Χρονοεπίδοση και αξιολόγηση απόδοσης εφαρμογών

Συμπεριφορά απόδοσης εφαρμογών
Benchmarks, Profiling

Δρ. Δημήτρης Ντελλής

GRNET

ntell [at] grnet.gr

European Union
European Regional Development Fund
ΠΕΠ
PRACE GR
NSRF 2007-2013

Περιεχόμενα

- Χρονοεπίδοση και κλιμάκωση εφαρμογών
- Παράγοντες που επηρεάζουν την επίδοση
- Αναζήτηση σημείων κώδικα που αποτελούν bottleneck.

Γενικά

- Benchmark : μελέτη της ταχύτητας εκτέλεσης μιας εφαρμογής
- Scaling : Η δυνατότητα μιας εφαρμογής να μικραίνει ο χρόνος εκτέλεσης όσο μεγαλώνει ο αριθμός των cores που χρησιμοποιούνται, και παράγοντες που την επηρεάζουν.
- Τι μετράμε και πόσο αντιπροσωπευτικό είναι
- Profiling για εύρεση σημείων/διαδικασιών που επιβραδύνουν την εφαρμογή.

Benchmarking

- Έστω κάποια εφαρμογή στην οποία έχουν εφαρμοστεί όλες οι πρακτικές βελτιτοποίησης για σειριακούς κώδικες.
- Μετράμε το χρόνο εκτέλεσης της σειριακής εφαρμογής. Ένα απλό παράδειγμα είναι η επίλυση γραμμικού συστήματος εξισώσεων.
- Πιθανότατα όλοι έχουν συνειδοποιήσει κάποια στιγμή ότι σειριακά ο χρόνος επίλυσης εξαρτάται από το μέγεθος του συστήματος που επιλύεται, όσο μεγαλύτερο το μέγεθος τόσο μεγαλύτερος και ο χρόνος εκτέλεσης.

- Όταν μεταβαίνουμε σε παράλληλη εκτέλεση θεωρητικά αναμένουμε να ελατώνεται ο χρόνος εκτέλεσης με τον αριθμό των cores που χρησιμοποιούμε, (ιδανικό scaling). Η πραγματικότητα είναι διαφορετική.

Benchmarking

- Όσο πιο πολλά process παίρουν μέρος σε μια παράλληλη διαδικασία, τόσο μικραίνει ο χρόνος υπολογισμού του καθενός process.
- Ασχέτως του πόσα processes χρησιμοποιούμε, χρειάζεται επικοινωνία μεταξύ των process. Όσο περισότερα procs συμπεριλαμβάνει μια παράλληλη εργασία, τόσο μεγαλύτερη είναι η επικοινωνία μεταξύ τους.

- Αρα, όσο μεγαλώνει ο αριθμός των processes έχουμε : το χρόνο υπολογισμού να μικραίνει και το χρόνο επικοινωνίας να αυξάνει. Κάποια στιγμή, αυξάνοντας τον αριθμό των procs η επικοινωνία θα γίνει ο κύριος παράγοντας για την επίδοση.
- Έστω ότι έχουμε ένα vector ή array και θέλουμε π.χ. να διπλασιάσουμε τις τιμές του κάθε στοιχείου, και ότι δεν έχουμε πρόβλημα επικοινωνίας. Στην ιδανική περίπτωση, πρέπει ο αριθμός των στοιχείων του vector/array να διαιρείται ακριβώς με τον αριθμό των processes. Αλλιώς εισάγεται η έννοια του load imbalance : Δεν έχουν όλα τα processes τον ίδιο αριθμό πράξεων να κάνουν.

- Όσο πιο κοντά είναι μεταξύ των process ο αριθμός των πράξεων που έχουν να κάνουν, τόσο μικρότερη είναι η επιρροή στην ταχύτητα.
- Στον όρο πράξεων ανά process μετράμε όλες τις πιθανές πράξεις, OpenMP/Threads συμπεριλαμβανομένων.

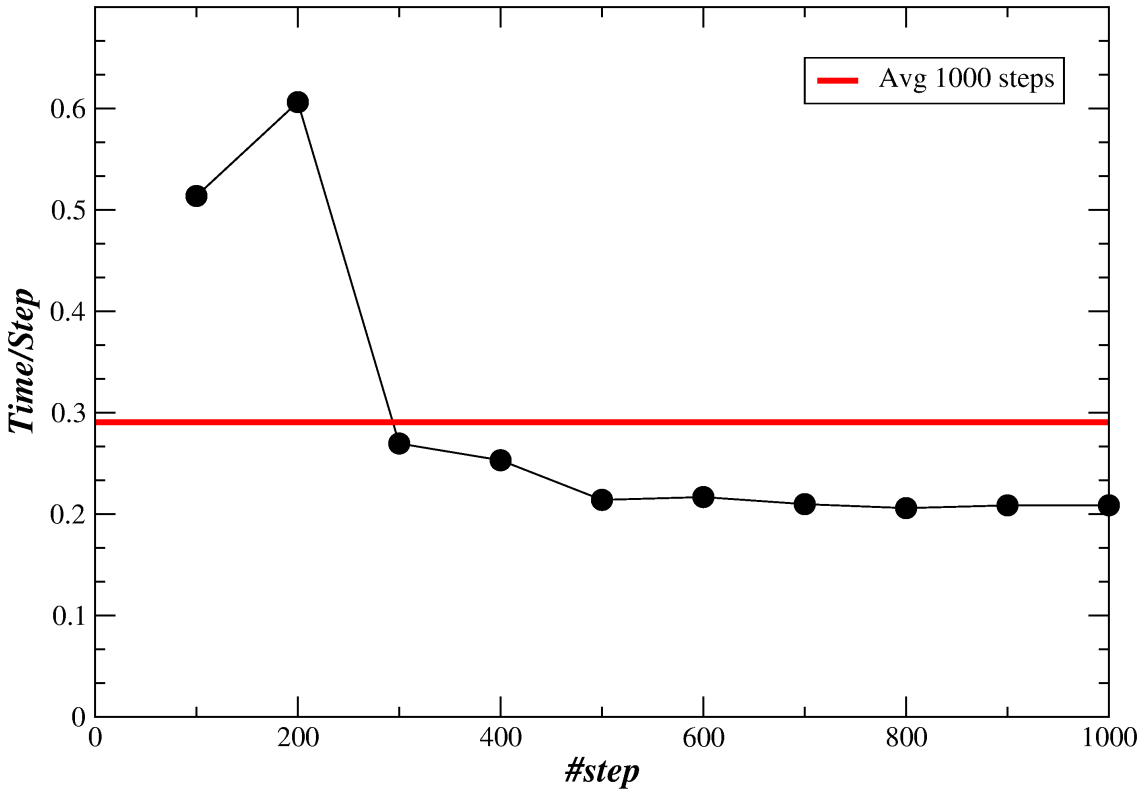
Benchmarking : Μετράμε σωστά ?

- Είναι αυτό που μετράμε αντιπροσωπευτικό της απόδοσης ?
  - Συνήθως στα benchmarks, μετράμε κάποια λίγα βήματα στην έναρξη των υπολογισμών. Συνηθίζεται στα πρώτα βήματα να γίνονται πολύ περισσότερα από ότι αργότερα, initialization, load balancer, auto tuning, κλπ.
  - Προφανώς αυτά τα βήματα πρέπει να μείνουν εκτός υπολογισμών, ή να τρέξουμε αρκετά ”τυπικά” βήματα ώστε η επιρροή του/των πρώτων να είναι μικρή στο συνολικό χρόνο εκτέλεσης.

Benchmarking : Μετράμε σωστά ? Μερικά παραδείγματα

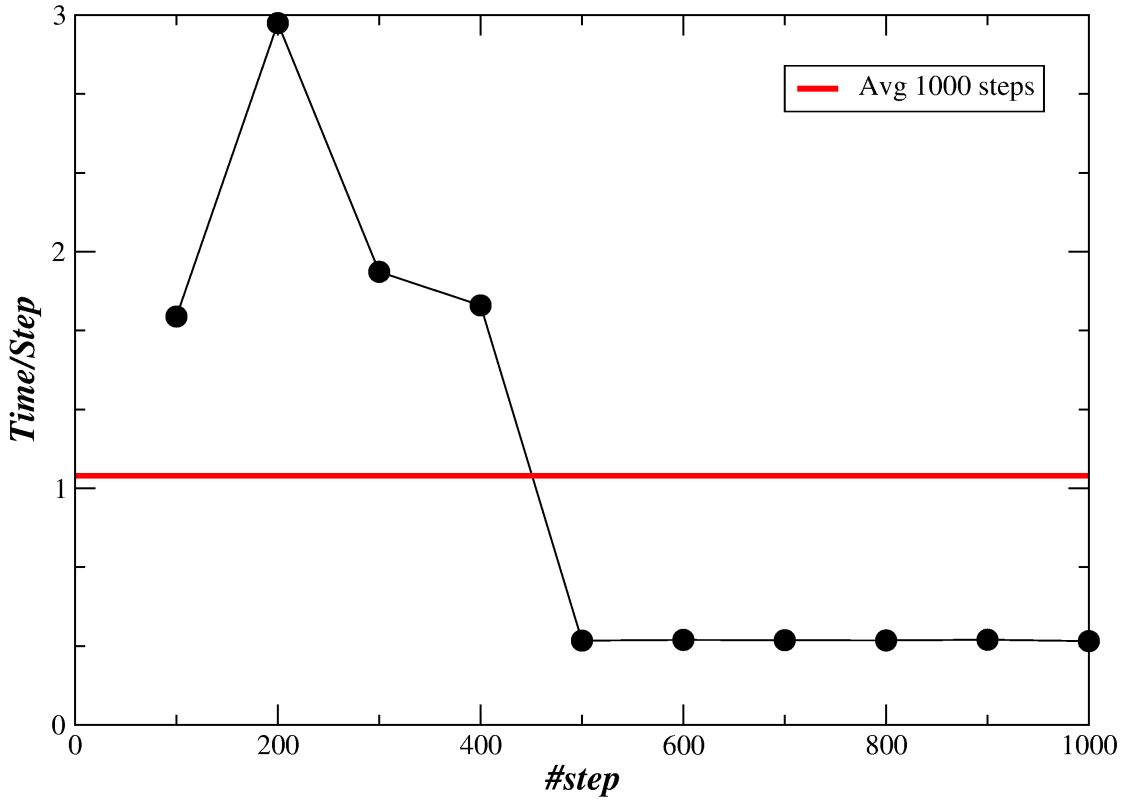
- Κώδικας : NAMD
- Machines HeCToR (Cray XE6), Juqueen (Bluegene/Q) 4096 cores
- Performance during first 1000 steps in averages of 100, average over 1000 steps.

Benchmarking : NAMD, 4096 cores, HeCToR (Cray XE6)

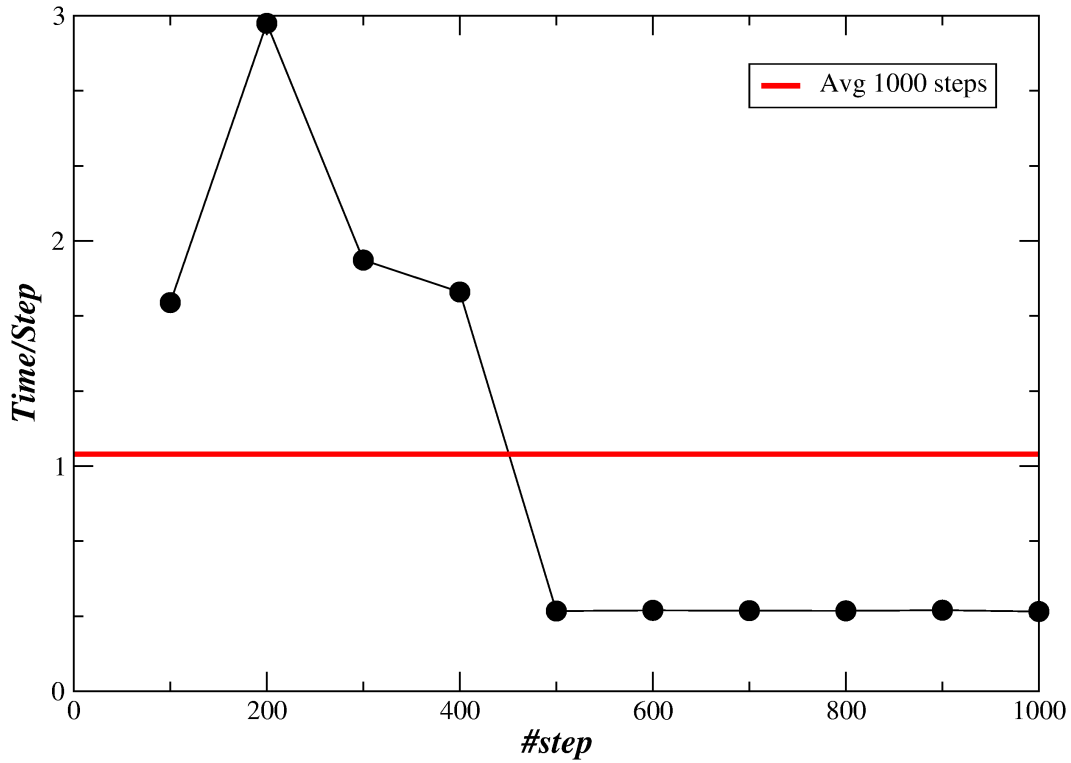


Benchmarking : Κάτι θα γινόταν στο σύστημα εκείνη την ώρα η πρώτη σκέψη.....

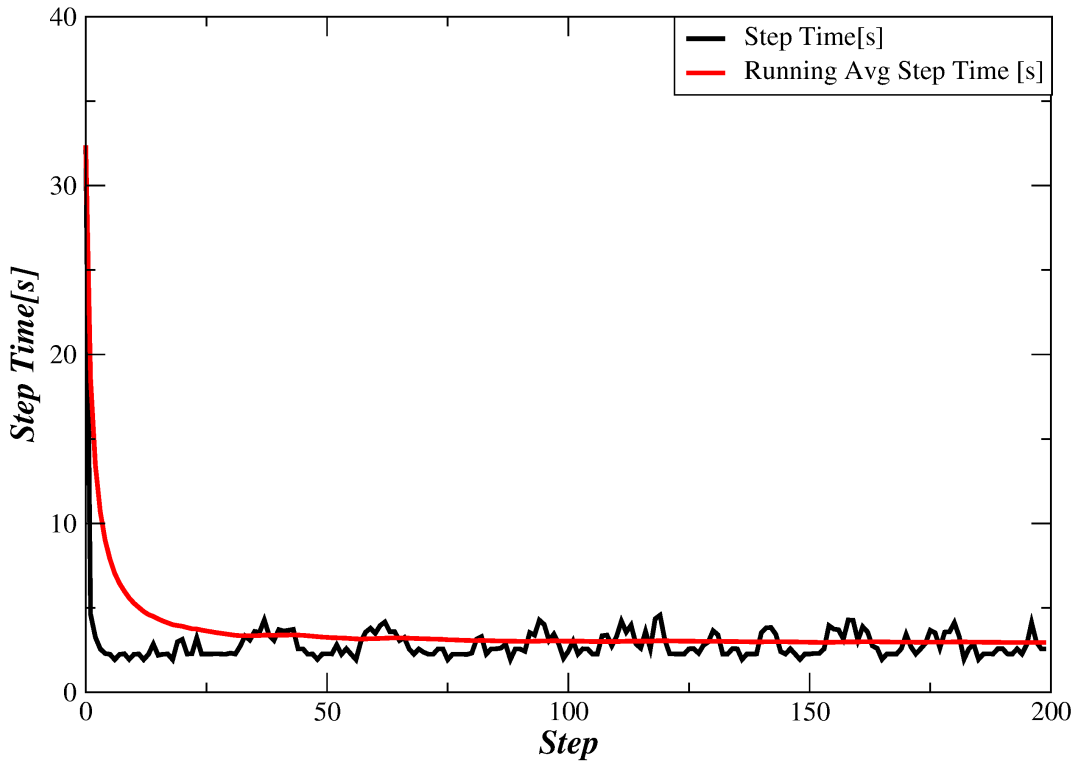
Benchmarking : NAMD, 4096 cores, Juqueen(Bluegene/Q)



Benchmarking : NAMD, 4096 cores, SuperMUC (IBM similar to ARIS but 8C)



Benchmarking : Και ένα παράδειγμα από το ARIS, CP2K, 64 Water, 40 cores, 200 MD Steps



Benchmarking : NAMD, 4096 cores on all

- Στα πρώτα 100-400 iterations δουλεύει ο load balancer, μετά από αυτά ότι load imbalance υπάρχει από τις αρχικές συνθήκες έχει ισοροπήσει, και υπάρχει σταθερό perfonmance.
- Αυτό που θα δεί τελικά ο χρήστης σε ένα τυπικό run μερικών εκατομυρίων iterations, είναι το όριο στο οποίο συγκλίνει το performance.

Benchmarking : CP2K, 40 cores, 64 Waters, TZVP, 200 MD Steps, on ARIS

- Στο πρώτο βήμα γίνονται : αρχικοποίηση, πολύ περισότερα εσωτερικά iterations, κλπ
- Αυτό που θα δεί τελικά ο χρήστης σε ένα τυπικό run μερικών χιλιάδων ή μυριάδων iterations, είναι το όριο στο οποίο συγκλίνει το performance.

Benchmarking : Συμπέρασμα

- Πρέπει να είμαστε προσεκτικοί στο τι μετράμε στην αξιολόγηση προγραμμάτων, συστημάτων, μεθόδων κλπ.
- Όπως είδαμε χτές, υπάρχουν περιπτώσεις όπως το WRF, όπου συστηματικά εμφανίζονται ανά διαστήματα steps με (σημαντικά) αυξημένο χρόνο εκτέλεσης. Στην περίπτωση αυτή, σαφώς πρέπει να λαμβάνονται υπόψιν τα βήματα αυτά στην αξιολόγηση, επειδή επηρεάζουν το performance που θα δεί ο τελικός χρήστης στις μεγάλης διάρκειας προσομοιώσεις του.

Profiling : mpiP

- mpiP : Lightweight, Scalable MPI Profiling
- http://mpip.sourceforge.net/
- Είναι μια βιβλιοθήκη, η οποία χωρίς ΚΑΜΙΑ αλλαγή στον κώδικα της εφαρμογής, εφόσον γίνει link σωστά αντικαθιστά τις MPI κλήσεις, κάνει μετρήσεις κατά την εκτέλεση και στο τέλος βγάζει report.
- Τρόπος Χρήσης :

Profiling : mpiP

- Φυσιολογικό Compilation : mpicc|mpif90 -O source.[c|f] -o exe
- Compilation για profiling με mpiP :
  - module load mpiP
  - mpicc|mpif90 -g -O source.[c|f] -L$MPIPROOT/lib -lunwind -lbfd -o exe
- Τρέχουμε την εφαρμογή ως συνήθως
- Αν όντως έχει γίνει σωστά το link, στην αρχή της εκτέλεσης εμφανίζεται ένα μήνυμα

```
mpiP:
mpiP: mpiP: mpiP V3.4.1 (Build Sep  7 2015/16:33:51)
mpiP: Direct questions and errors to mpip-help@lists.sourceforge.net
mpiP:
```

Profiling : mpiP

- Μετά την εκτέλεση, εμφανίζεται στην directory που τρέξαμε ένα text file με όνομα : EXE.MPITASKS.PID.NUMBER.mpiP
- Το file περιέχει το summary και την ανάλυση του profile.
- Το καλό του mpiP είναι ότι πολύ γρήγορα μπορούμε να βρούμε ποια είναι τα σημεία του κώδικα που έχουμε τις top καθυστερήσεις, και πιθανότατα και το λόγο.

Profiling : mpiP, σημαντικά σημεία του summary

```
@ Report generation          : Collective
@ MPI Task Assignment        : 0 node049
@ MPI Task Assignment        : 1 node049
.....
@ MPI Task Assignment        : 19 node049
.....
---------------------------------------------------------------------------
@--- MPI Time (seconds) ------------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0   1.71e+03       13.3     0.78
........
  19   1.71e+03        163     9.55
   *   3.42e+04   2.04e+03     5.97
.....
---------------------------------------------------------------------------
@--- Callsites: 43 -----------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 parallel.c           759 par_broadcast           Bcast
  2   0 parallel.c           295 par_imax                Allreduce
.....
```

```
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%      COV
Allreduce              32   2.03e+06    5.94   99.44     0.75
.....
---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total       Avrg  Sent%
Allreduce              10       4000   3.91e+09   9.77e+05  96.58
Bcast                  27         20   1.28e+07    6.4e+05   0.32
Bcast                  30         20   1.28e+07    6.4e+05   0.32
```

- Εξάσκηση με mpiP, δοκιμάστε είτε δικό σας κώδικα είτε το χτεσινό παράδειγμα για BLAS/Scalapack. Φτιάξτε το Makefile για mpiP, κάντε compile, τρέξτε την εφαρμογή, δείτε τα αποτελέσματα.
- Απορίες - συζήτηση

- module load scalasca
- Compilation για scalasca : scalasca -instrument mpicc|mpif90 -O source.[c|f] -o exe
- Run : Στο SLURM, αλλάζει μόνο το run : scalasca -analyze srun EXE ARGS
- Παράγεται μιά directory με τα results, scorep_EXENAME_MPITASKS_sum
- Results : scalasca -examine το όνομα της directory
- ΧΡΕΙΑΖΕΤΑΙ X11.
- Πρακτική εξάσκηση με scalasca.

- Εξάσκηση με Scalasca, δοκιμάστε είτε δικό σας κώδικα είτε το χτεσινό παράδειγμα για BLAS/Scalapack. Φτιάξτε το Makefile για Scalasca, κάντε compile, τρέξτε την εφαρμογή, δείτε τα αποτελέσματα.
- Απορίες συζήτηση.

- Εργαλεία όπως Allinea DDT (€, $)
- Με τα γνωστά εργαλεία, όχι ότι καλύτερο αλλά δουλεύει καλά για μικρό αριθμό MPI Tasks. mpirun -np 4 xterm -e gdb EXE.
- Ανοίγει ένα X-terminal που τρέχει GBD EXE για κάθε ένα MPI Task.

Ερωτήσεις ?