων εισόδου και κάθε διεργασία αναλαμβάνει ένα τμήμα δεδομένων.
• Όπως είναι γνωστό, στη τεχνική Διαίρει και Βασίλευε, το πρόβλημα διαιρείται σε υποπροβλήματα της ίδιας μορφής όπως το αρχικό. Κάθε υποπρόβλημα μπορεί να διαιρεθεί με αναδρομικό τρόπο σε ακόμα μικρότερα προβλήματα μέχρι να προκύψουν προβλήματα που η λύση τους προκύπτει κατευθείαν
• Για παράδειγμα, στην άθροιση των στοιχείων ενός πίνακα, μπορούμε να χωρίσουμε τον πίνακα εισόδου σε δύο υποπίνακες και στη συνέχεια να λύσουμε αναδρομικά το πρόβλημα της άθροισης στους δύο υποπίνακες. Το τελικό άθροισμα προκύπτει από το άθροισμα των μερικών αθροισμάτων των δύο υποπινάκων.
• Η αναδρομική διαδικασία της διαίρει και βασίλευε στρατηγικής μπορεί εύκολα να υλοποιηθεί παράλληλα διότι ξεχωριστές διεργασίες μπορούν να αναλάβουν τα προβλήματα που προκύπτουν από την αναδρομική διαίρεση.

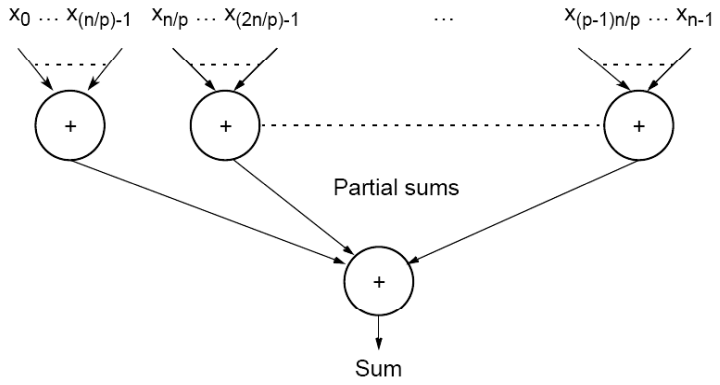
Παραδείγματα με εφαρμογή στρατηγικών Διαμέρισης και Διαίρει και Βασίλευε

• Λειτουργίες σε ακολουθίες αριθμών όπως η άθροιση
•Μερικοί αλγόριθμοι ταξινόμησης συγκεκριμένα αλγόριθμοι που μπορούν να σχεδιαστούν με αναδρομή

• Εύρεση ολοκληρώματος συναρτήσεων

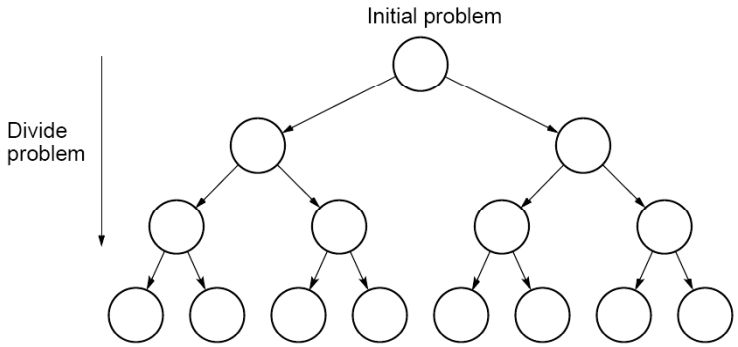
• Το πρόβλημα N-body

Παράδειγμα διαμέρισης – Άθροισμα των στοιχείων μίας λίστας

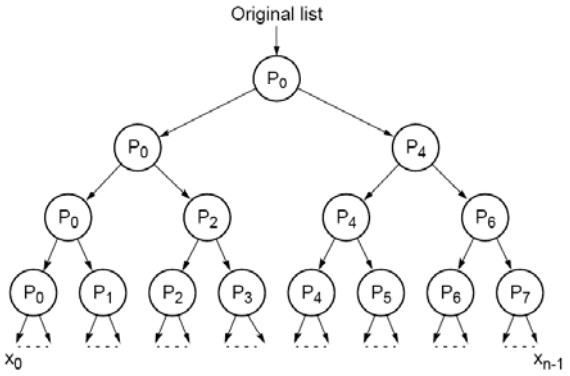


- Τα στοιχεία της λίστας χωρίζονται σε p υπολίστες όπου p είναι το πλήθος των διεργασιών που θα εκτελέσουν το παράλληλο πρόγραμμα.
- Η διεργασία p_i (i=0...p-1) αναλαμβάνει να αθροίσει τα στοιχεία $x_{i*n/p}$ ...$x_{(i+1)*n/p-1}$ και στη συνέχεια στέλνει το αποτέλεσμα της τοπικής άθροισης s_i στη master διεργασία η οποία αθροίζει όλα τα μερικά αθροίσματα s_i και υπολογίζει το τελικό αποτέλεσμα

Τεχνική Διαίρει και Βασίλευε

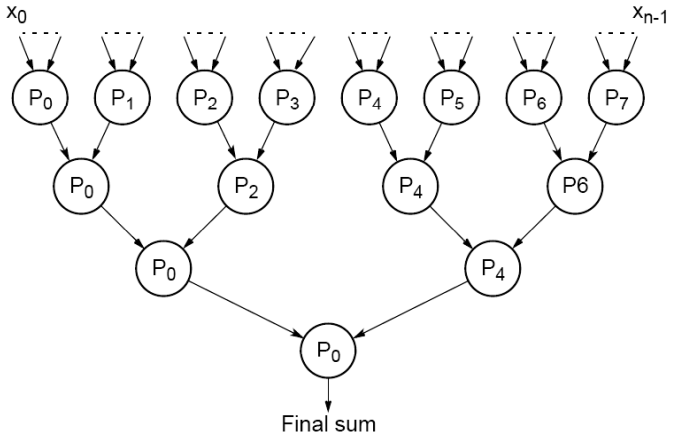


Η τεχνική διαίρει και βασίλευε μπορεί να αναπαρασταθεί με τη βοήθεια ενός δένδρου.
Σε κάθε επίπεδο, κάθε πρόβλημα διαιρείται σε δύο υποπροβλήματα μέχρι να φτάσουμε στο επίπεδο των φύλλων όπου η λύση των υποπροβλημάτων είναι προφανής. Π.χ. στο πρόβλημα της άθροισης των στοιχείων μίας λίστας, το επίπεδο των φύλλων περιέχει μόνο στοιχειώδη προβλήματα, συγκεκριμένα άθροιση δύο στοιχείων.

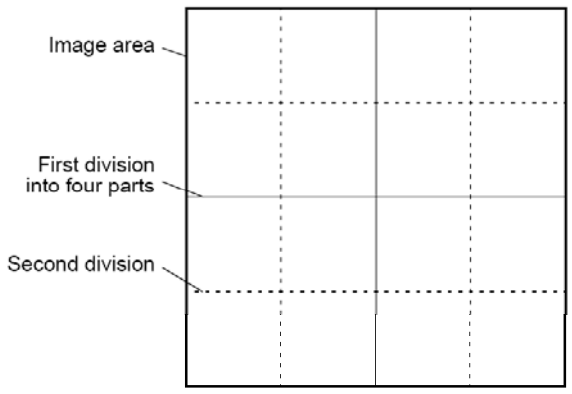


Η διαίρει και βασίλευε τεχνική εύκολα να υλοποιηθεί παράλληλα. Στο πρόβλημα της άθροισης n στοιχείων x_0 ...x_{n-1} αρχικά η διαδικασία P_0 διαθέτει όλη τη λίστα των στοιχείων. Στη συνέχεια διαιρεί την λίστα σε δύο υπολίστες και δίνει τη δεύτερη υπολίστα στη διεργασία P_4 . Στη συνέχεια, οι διεργασίες P_0 και P_4 διαιρούν τις υπολίστες τους σε δύο μικρότερες υπολίστες και δίνουν τη δεύτερη υπολίστα στις διεργασίες P_2 και P_6. Αυτή η διαδικασία επαναλαμβάνεται μέχρι κάθε διεργασία να μείνει με n/8 στοιχεία της αρχικής λίστας.

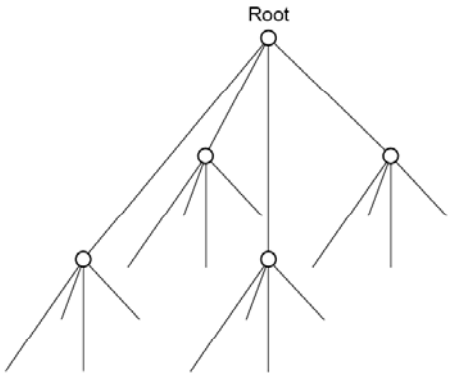
Μερική άθροιση



Στη συνέχεια κάθε διεργασία βρίσκει το άθροισμα των n/8 στοιχείων που διαθέτει. Στη συνέχεια οι μισές διεργασίες στέλνουν τα μερικά τους αθροίσματα στις άλλες μισές. Οι διεργασίες αυτές αθροίζουν το δικό τους μερικό άθροισμα με αυτό που λαμβάνουν και στη συνέχεια η παραπάνω διαδικασία επαναλαμβάνεται στις τέσσερις εναπομείνασες μέχρι το τελικό άθροισμα να προκύψει στη διεργασία P_0



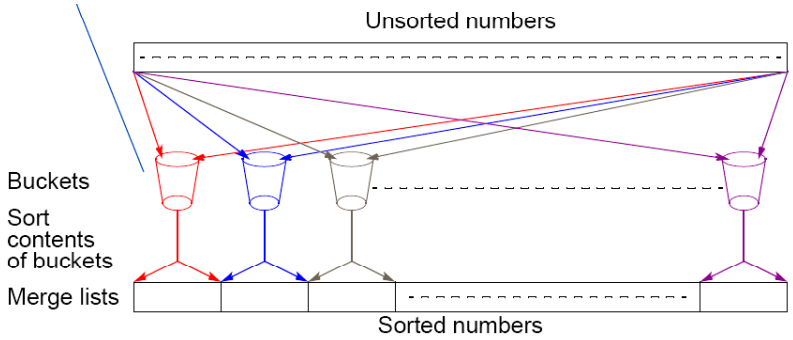
Διαδοχική διαίρεση της εικόνας σε 4 περιοχές



Quadtree

Στη τεχνική διαίρει και βασίλευε, το αρχικό πρόβλημα μπορεί να διαιρείται σε περισσότερα από δύο υποπροβλήματα σε κάθε βήμα. Μία συχνή περίπτωση προκύπτει στην εφαρμογή της διαίρει και βασίλευε τεχνικής στην επεξεργασία εικόνας. Σε αυτές τις περιπτώσεις εφαρμογών, σε κάθε βήμα η εικόνα διαιρείται σε 4 ίσου μεγέθους περιοχές. Το αντίστοιχο δέντρο που περιγράφει τον υπολογισμό είναι το τετραδικό δέντρο (Quadtree)

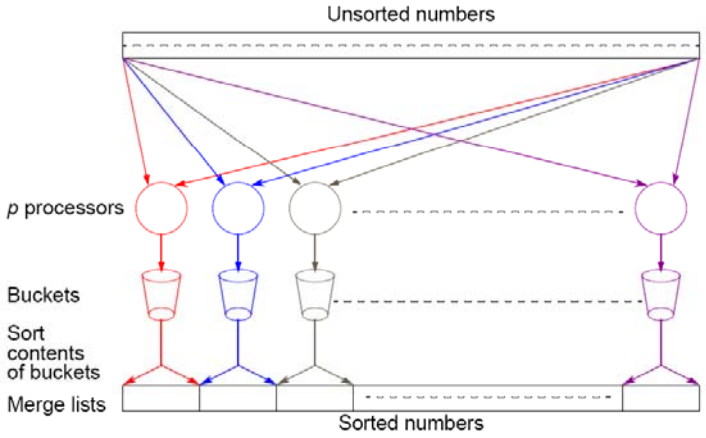
Bucket sort



• Ο bucket sort είναι ένας αλγόριθμος ταξινόμησης όπου η βασική υπόθεση είναι ότι τα n στοιχεία του πίνακα που θα ταξινομηθεί είναι στο διάστημα [0..α-1].
• Στη συνέχεια ορίζονται m “κάδοι” (“buckets”) και κάθε κάδος αποθηκεύει στοιχεία σε μία συγκεκριμένη περιοχή. Πιο συγκεκριμένα ο κάδος i (i=0,...,m-1) περιέχει στοιχεία στη περιοχή [i*α/m...(i+1)*α/m-1].
• Τα στοιχεία σε κάθε κάδο ταξινομούνται με τη χρήση ενός ακολουθιακού αλγόριθμου ταξινόμησης.
• Στη συνέχεια, οι ταξινομημένες λίστες των κάδων συγχωνεύονται σε μία ταξινομημένη λίστα. Η συγχώνευση είναι εύκολη τοποθετώντας στη τελική λίστα πρώτα τα περιεχόμενα του κάδου 0 μετά του κάδου 1 κοκ.
• Η συνολική πολυπλοκότητα του ακολουθιακού αλγορίθμου είναι O(nlog(n/m)) εφόσον τα στοιχεία του πίνακα κατανέμονται ομοιόμορφα στο διάστημα [0..α-1].

Παράλληλη υλοποίηση του bucket sort
Απλή προσέγγιση

- Κάθε διεργασία αναλαμβάνει την ταξινόμηση ενός κάδου (p=m).
- Κάθε διεργασία πρέπει να ελέγχει όλα τα στοιχεία του πίνακα εισόδου προκειμένου να απομονώσει τα στοιχεία που προορίζονται για το κάδο του.
- Αυτό έχει ως αποτέλεσμα να γίνονται πολλοί άχρηστοι υπολογισμοί.
- Εναλλακτικά κάθε διεργασία μπορεί να αφαιρεί από τον αρχικό πίνακα τα στοιχεία που θα τοποθετηθούν στο κάδο της. Έτσι οι υπόλοιπες διεργασίες δεν θα εξετάσουν αυτά τα στοιχεία

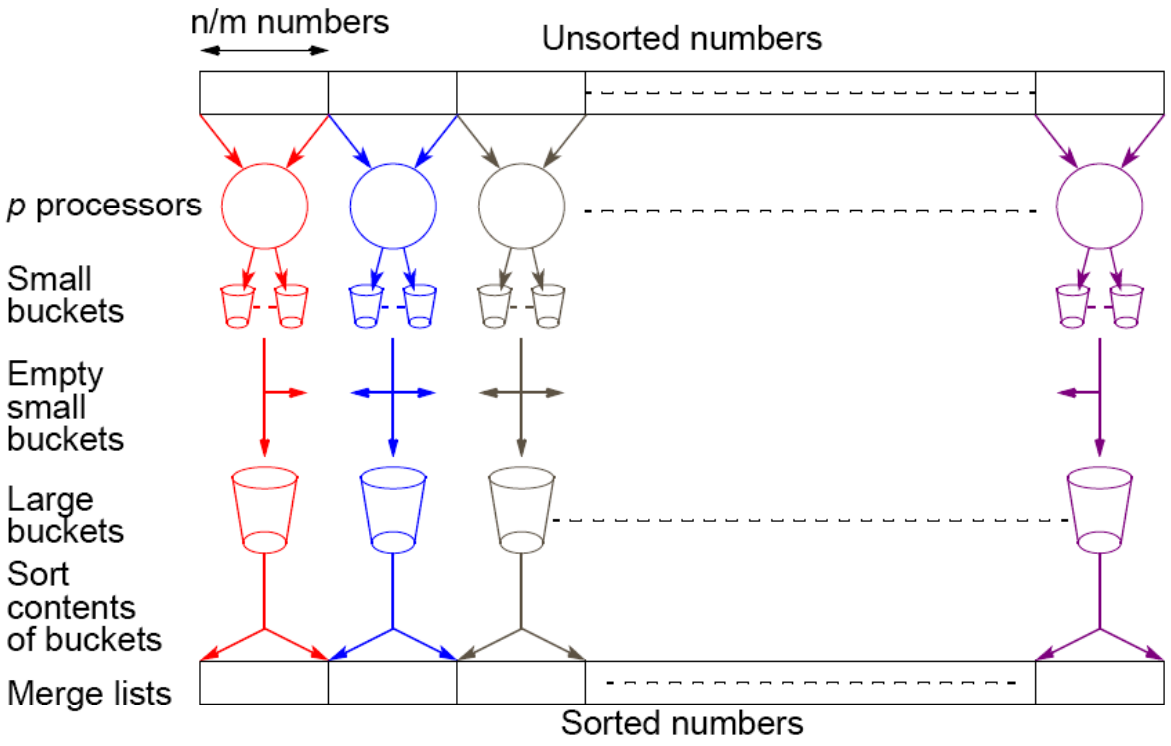


Καλύτερη παράλληλη λύση

- Η πιο αποδοτική παράλληλη υλοποίηση για το bucket sort διαιρεί τον αρχικό πίνακα σε p περιοχές και κάθε διεργασία αναλαμβάνει μία από αυτές τις περιοχές. Η i-οστη περιοχή (i=0...p-1) περιέχει τα στοιχεία $x_{i*n/p}$ ...$x_{(i+1)*n/p-1}$ του αρχικού πίνακα.

- Κάθε διεργασία διατηρεί *p* "μικρούς" κάδους και χωρίζει τα στοιχεία της περιοχής του σε αυτούς τους μικρούς κάδους ακολουθώντας τη λογική του bucket sort δηλ. ο κάδος i (i=0,...,p-1) περιέχει στοιχεία στη περιοχή [i*α/p...(i+1)*α/p-1].

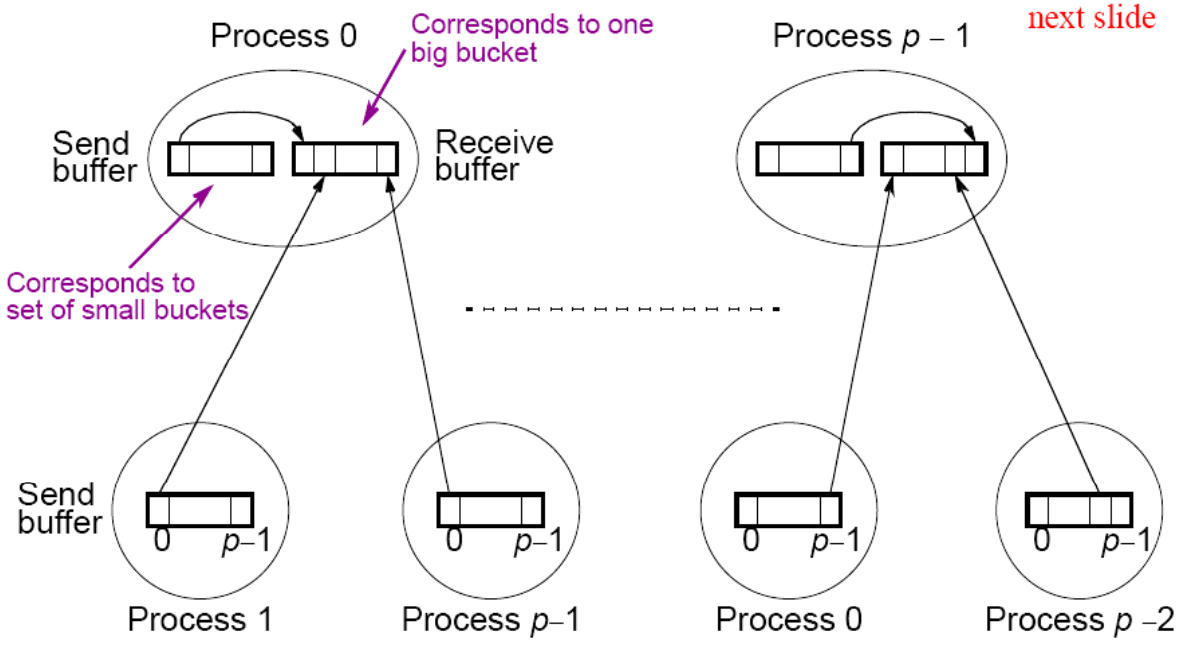
- Στη συνέχεια οι μικροί κάδοι αδειάζουν σε p τελικούς κάδους όπου ο τελικός κάδος i (i=0,...,p-1) περιέχει στοιχεία στη περιοχή [i*α/p...(i+1)*α/p-1]. Κάθε διεργασία αναλαμβάνει ένα από αυτούς τους κάδους.
- Πιο συγκεκριμένα, κάθε διεργασία στέλνει τα περιεχόμενα του i-οστού μικρού κάδου στην i-οστη διεργασία η οποία έχει υπ' ευθύνη της τον τελικό κάδο i.
- Στη συνέχεια κάθε διεργασία ταξινομεί τα περιεχόμενα του τελικού της κάδου και κατόπιν τα ταξινομημένα περιεχόμενα των τελικών κάδων συγχωνεύονται στη τελική λίστα.
- Σε αυτή την επικοινωνία κάθε διεργασία στέλνει διαφορετικά δεδομένα σε κάθε άλλη διεργασία. Αυτό το μοτίβο επικοινωνίας είναι γνωστό ως all-to-all broadcast



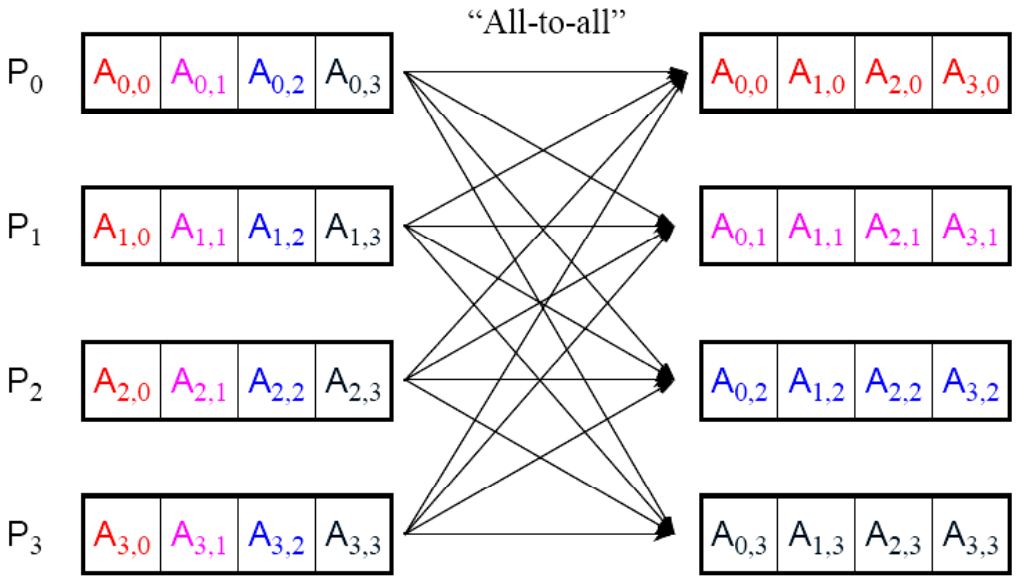
"all-to-all" broadcast

Κάθε διεργασία στέλνει διαφορετικά δεδομένα σε κάθε άλλη διερνασία

See also next slide

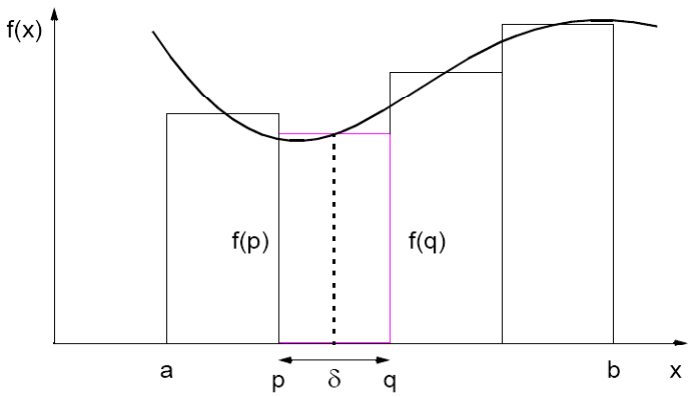


Η μεταφορά των δεδομένων στην “all-to-all” επικοινωνία μπορεί να ειδωθεί ως αναστροφή ενός πίνακα. Συγκεκριμένα, κάθε γραμμή αυτού του πίνακα αντιστοιχεί στα δεδομένα μίας διεργασίας και η all-to-all επικοινωνία μεταφέρει τις γραμμές αυτού του πίνακα στις στήλες

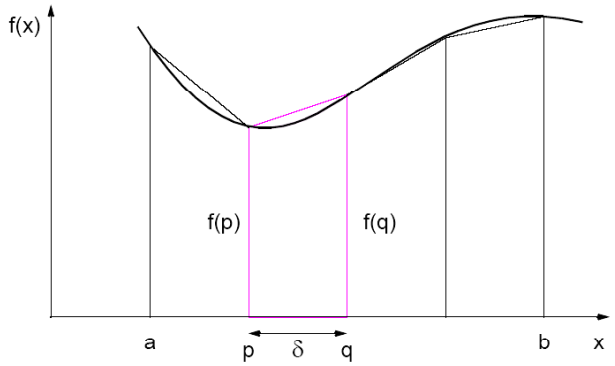


Αριθμητική ολοκλήρωση με τη χρήση ορθογωνίων

Για τον υπολογισμό του ολοκληρώματος $\int_a^b f(x)dx$, χωρίζουμε το διάστημα ολοκλήρωσης [a...b] σε διαστήματα μήκους δ. Στη συνέχεια για κάθε υποδιάστημα [p,..q] προσεγγίζουμε το ολοκλήρωμα της συνάρτησης με το εμβαδό του ορθογωνίου που έχει πλάτος δ και ύψος ίσο με f((p+q)/2). Το συνολικό ολοκλήρωμα προκύπτει κατά προσέγγιση από το άθροισμα των εμβαδών αυτών των ορθογωνίων.



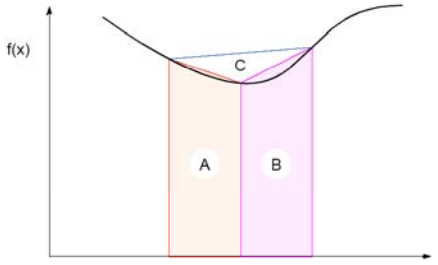
Αριθμητική ολοκλήρωση με τη χρήση τραπεζίων



- Εναλλακτικά μπορούμε να χρησιμοποιήσουμε τραπέζια αντί για ορθογώνια
- Και στις δύο περιπτώσεις, ο υπολογισμός στα διάφορα υποδιαστήματα μπορεί να μοιραστεί ισομερώς στις διάφορες διεργασίες. Επειδή το πλήθος των υποδιαστημάτων είναι εκ των προτέρων γνωστό, μπορούμε να ακολουθήσουμε στατική καταχώριση έργων στις διεργασίες.
- Το πρόβλημα με αυτό τον τρόπο προσέγγισης ολοκληρώματος είναι ότι σε περιπτώσεις συναρτήσεων που δεν έχουν ομαλό σχήμα, η προσέγγιση δεν είναι καλή

Προσαρμοζόμενος Τετραγωνισμός (Adaptive Quadrature)

- Στη τεχνική αυτή, η λύση προσαρμόζεται στο σχήμα της γραφικής παράστασης. Τα υποδιαστήματα συνεχώς διαιρούνται μέχρι η προσέγγιση να είναι ικανοποιητική. Συγκεκριμένα σε κάθε βήμα ελέγχεται αν το εμβαδό της μεγαλύτερης από τις δύο περιοχές A, B, C (C στο σχήμα) πλησιάζει το άθροισμα των εμβαδών των δύο άλλων περιοχών (A+B).
- Επειδή δεν είναι εκ των προτέρων γνωστό το πλήθος των τελικών διαστημάτων που προκύπτει από τις διαδοχικές διαιρέσεις, απαιτείται δυναμική καταχώριση των υπολογισμών στις διαθέσιμες διεργασίες.
- Μία πιθανή λύση είναι αρχικά το διάστημα ολοκλήρωσης να διαιρεθεί σε ίσου μήκους διαστήματα και να ανατεθεί ένα διάστημα σε κάθε διεργασία
- Κάθε διεργασία εκτελεί τον παραπάνω έλεγχο, και στη περίπτωση διαίρεσης, η διεργασία κρατάει το ένα υποδιάστημα και επιστρέφει το δεύτερο στη δεξαμενή των έργων (work pool).
- Αυτή η διαδικασία επαναλαμβάνεται μέχρι η προσέγγιση να είναι ικανοποιητική.
- Στη συνέχεια η διεργασία υπολογίζει το εμβαδό στο συγκεκριμένο διάστημα και το αποτέλεσμα επιστρέφεται στη master διεργασία.
- Στη συνέχεια, η διεργασία αναλαμβάνει το επόμενο διαθέσιμο διάστημα από τη δεξαμενή έργων και η παραπάνω διαδικασία επαναλαμβάνεται.
- Η master διεργασία υπολογίζει το συνολικό ολοκλήρωμα αθροίζοντας τα αποτελέσματα που στέλνουν οι slave διεργασίες



Simple program to compute π

Using C++ MPI routines

```
/*****************************************************************************
pi_calc.cpp calculates value of pi and compares with actual
value (to 25digits) of pi to give error. Integrates function f(x)=4/(1+x^2).
July 6, 2001 K. Spry CSCI3145
*****************************************************************************/
#include <math.h> //include files
#include <iostream.h>
#include "mpi.h“

void printit();                                          //function prototypes
int main(int argc, char *argv[])
{
double actual_pi = 3.14159265358979323846264 3;
                                                         //for comparison later
int n, rank, num_proc, i;
double temp_pi, calc_pi, int_size, part_sum, x;
char response = 'y', resp1 = 'y';
MPI::Init(argc, argv);                                   //initiate MPI
```

```
num_proc = MPI::COMM_WORLD.Get_size();
rank = MPI::COMM_WORLD.Get_rank();
if (rank == 0) printit();          /* I am root node, print out welcome */

while (response == 'y') {
        if (resp1 == 'y') {
        if (rank == 0) {          /*I am root node*/
        cout <<"_____" <<endl;
        cout <<"\nEnter the number of intervals: (0 will exit)" << endl;
        cin >> n;}
} else n = 0;

MPI::COMM_WORLD.Bcast(&n, 1, MPI::INT, 0);     //broadcast n
if (n==0) break; //check for quit condition
```

```
  else {
  int_size = 1.0 / (double) n;                    //calcs interval size
  part_sum = 0.0;

  for (i = rank + 1; i <= n; i += num_proc)
   {                                              //calcs partial sums
          x = int_size * ((double)i - 0.5);
          part_sum += (4.0 / (1.0 + x*x));
  }
  temp_pi = int_size * part_sum;
                          //collects all partial sums computes pi

  MPI::COMM_WORLD.Reduce(&temp_pi,&calc_pi, 1,
  MPI::DOUBLE, MPI::SUM, 0);
```

```
if (rank == 0) {                                    /*I am server*/
cout << "pi is approximately " << calc_pi
<< ". Error is " << fabs(calc_pi - actual_pi)
<< endl
<<"_____________________________________________"
<< endl;
}
}                                                   //end else
if (rank == 0) { /*I am root node*/
cout << "\nCompute with new intervals? (y/n)" << endl; cin >> resp1;
}
}//end while
MPI::Finalize();                                    //terminate MPI
return 0;
}                                                   //end main
```

```
//functions
void printit()
{
cout << "\n*********************************" << endl
<< "Welcome to the pi calculator!" << endl
<< "Programmer: K. Spry" << endl
<< "You set the number of divisions \nfor estimating the
integral:
\n\tf(x)=4/(1+x^2)"
<< endl
<< "*********************************" << endl;
}                   //end printit
```

Το πρόβλημα *N*-Body

Κεντρικό ζητούμενο σε αυτό το πρόβλημα είναι να ευρεθούν οι θέσεις και οι κινήσεις των σωμάτων στο χώρο όταν εξασκούνται βαρυτικές δυνάμεις μεταξύ των σωμάτων σύμφωνα με τους νευτώνειους νόμους της Φυσικής.
Η ελκτική δύναμη που ασκείται μεταξύ δύο σωμάτων με μάζες m_a και m_b δίνεται από την ακόλουθη σχέση:

$$F = \frac{Gm_a m_b}{r^2}$$

Όπου G είναι σταθερά και r είναι η απόσταση μεταξύ δύο σωμάτων. Επίσης, σύμφωνα με το δεύτερο νόμο του Νεύτωνα, όταν σε ένα σώμα ασκείται δύναμη F το σώμα αποκτά επιτάχυνση γ σύμφωνα με τον τύπο:

$$F=m\gamma$$

όπου m είναι η μάζα του σώματος.
Για ένα μικρό διάστημα Δt η επιτάχυνση γ μπορεί να γραφεί ως

$$\gamma = \frac{v^{t+1} - v^t}{\Delta t}$$

όπου v^{t+1} και v^t είναι οι ταχύτητα του σώματος τις χρονικές στιγμές t+1 και t. Η νέα ταχύτητα v^{t+1} δίνεται από τη σχέση

$$v^{t+1} = v^t + \frac{F\Delta t}{m}$$

Το πρόβλημα N-body

Στο ίδιο χρονικό διάστημα Δ*t*, η θέση του σώματος αλλάζει σύμφωνα με τον τύπο

$$x^{t+1} - x^t = v\Delta t$$

όπου x^t είναι η θέση του σώματος τη χρονική στιγμή *t*.
Μόλις τα σώματα μετακινηθούν στις νέες τους θέσεις οι δυνάμεις αλλάζουν και επομένως ο παραπάνω υπολογισμός πρέπει να επαναληφθεί

Ακολουθιακό κώδικας για το πρόβλημα N-body

Συνολικά, ο υπολογισμός για το N-body πρόβλημα μπορεί να περιγραφεί ως ακολούθως

```
for (t = 0; t < tmax; t++)            /* for each time period */
        for (i = 0; i < N; i++) {     /* for each body */
              F = Force_routine(i); /* compute force on ith body */
              v[i]new = v[i] + F * dt / m;
                                      /* compute new velocity */
              x[i]new = x[i] + v[i]new * dt;   /* and new position */
        }
for (i = 0; i < nmax; i++) {          /* for each body */
          x[i] = x[i]new;             /* update velocity & position*/
          v[i] = v[i]new;
}
```

Παράλληλη υλοποίηση

• Ο ακολουθιακός αλγόριθμος έχει πολυπλοκότητα $O(N^2)$ (για μία επανάληψη) καθώς κάθε ένα από τα N σώματα επηρεάζεται από τα υπόλοιπα $N - 1$.

• Σε μία απλή στατική κατανομή έργων, κάθε μία από τις p συνολικά διεργασίες αναλαμβάνει τον υπολογισμό των ασκούμενων δυνάμεων σε n/p σώματα.

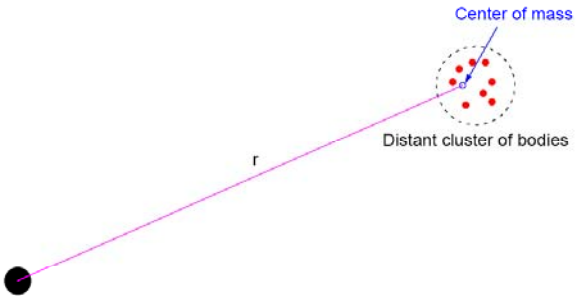
• Αυτή η κατανομή συνεπάγεται πυκνή κυκλοφορία μηνυμάτων (all-to-all broadcast) αφού κάθε διεργασία πρέπει να στέλνει σε όλες τις υπόλοιπες τις νέες συντεταγμένες των σωμάτων μετά την μετατόπιση τους.

• Εναλλακτικά μπορούμε να ακολουθήσουμε τη τεχνική master-slave με τη master διεργασία να συντηρεί μία δεξαμενή έργων. Τα έργα σε αυτή την περίπτωση είναι ο υπολογισμός της ελκτικής δύναμης μεταξύ δύο σωμάτων. Με N σώματα, υπάρχουν συνολικά N(N+1)/2 τέτοια ζεύγη και επομένως N(N+1)/2 έργα προς εκτέλεση

• Κάθε slave διεργασία αναλαμβάνει το επόμενο διαθέσιμο έργο από τη δεξαμενή της master διεργασίας και το αποτέλεσμα της επεξεργασίας (η ελκτική δύναμη μεταξύ των δύο σωμάτων) επιστρέφεται πίσω στη master διεργασία

• Η master διεργασία προσθέτει τα αποτελέσματα των slave διεργασιών για να υπολογίσει τη συνισταμένη δύναμη σε κάθε σώμα

• Η χρονική πολυπλοκότητα του ακολουθιακού αλγόριθμου μπορεί να μειωθεί με τη παρατήρηση ότι μία ομάδα (cluster) από απομακρυσμένα σώματα μπορούν να προσεγγιστούν ως ένα μακρινό αντικείμενο του οποίου η συνολική μάζα είναι τοποθετημένη στο κέντρο βάρους της ομάδας:

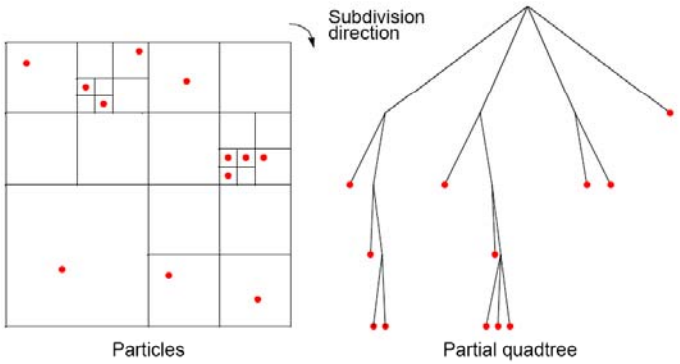


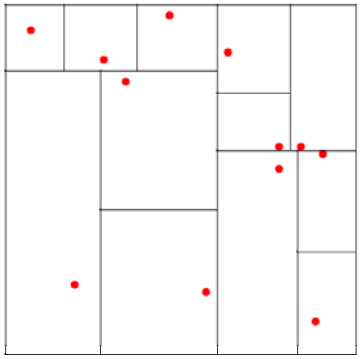
• Ο αλγόριθμος Barnes-Hut βασίζεται στη προηγούμενη παρατήρηση και έχει ως εξής:
• Στην αρχή όλος ο χώρος με τα N σώματα περιέχεται σε ένα μόνο κύβο.
• Στη συνέχεια, αυτός ο κόμβος διαιρείται σε οκτώ υποκύβους
• Αν ένας υποκύβος δεν περιέχει σώματα, ο υποκύβος διαγράφεται και ο αλγόριθμος δεν ασχολείται με αυτόν περαιτέρω.
• Εάν ένας υποκύβος περιέχει ένα σώμα, αυτός ο υποκύβος κρατείται.
• Εάν ένας υποκύβος περιέχει περισσότερα του ενός σώματα, ο κύβος αυτός διαιρείται αναδρομικά μέχρι κάθε υποκύβος να περιέχει ένα μόνο σώμα.

• Η προηγούμενη διαδικασία δημιουργεί ένα οκταδικό δέντρο (*octtree)* δηλ. ένα δέντρο όπου κάθε κόμβος έχει το πολύ οκτώ παιδιά
• Κάθε κόμβος του δέντρου αντιστοιχεί σε ένα υποκύβο και τα φύλλα αυτού του δέντρου αντιστοιχούν στους υποκύβους που περιέχουν ένα μόνο σώμα.
• Μετά τη κατασκευή του δέντρου, σε κάθε κόμβο αποθηκεύεται η συνολική μάζα και το κέντρο βάρους του υποκύβου που αντιστοιχεί στο κόμβο.
• Η δύναμη σε κάθε σώμα μπορεί να προσδιορισθεί με τη διάσχιση του δέντρου με αρχή τη ρίζα μέχρι να συναντήσουμε ένα κόμβο όπου η παραπάνω προσέγγιση μπορεί να εφαρμοσθεί.
• Συγκεκριμένα το κριτήριο για την εφαρμογή της προσέγγισης είναι η απόσταση του υπό εξέταση σώματος r από το κέντρο βάρους του cluster να ικανοποιεί την ακόλουθη ανισότητα:

$$r \geq \frac{d}{\theta}$$

όπου d είναι η διάσταση του υποκύβου και θ είναι μία σταθερά συνήθως μικρότερη της μονάδας.
• Τόσο η κατασκευή του δέντρου όσο και ο υπολογισμός του δέντρου με βάση το δέντρο έχει πολυπλοκότητα O(N log N).
• Με τη μετακίνηση των σωμάτων, το octtree θα πρέπει πάλι να κατασκευαστεί.
• Η παράλληλη υλοποίηση του αλγόριθμου Barnes-Hut παρουσιάζει δυσκολίες. Το οκταδικό δέντρο έχει μη κανονική μορφή και η κατανεμημένη υλοποίηση του δεν είναι εύκολη και συνήθως δημιουργεί προβλήματα ανισοκατανομής φόρτου εργασίας στις διεργασίες.
• Επίσης, η διάτρεξη του κατανεμημένου δέντρου για τον υπολογισμό των δυνάμεων που ασκούνται στα σώματα έχει ως αποτέλεσμα τη συχνή ανταλλαγή μηνυμάτων μεταξύ των διεργασιών





Orthogonal Recursive Bisection

• Στο σχήμα βλέπουμε ένα παράδειγμα αναδρομικής διαίρεσης του δισδιάστατου χώρου μέχρι κάθε τετράγωνο να περιέχει ένα μόνο σώμα
• Εναλλακτικά, μπορούμε να διαιρέσουμε το χώρο κατά τέτοιο τρόπο ώστε σε κάθε επίπεδο της δενδρικής δομής τα υποδέντρα να περιέχουν το ίδιο πλήθος σωμάτων. Η τεχνική αυτή είναι γνωστή ως Αναδρομική Ορθογώνια Διχοτόμηση.
• Για 2-διάστατο χώρο, η διχοτόμηση αυτή επιτυγχάνεται ως εξής:
• Πρώτα, βρίσκεται η κάθετη γραμμή η οποία διαιρεί την περιοχή σε δύο περιοχές με το ίδιο πλήθος σωμάτων.
• Για κάθε μία από αυτές τις περιοχές, προσδιορίζεται η οριζόντια γραμμή που διαιρεί την περιοχή σε δύο νέες περιοχές με το ίδιο πλήθος σωμάτων.
• Αυτή η διαδικασία επαναλαμβάνεται μέχρι να προκύψουν περιοχές που περιέχουν ένα μόνο σώμα
• Αυτός ο τρόπος διχοτόμησης του χώρου βοηθά στην εξισορρόπηση φορτιού (load balancing) σε μία παράλληλη υλοποίηση της μεθόδου