Optimisation de programmes parallélisés de calcul scientifique

Journée ProDev 2018

Edlira Nano
Calanques BioInformatics Core Facility, INSERM TAGC et IBDM

Plan

Deux programmes hérités à optimiser / étendre

La metaphysique de l'optimisations

Changer de langage de programmation ?

HH-MOTiF

HH-MOTiF : de novo detection of short linear motifs in proteins by Hidden Markov Model comparisons (R. Prytuliak and al., Nucleic Acids Research, 2017)

HH-MOTiF

HH-MOTiF : de novo detection of short linear motifs in proteins by Hidden Markov Model comparisons (R. Prytuliak and al., Nucleic Acids Research, 2017)

Utilise des informations d'évolution en créant des chaînes de Markov cachées (Hidden Markov Models) pour chaque protéine et ses orthologues, afin de trouver des SLiMs (Short Linear Motifs : sites protéiques fonctionnels).

HH-MOTiF

HH-MOTiF : de novo detection of short linear motifs in proteins by Hidden Markov Model comparisons (R. Prytuliak and al., Nucleic Acids Research, 2017)

Utilise des informations d'évolution en créant des chaînes de Markov cachées (Hidden Markov Models) pour chaque protéine et ses orthologues, afin de trouver des SLiMs (Short Linear Motifs : sites protéiques fonctionnels).

Ecrit en Python, utilise numpy et mpi4py. Fait appel à des programmes/librairies externes (écrites en C).

SLALOM : StatisticaL Analysis of Locus Overlap Method (publication à paraître)

Outil d'analyse de séries temporelles (time-series analysis) au sens large : séquence de caractères alphabétiques, séquence de mesures prise au cours du temps, tout type de données symboliques discrètes.

SLALOM : StatisticaL Analysis of Locus Overlap Method (publication à paraître)

Outil d'analyse de séries temporelles (time-series analysis) au sens large : séquence de caractères alphabétiques, séquence de mesures prise au cours du temps, tout type de données symboliques discrètes.

Identifie et analyse la qualité de chevauchement de séquences entre deux séries temporelles données. Analyses hautement paramétrables par l'utilisateur.

SLALOM : StatisticaL Analysis of Locus Overlap Method (publication à paraître)

Outil d'analyse de séries temporelles (time-series analysis) au sens large : séquence de caractères alphabétiques, séquence de mesures prise au cours du temps, tout type de données symboliques discrètes.

Identifie et analyse la qualité de chevauchement de séquences entre deux séries temporelles données. Analyses hautement paramétrables par l'utilisateur.

Ecrit en Python, paradigme objet, utilise comme structure de données les dictionnaires (default and ordered dict).

Plan

Deux programmes hérités à optimiser / étendre

La metaphysique de l'optimisations

Changer de langage de programmation ?

Quelles optimisations ?

- Interfaçage de Python en C (Cython) ?
- Changement de langage, dans le but de :
  - améliorer les performances de la parallélisation ?
  - faire des calculs plus rapides (accès mémoires et structures de données plus performantes) ?
  - prévoir les futures extensions (réseaux de graphes, machine learning, ...) ?

Python, Cython et parallélisation

mpi4py

Cours de Loic Gouarin sur MPI en Python (GDR Calcul, école *Python en calcul scientifique* 2010) : mpi4py est quasiment aussi performant que C + MPI

Python, Cython et parallélisation

mpi4py

Cours de Loic Gouarin sur MPI en Python (GDR Calcul, école *Python en calcul scientifique* 2010) : mpi4py est quasiment aussi performant que C + MPI

Optimisation Cython et Numpy

GDR Calcul, école *Python avancé en calcul scientifique*, 2013 :

- Konrad Hinsen, *Numpy* : optimisation des tableaux numpy avec l'approche de programmation par tableaux
- Loic Gouarin, *Cython* : optimisation (benchmarks) de code Python par passage en Cython

Oui mais ... non

Dans *Proc. of the 6th Workshop on Python for High-Performance and Scientific Computing* en 2016, Ross Smith publie *Performance of MPI codes written in Python with NumPy and mpi4py* et nous montre sur deux exemples d'algorithmes de calcul scientifique parallèle connus que :

- un code compilé (C++ et MPI) est plus rapide, beaucoup plus rapide, que du code Python utilisant numpy et mpi4py ;
- en Python on a au moins deux fois moins de lignes de code qu'en C++.

Plan

Deux programmes hérités à optimiser / étendre

La metaphysique de l'optimisations

Changer de langage de programmation ?

Comparaisons de langages de programmation

"No benchmark is perfect, but Computer Language Benchmarks Game is a good starting point" - *Algorithms*, R. Sedgewick and K. Wayne, 4th edition book.

The Computer Language Benchmarks Game
https ://benchmarksgame.alioth.debian.org

Rust, pourquoi ?

- Rust est un langage de programmation multi-paradigme (procédural, fonctionnel, orienté objet), compilé et orienté système.
- Rust est conçu pour être *un langage sécurisé, concurrent, pratique*.
- Il permet une gestion de mémoire manuelle très fine (comme en C, C++), qu'on peut choisir et adapter au programme, mais sûre (pas de segfault, pas de buffer overflow).
- Pas de GC : donc on peut faire de la programmation temps réel, programmation OS, programmation web browser sans crainte.
- Gestion de la concurrence intégrée au langage.
- Il est très expressif, s'inspire de certains traits de langages fonctionnels (les typeclasses d'Haskell, les pattern matching d'OCaml).

On organise une formation ? Dites-nous si vous êtes interessés.

Et pourquoi pas Go ?

En dehors des cas de programmation qui banissent l'usage du Garbage Collector, Go est une alternative très intéréssante. Sa syntaxe est simple, son compilateur est rapide et très facile d'utilisation (interfaçage avec C incroyablement facile).

Et pourquoi pas Go ?

En dehors des cas de programmation qui banissent l'usage du Garbage Collector, Go est une alternative très intéréssante. Sa syntaxe est simple, son compilateur est rapide et très facile d'utilisation (interfaçage avec C incroyablement facile).

Mais d'autres langages avec GC le sont (cf *Golang's real-time GC in Theory and Practice*, Will Sewell, Pusher blog, pour une comparaison des performances des GC et leurs optimisations).

Et pourquoi pas Go ?

En dehors des cas de programmation qui banissent l'usage du Garbage Collector, Go est une alternative très intéréssante. Sa syntaxe est simple, son compilateur est rapide et très facile d'utilisation (interfaçage avec C incroyablement facile).

Mais d'autres langages avec GC le sont (cf *Golang's real-time GC in Theory and Practice*, Will Sewell, Pusher blog, pour une comparaison des performances des GC et leurs optimisations).

Mais, d'autres langages avec GC ont une syntaxe intuitive et très expressive (langages fonctionnels) et le temps gagné en compilation dans Go est en grande partie dû à l'absence de tuning/optimisation fine de la compilation que d'autres langages permettent (voir Haskell ou OCaml).

Merci de votre attention.