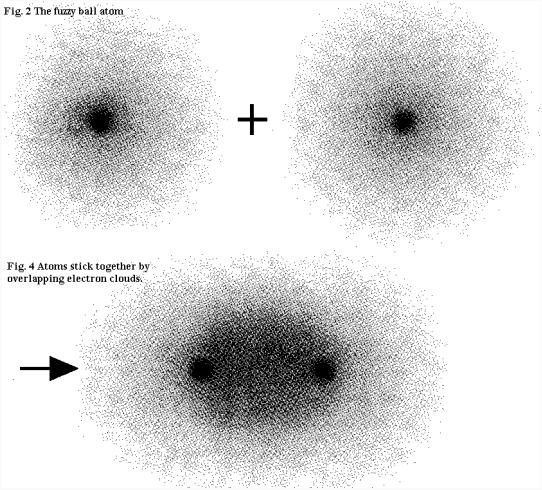
QCaml : Chimie quantique avec OCaml

Anthony Scemama

27/06/2023

Laboratoire de Chimie et Physique Quantiques, UPS/CNRS Toulouse

Problème scientifique



https://www.uq.edu.au/_School_Science_Lessons/TWImagesatoms.html

- Nuage électronique
- Répartition des électrons autour des noyaux dans les molécules

Problème scientifique

Résolution de l'équation de Schrödinger:

$$\mathcal{H}\Psi(\mathrm{R}) = -\frac{1}{2}\nabla^2\Psi(\mathrm{R}) + V(\mathrm{R}) = E_0\Psi(\mathrm{R})$$

Ψ : Fonction d'onde électronique (noyaux fixes)

R : Vecteur de $\mathbb{R}^{3N}$ contenant les positions des électrons

E_0 : Énergie électronique

$-\frac{1}{2}\nabla^2$: Énergie cinétique

V : Énergie potentielle: contient l'attraction électron-noyau, et la répulsion électron-électron et noyau-noyau

C'est une EDP dans un espace à $3N$ dimensions !

Codes de calcul de structure électronique

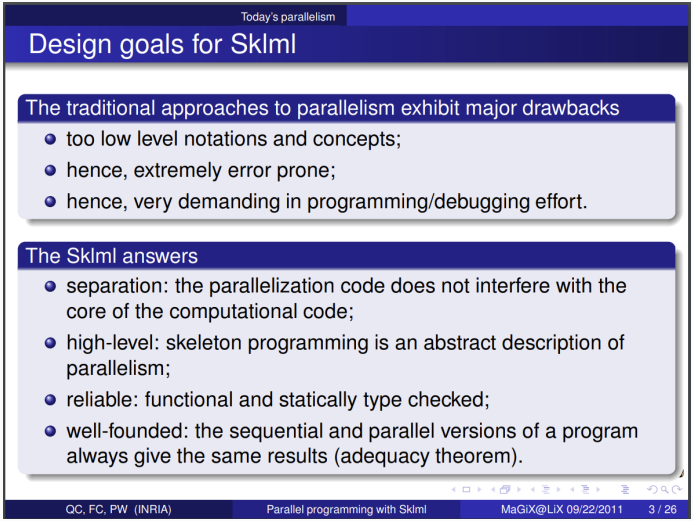
Langages Usuels

- Fortran / C/C++
- Python / Julia

Problématique importante: HPC

- Solutions usuelles pour le calcul parallèle
  - OpenMP (mémoire partagée, multi-thread)
  - MPI (mémoire distribuée)
- Solution originale pour le parallélisme: OCaml+Sklml (2011)
  - Quentin Carbonneaux, François Clément, Pierre Weis
  - `https://who.rocq.inria.fr/Francois.Clement/soft/sklml/doc/pdf/UserManual.pdf`

Sklml (2011)



https://who.rocq.inria.fr/Francois.Clement/soft/sklml/doc/pdf/Talks/MaGiX@LiX/slides/talk.pdf

Sklml (2011)

The Sklml system guaranties that:

- the parallel and sequential programs give the same results;
- hence, if the code runs properly in sequential mode, it is guaranteed to be correct in parallel mode.

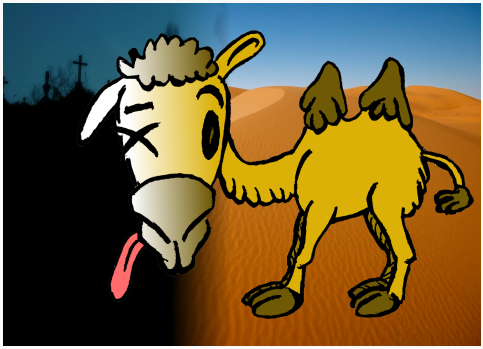
Hence, the methodoly:

1. develop and debug using the sequential semantics;
2. start the heavy parallel computation after changing a flag in the makefile!

https://who.rocq.inria.fr/Francois.Clement/soft/sklml/doc/pdf/Talks/MaGiX@LiX/slides/talk.pdf

Pourquoi QCaml (2019)?

- Bibliothèque pour la chimie quantique
- Écrite avec le paradigme fonctionnel
- Parallélisée avec Sklml
- Reposant sur Lacaml pour l'algèbre linéaire
- Projet exploratoire, orthogonal aux techniques usuelles
- Côté éducatif: Literate Programming avec Org-mode?



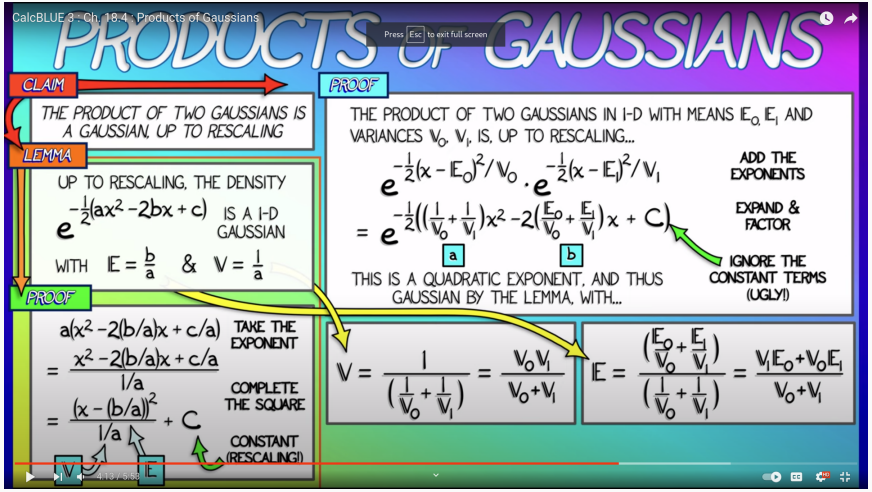
Répulsion électron-électron

$$[pr|qs] = \iint \phi_p(\mathrm{r}_1)\phi_q(\mathrm{r}_2)\frac{1}{|\mathrm{r}_1 - \mathrm{r}_2|}\phi_r(\mathrm{r}_1)\phi_s(\mathrm{r}_2)\mathrm{d}\mathrm{r}_1\mathrm{d}\mathrm{r}_2$$

$$\phi_p(\mathrm{r} - \mathrm{R}_\mathrm{A}) = P_p(x, y, z) \times e^{-\zeta_p|\mathrm{r}-\mathrm{R}_\mathrm{A}|^2}$$

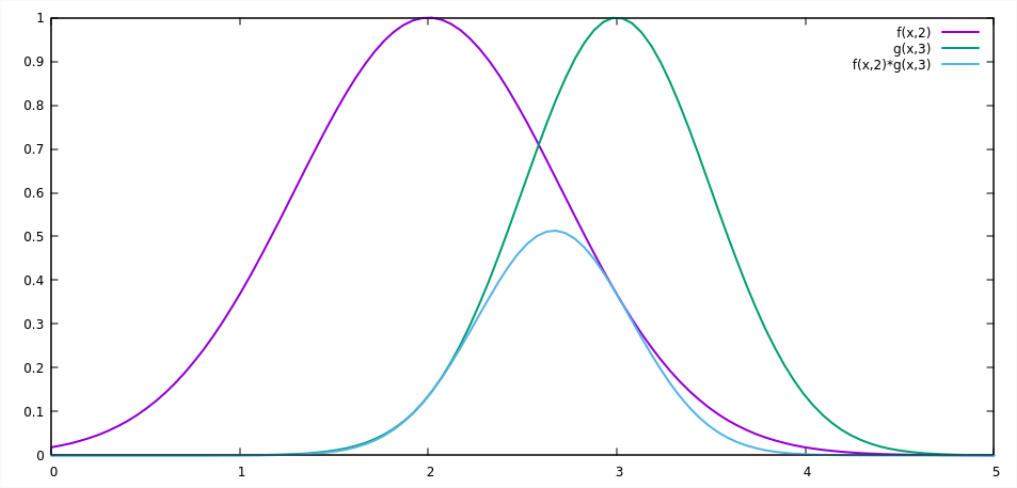
- ϕ: Polynôme $\times$ Gaussienne, centrée sur un atome
- Environ 50 fonctions / atome
- Produits de polynômes: Polynôme d'ordre plus élevé
- *Gaussian Product Theorem*: Le produit de deux Gaussiennes est une Gaussienne

Produits de Gaussiennes



https://youtu.be/lUORBdNccHc

Produits de Gaussiennes



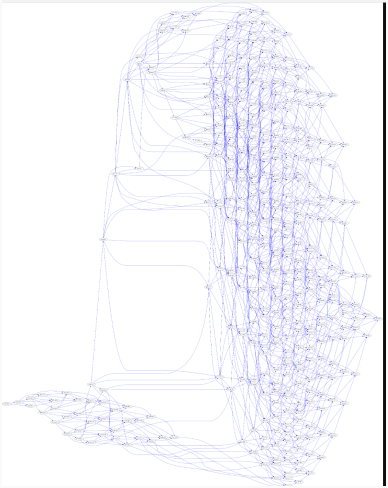
Applications partielles de fonctions

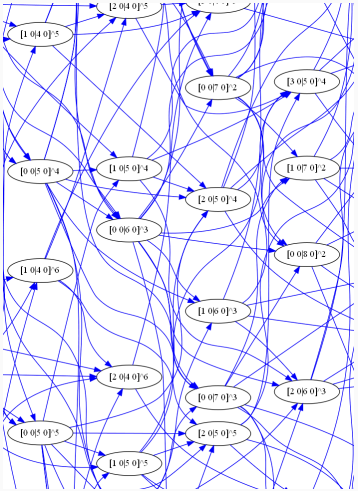
```
let create_make_of p_a' p_b' =
  let a_minus_b    = Co.( Ps.center p_a' |- Ps.center p_b' ) in
  let a_minus_b_sq = Co.dot a_minus_b a_minus_b in
  let norm_scales  = lazy (compute_norm_scales p_a' p_b') in
  let ang_mom      = Am.( Ps.ang_mom p_a' + Ps.ang_mom p_b' ) in
  function p_a ->
    let norm_coef_a = Ps.normalization p_a in
    let alfa_a      = Co.( Ps.exponent p_a |. Ps.center p_a ) in
    function p_b ->
      let exponent = Ps.exponent p_a +. Ps.exponent p_b in let exponent_inv = 1. /. exponent in
      let normalization = (norm_coef_a *. Ps.normalization p_b) *. (pi *. exponent_inv)**1.5 *.
          exp (-. Ps.exponent p_a *. Ps.exponent p_b *. a_minus_b_sq *. exponent_inv) in
      function cutoff -> if abs_float normalization > cutoff then (
            let beta_b         = Co.( Ps.exponent p_b |. Ps.center p_b ) in
            let center         = Co.(exponent_inv |. (alfa_a |+ beta_b)) in
            let center_minus_a = Co.(center |- Ps.center p_a) in
            Some {ang_mom ; exponent ; exponent_inv ; center ; center_minus_a ; a_minus_b ;
              a_minus_b_sq ; normalization ; norm_scales ; shell_a = p_a; shell_b = p_b }
       ) else None
```

Pour la partie polynomiale:

$$
\begin{aligned}
[\mathbf{a}+1_i,\mathbf{b}|\mathbf{cd}]^{(m)} &= PA_i[\mathbf{ab}|\mathbf{cd}]^{(m)} + WP_i[\mathbf{ab}|\mathbf{cd}]^{(m+1)} \\
&+ \frac{a_i}{2\zeta}\left([\mathbf{a}-1_i,\mathbf{b}|\mathbf{cd}]^{(m)} - \frac{\eta}{\zeta+\eta}[\mathbf{a}-1_i,\mathbf{b}|\mathbf{cd}]^{(m+1)}\right) \\
&+ \frac{b_i}{2\zeta}\left([\mathbf{a},\mathbf{b}-1_i|\mathbf{cd}]^{(m)} - \frac{\eta}{\zeta+\eta}[\mathbf{a},\mathbf{b}-1_i|\mathbf{cd}]^{(m+1)}\right) \\
&+ \frac{c_i}{2(\zeta+\eta)}[\mathbf{ab}|\mathbf{c}-1_i,\mathbf{d}]^{(m+1)} + \frac{d_i}{2(\zeta+\eta)}[\mathbf{ab}|\mathbf{c},\mathbf{d}-1_i]^{(m+1)}
\end{aligned}
$$

Algorithmes récursifs





Algorithmes récursifs

- Fonctions Lazy: évite le calcul d'intermédiaires inutiles
- Stockage d'intermédiaires dans des `Hashtbl`: casse la complexité de l'algorithme
- Foncteur permettant de facilement changer d'opérateur:

$$\frac{1}{|r_1 - r_2|} \longrightarrow f(r_1, r_2)$$

  sans besoin de modifier l'algorithme récursif

Parallélisme

- En 2019: OCaml 4.07.1 $\Longrightarrow$ pas de multicore
- Sklml
  - La couche de communications dans Sklml n'est pas suffisamment performante
  - Début de ré-écriture de Sklml utilisant `Stream` et la bibliothèque MPI
  - Prototype qui fonctionne bien, calculs distribués OK
- Limitations:
  - Problèmes avec le bibliothèque MPI (tailles des tableaux en `int32`).
  - Pas de mémoire partagée, donc peu de mémoire par coeur (64 coeurs et 128Go de RAM imposent 2Go max par coeur)
  - Tentatives utilisant `Unix.map_file` pour partager des gros tableaux, mais trop de bricolage...
- Choix plus raisonnable: Attendre que la branche multicore soit fusionnée.

`f (p,q)` : Calcule toutes les intégrales $[pq|rs]$ et les stocke dans un `Bigarray`.

Avant

```
Array.iter f shell_pairs;
```

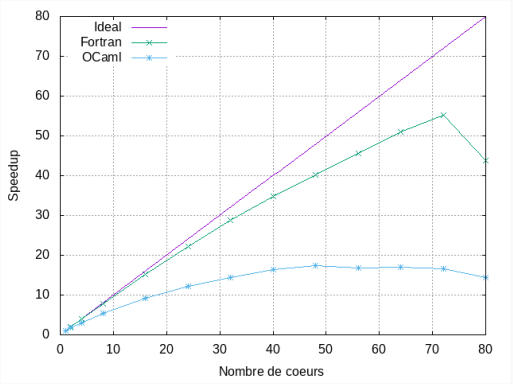
Après

```
let _ =
  let pool = Domainslib.Task.setup_pool ~num_domains:Qcaml.num_domains () in
  Domainslib.Task.run pool (fun _ ->
      shell_pairs
      |> Array.map (fun sp -> Domainslib.Task.async pool (fun _ -> f sp) )
      |> Array.iter (fun task ->
             ignore (Domainslib.Task.await pool task))
    )
    in
    Domainslib.Task.teardown_pool pool;
```

Comparaison OCaml 5.0 / Fortran

Threads	OCaml (s)	Fortran (s)
1	114.14	260.88
2	65.33	130.95
4	36.89	65.91
8	21.15	33.34
16	12.51	17.10
24	9.29	11.73
32	7.93	9.06
40	6.94	7.50
48	6.56	6.48
56	6.80	5.72
64	6.74	5.11
72	6.89	4.73
80	7.98	5.96

- Fortran: algorithme différent (plus simple à écrire)
- Turpan (CALMIP): ARM Ampere 80 cores



Retour sur l'utilisation d'OCaml comme langage

Avantages

- Certains algorithmes sont plus faciles à implémenter: gain en efficacité
- Le paradigme fonctionnel incite à écrire des fonctions pures
- Le parallélisme avec OCaml 5.0 est très simple avec des fonctions pures
- Très peu de bugs: grande confiance dans le code.
- Grande ré-utilisabilité du code
- Portabilité: Très facile à installer sur tout type de machine (x86/ARM, Linux/Mac) si on sait utiliser OPAM.

Retour sur l'utilisation d'OCaml comme langage

Inconvénients

- Les contributeurs extérieurs ont peur de modifier le code car ils ne connaissent pas le langage
- OCaml n'est en général pas installé sur les supercalculateurs
- Interface vers la bibliothèque MPI pas suffisamment robuste
- Efficacité du parallélisme décevante
- Le profiler montre beaucoup de temps passé dans les allocations, et il est parfois difficile de s'en sortir

Utilisation de notebooks

Jupyter ou Emacs org-mode

- Équations, images, structuration du document
- Prototypage rapide, comme avec `utop`
- Plots de fonctions
- Export de fichier `.ml` qui peut ensuite être compilé en natif

org-mode

- Mélange de plusieurs langages possible pour le post-traitement
- S'interface bien avec Git
- Emacs keybindings, mais on peut utiliser le `Evil mode`

Utilisation d'org-mode pour le code source

Avantage

- `.mli`, `.ml` et tests côte-à-côte dans le même fichier
- Équations et images dans les commentaires
- Génération de la doc en HTML avec le code

Inconvénients

- Dépendance à Emacs
- Makefile compliqué
- Temps de compilation trop long (génération des fichiers)
- Difficulté pour les développeurs extérieurs au projet

Conclusion

- Plutôt une fausse bonne idée...