CHIMIE QUANTIQUE ET PARALLÉLISME MASSIF

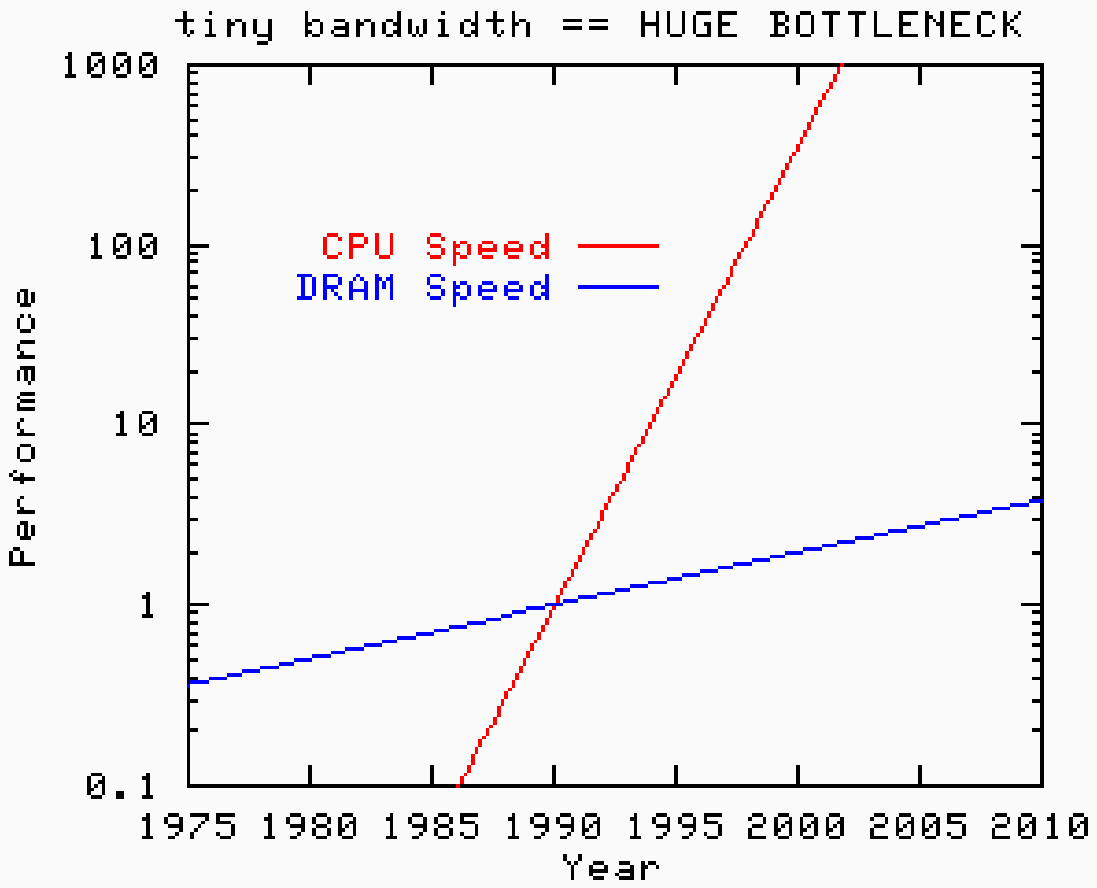
Anthony Scemama

6/11/2015

Lab. Chimie et Physique Quantiques, IRSAMC, UPS/CNRS, Toulouse

PROBLÈMES LIÉS AUX MACHINES MODERNES

LE “MUR” DE LA MÉMOIRE



- CPU : $\times 1.55$ / an
- Mémoire : $\times 1.10$ / an

Stream benchmark : http://www.cs.virginia.edu/stream

Le “mur” de la mémoire

Latence Temps de transfert d’une seule donnée entre deux points

Bande passante Quantité de données qui passent par un point par unité de temps

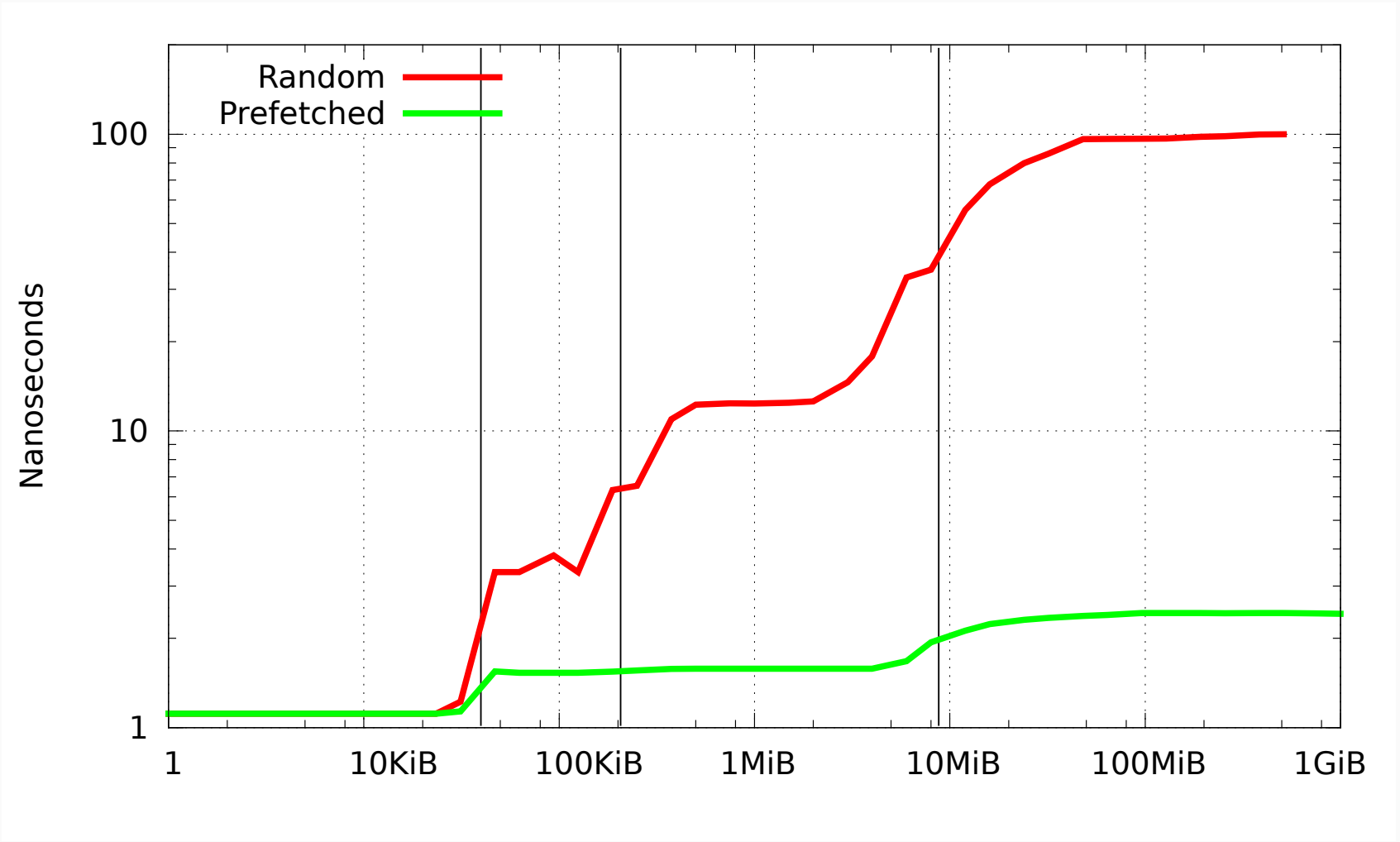
Améliorations de la latence et de la bande passante sur 20 ans:

	Latence	Bande passante
Disque	8×	143×
RAM	4×	120×
Ethernet	16×	1000×
CPU	21×	2250×

- Mémoires hierarchiques (caches) : masquent les latences
- Les accès aléatoires sont de plus en plus coûteux

LATENCE (NANOSECONDES)

Accès dans un tableau de taille croissante:



Latence (nanosecondes)

Intel(R) Xeon(R) CPU E3-1271 v3 @ 3.60GHz

1 cycle = 0.28 ns, peak SP throughput = 0.0087 ns/flop

Integer	ADD	MUL	DIV	MOD	Bit
32 bit	0.28	0.84	6.60	7.07	0.28
64 bit	0.28	0.84	11.80	11.75	0.28
Floating Point	ADD	MUL	DIV		
32 bit	0.84	1.39	3.77		
64 bit	0.84	1.39	5.71		
Data read	Random	Prefetched			
L1	1.11	1.11			
L2	3.3	1.54			
L3	12.3	1.58			
RAM	100.	2.4			

`http://lmbench.sourceforge.net`

LATENCE (CYCLES)

Intel(R) Xeon(R) CPU E3-1271 v3 @ 3.60GHz

1 cycle = 0.28 ns, peak SP throughput = 32 flops/cycle

Integer	ADD	MUL	DIV	MOD	Bit
32 bit	1	3	23	25	1
64 bit	1	3	42	42	1
Floating Point	**ADD**	**MUL**	**DIV**		
32 bit	3	5	13		
64 bit	3	5	20		
Data read	**Random**	**Prefetched**			
L1	4	4			
L2	12	5.5			
L3	44	5.6			
RAM	357	8.6			

`http://lmbench.sourceforge.net`

- Accès aléatoire à la mémoire très coûteux : 357 cycles ⇔ 11 424 flops SP
- Accès régulier à la mémoire : déclenche les prefetchers hardware pour masquer la latence

D'autres nombres

Mutex lock/unlock	100 ns
Infiniband (RDMA)	1 200 ns
Ethernet (TCP/IP)	50 000 ns
Disk seek (SSD)	50 000 ns
Disk seek (15k rpm)	2 000 000 ns

IMPORTANCE DES ACCÈS AUX DONNÉES

Matrice des distances

```
do j=1,n
  do i=1,j
    dist(i,j) = X(i,1)*X(j,1) + X(i,2)*X(j,2) + X(i,3)*X(j,3)
  end do
end do

do j=1,n
  do i=j+1,n
    dist(i,j) = dist(j,i)
  end do
end do
```

- `n`=133 : 6.25 μs, 7.1 GFlops/s
- `n`=4125 : 91.45 ms, 0.93 GFlops/s
- `dist(j,i)` : Accès distant

Matrice des distances

```
do j=1,n
  do i=1,n ! <- 2 fois plus de flops !
    dist(i,j) = X(i,1)*X(j,1) + X(i,2)*X(j,2) &
              + X(i,3)*X(j,3)
  end do
end do
```

- `n`=133 : 4.32 μs, 20.47 GFlops/s : $\times$1.8
- `n`=4125 : 14.1 ms, 12.07 GFlops/s : $\times$6.5
- `X(j,1:3)` : constants dans la boucle interne
- `X` : tient dans le cache L2, accès prévisible

BANDE PASSANTE

Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz):

`a(i) = a(i) + b(i)*c(i)`

- 2 FMA vectoriels par cycle : `a = a + b*c`
  vecteurs de 4 flottants en DP (4×8 octets)

BANDE PASSANTE

Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz):

`a(i) = a(i) + b(i)*c(i)`

- 2 FMA vectoriels par cycle : `a = a + b*c`
  vecteurs de 4 flottants en DP (4×8 octets)
- 16 octets par flop

BANDE PASSANTE

Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz):

`a(i) = a(i) + b(i)*c(i)`

- 2 FMA vectoriels par cycle : `a = a + b*c`
  vecteurs de 4 flottants en DP (4×8 octets)
- 16 octets par flop
- Puissance crête
  2.6 GHz $\times$ ($2 \times 2 \times 4$ flops) $\times$ 12 coeurs = 499.2 GFlops/s (DP)

BANDE PASSANTE

Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz):

`a(i) = a(i) + b(i)*c(i)`

- 2 FMA vectoriels par cycle : `a = a + b*c`
  vecteurs de 4 flottants en DP (4×8 octets)
- 16 octets par flop
- Puissance crête
  2.6 GHz $\times$ ($2 \times 2 \times 4$ flops) $\times$ 12 coeurs = 499.2 GFlops/s (DP)
- Bande passante nécessaire :
  499.2 Gflops/s $\times$ 16 o/flops = 8 TiB/s

BANDE PASSANTE

Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz):

`a(i) = a(i) + b(i)*c(i)`

- 2 FMA vectoriels par cycle : `a = a + b*c`
  vecteurs de 4 flottants en DP (4×8 octets)
- 16 octets par flop
- Puissance crête
  2.6 GHz $\times$ ($2 \times 2 \times 4$ flops) $\times$ 12 coeurs = 499.2 GFlops/s (DP)
- Bande passante nécessaire :
  499.2 Gflops/s $\times$ 16 o/flops = 8 TiB/s
- Bande passante mémoire max
  4 canaux $\times$ 2133 MHz $\times$ 8 octets = 68.2 GiB/s

BANDE PASSANTE

Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz):

`a(i) = a(i) + b(i)*c(i)`

- 2 FMA vectoriels par cycle : `a = a + b*c`
  vecteurs de 4 flottants en DP (4×8 octets)
- 16 octets par flop
- Puissance crête
  2.6 GHz $\times$ ($2 \times 2 \times 4$ flops) $\times$ 12 coeurs = 499.2 GFlops/s (DP)
- Bande passante nécessaire :
  499.2 Gflops/s $\times$ 16 o/flops = 8 TiB/s
- Bande passante mémoire max
  4 canaux $\times$ 2133 MHz $\times$ 8 octets = 68.2 GiB/s

La bande passante mémoire est 117x trop faible !

1. La réduction du nombre de flops d'un algorithme ne conduit pas forcément à l'accélération d'un programme.
   Ex: Algorithmes "linear scaling" sont linéaires en stockage et en calcul, donc probablement memory-bound (pré-facteur 117×, *etc*).

CONCLUSIONS

1. La réduction du nombre de flops d'un algorithme ne conduit pas forcément à l'accélération d'un programme.
   Ex: Algorithmes "linear scaling" sont linéaires en stockage et en calcul, donc probablement memory-bound (pré-facteur 117×, *etc*).
2. La performance crête d'une machine n'est pas du tout représentative du temps de restitution des applications scientifiques : elle est de plus en plus difficile à atteindre.

Pour avoir un programme efficace il faut:

- Réutiliser au maximum ce qui peut être dans les caches
- Minimiser les accès aléatoires à la RAM : avoir des accès réguliers
- Minimiser les accès au réseau (communications bloquantes)
- Oublier les I/O sur disque
- Utiliser des bibliothèques optimisées dès que possible (MKL, PETSc, MUMPS, *etc*)

Machine exaflopique (10^{18} flops/s) prévue en 2022

- Unité de mesure : flops/s $\rightarrow$ augmentation de la puissance crête

Machine exaflopique (10^{18} flops/s) prévue en 2022

- Unité de mesure : flops/s $\rightarrow$ augmentation de la puissance crête
- Encore plus de coeurs
- Un plus de RAM, donc moins de RAM par coeur

Machine exaflopique (10^{18} flops/s) prévue en 2022

- Unité de mesure : flops/s $\rightarrow$ augmentation de la puissance crête
- Encore plus de coeurs
- Un plus de RAM, donc moins de RAM par coeur
- La latence du réseau ne va pas beaucoup réduire
- La fréquence des CPUs va probablement baisser pour réduire la consommation
- Généralisation des accélérateurs (GPU, Xeon Phi) : hétérogénéité

Machine exaflopique (10^{18} flops/s) prévue en 2022

- Unité de mesure : flops/s $\rightarrow$ augmentation de la puissance crête
- Encore plus de coeurs
- Un plus de RAM, donc moins de RAM par coeur
- La latence du réseau ne va pas beaucoup réduire
- La fréquence des CPUs va probablement baisser pour réduire la consommation
- Généralisation des accélérateurs (GPU, Xeon Phi) : hétérogénéité

Ça s'annonce mal si on ne fait rien...

CHIMIE QUANTIQUE (MOLÉCULAIRE)

1. Méthodes auto-cohérentes
2. Méthodes post-Hartree-Fock perturbatives
3. Méthodes post-Hartree-Fock variationnelles
4. Méthodes post-Hartree-Fock stochastiques

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$
- Diagonalisations successives de matrices construites à partir d'intégrales atomiques (Equations de Roothaan)

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$
- Diagonalisations successives de matrices construites à partir d'intégrales atomiques (Equations de Roothaan)
- Réplication des intégrales : parallélisation distribuée facile (ScaLapack)

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$
- Diagonalisations successives de matrices construites à partir d'intégrales atomiques (Equations de Roothaan)
- Réplication des intégrales : parallélisation distribuée facile (ScaLapack)
- Si les intégrales ne tiennent pas en mémoire : direct SCF

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$
- Diagonalisations successives de matrices construites à partir d'intégrales atomiques (Equations de Roothaan)
- Réplication des intégrales : parallélisation distribuée facile (ScaLapack)
- Si les intégrales ne tiennent pas en mémoire : direct SCF
- Orbitales atomiques sont locales $\rightarrow$ Intégrales atomiques très creuses

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$
- Diagonalisations successives de matrices construites à partir d'intégrales atomiques (Equations de Roothaan)
- Réplication des intégrales : parallélisation distribuée facile (ScaLapack)
- Si les intégrales ne tiennent pas en mémoire : direct SCF
- Orbitales atomiques sont locales $\rightarrow$ Intégrales atomiques très creuses
- Schémas creux : résolution par des produits de matrices creuses (gradient conjugué)

Optimisation des orbitales moléculaires (SCF) : HF, Semi-empirique, DFT, SCC-DFTB

- Optimisation non-linéaire : $\mathcal{H}[\mathbf{C}].\mathbf{C} = E.\mathbf{S}.\mathbf{C}$
- Diagonalisations successives de matrices construites à partir d'intégrales atomiques (Equations de Roothaan)
- Réplication des intégrales : parallélisation distribuée facile (ScaLapack)
- Si les intégrales ne tiennent pas en mémoire : direct SCF
- Orbitales atomiques sont locales $\rightarrow$ Intégrales atomiques très creuses
- Schémas creux : résolution par des produits de matrices creuses (gradient conjugué)
- Schémas itératifs (SCF), donc barrières de synchronisation

Complexité

- Hartree-Fock : $\mathcal{O}(N^4)$. Calcul des intégrales $\langle pq|rs\rangle$ en base d'AOs.
- LDA, DFTB, semi-empirique : $\mathcal{O}(N^3)$ lié à la diagonalisation
- Les AOs sont très locales (car non-orthogonales) : Schémas approchés en $\mathcal{O}(N)$ efficaces
- L'aspect iteratif nuit au parallélisme : besoin de réseaux à faible latence

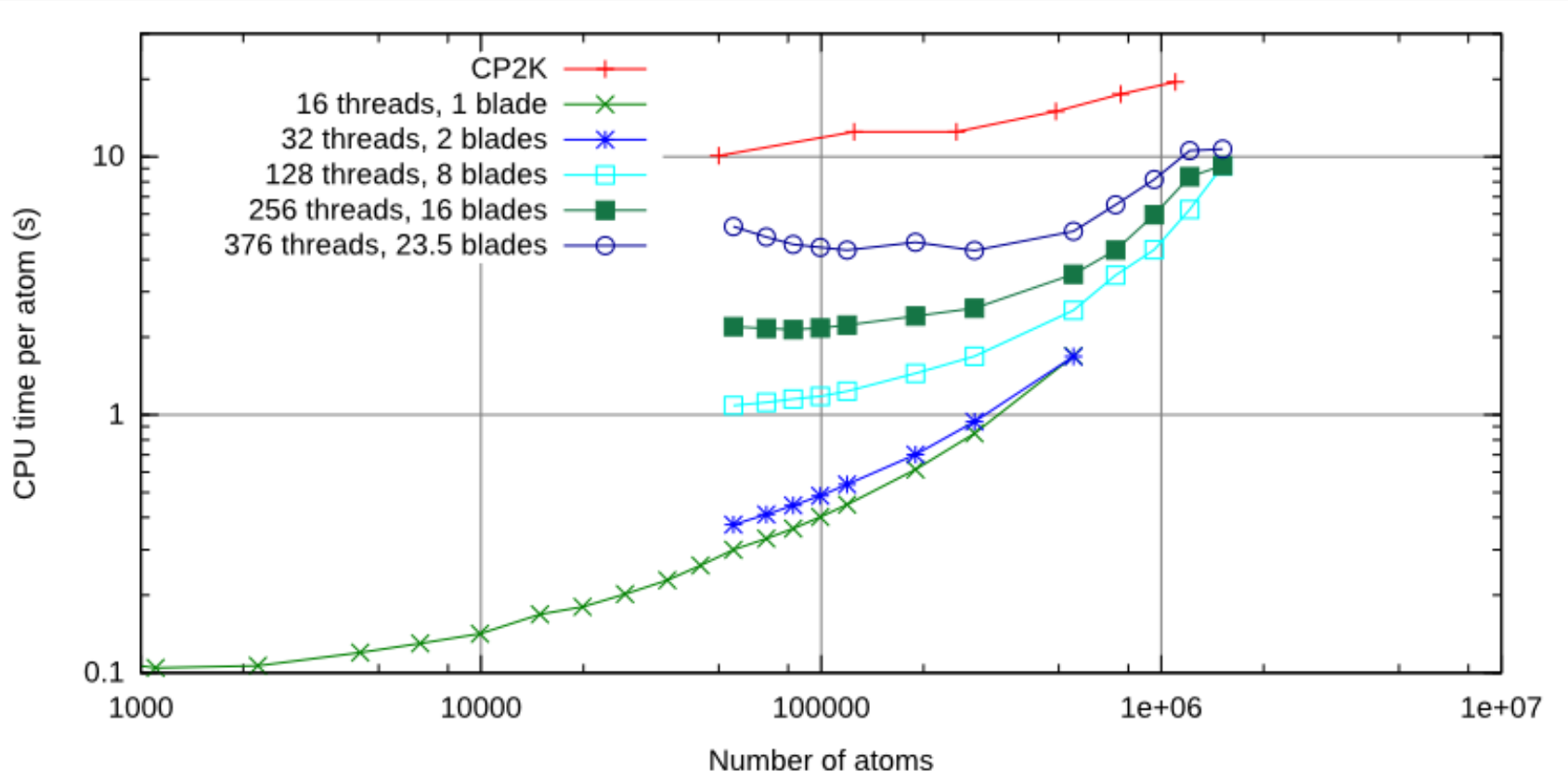
Exemple : SCC-DFTB[1]

deMon-Nano : Pas de $\mathcal{O}(N)$, mais plutôt un $\mathcal{O}(N^2)$

- $\mathcal{O}(N)$ en stockage et $\mathcal{O}(N^2)$ en calcul
- Réutilisation des données : meilleur efficacité des cycles CPU
- Jusqu'à 1 500 000 atomes, $\mathcal{O}(N^2)$ plus rapide et moins approché que $\mathcal{O}(N)$

[1] A. S., N. Renon and M. Rapacioli, *JCTC*, **10:6**, 2344–2354 (2014)

1. MÉTHODES AUTO-COHÉRENTES



$$E = E_0 - \sum_{ijab} c_{ijab} \frac{\langle ij|ab\rangle^2}{\Delta E}$$

- Approches en base de MOs : Transformation à 4 indices partielle $oo \rightarrow vv \Rightarrow \mathcal{O}(N^5)$

$$E = E_0 - \sum_{ijab} c_{ijab} \frac{\langle ij|ab\rangle^2}{\Delta E}$$

- Approches en base de MOs : Transformation à 4 indices partielle $oo \rightarrow vv \Rightarrow \mathcal{O}(N^5)$
- Approches en base d'AOs :
  - Complet : Beaucoup de communications entre les nœuds
  - Local : Schémas approchés en $\mathcal{O}(N)$, integral-direct

$$E = E_0 - \sum_{ijab} c_{ijab} \frac{\langle ij|ab\rangle^2}{\Delta E}$$

- Approches en base de MOs : Transformation à 4 indices partielle $oo \rightarrow vv \Rightarrow \mathcal{O}(N^5)$
- Approches en base d'AOs :
  - Complet : Beaucoup de communications entre les nœuds
  - Local : Schémas approchés en $\mathcal{O}(N)$, integral-direct
- Schéma non-itératif : pas de barrière de synchronisation

$$E = E_0 - \sum_{ijab} c_{ijab} \frac{\langle ij|ab\rangle^2}{\Delta E}$$

- Approches en base de MOs : Transformation à 4 indices partielle $oo \rightarrow vv \Rightarrow \mathcal{O}(N^5)$
- Approches en base d'AOs :
  - Complet : Beaucoup de communications entre les nœuds
  - Local : Schémas approchés en $\mathcal{O}(N)$, integral-direct
- Schéma non-itératif : pas de barrière de synchronisation
- Pas de principe variationnel : sensibilité aux erreurs numériques

CI,CC,CAS,MR-CI,MR-CC,FCI, *etc*

- Transformation à 4 indices coûteuse et difficilement parallélisable (communications inter-nœuds importantes)

CI,CC,CAS,MR-CI,MR-CC,FCI, *etc*

- Transformation à 4 indices coûteuse et difficilement parallélisable (communications inter-nœuds importantes)
- Le stockage des intégrales pose problème

CI,CC,CAS,MR-CI,MR-CC,FCI, *etc*

- Transformation à 4 indices coûteuse et difficilement parallélisable (communications inter-nœuds importantes)
- Le stockage des intégrales pose problème
- Nécessitent (sauf CAS) la transformation à 4 indices complète ($oo \rightarrow vv$ **et** $vv \rightarrow vv$)

CI,CC,CAS,MR-CI,MR-CC,FCI, *etc*

- Transformation à 4 indices coûteuse et difficilement parallélisable (communications inter-nœuds importantes)
- Le stockage des intégrales pose problème
- Nécessitent (sauf CAS) la transformation à 4 indices complète ($oo \rightarrow vv$ **et** $vv \rightarrow vv$)
- Schémas itératifs (Davidson), donc beaucoup de barrières de synchronisation

CI,CC,CAS,MR-CI,MR-CC,FCI, *etc*

- Transformation à 4 indices coûteuse et difficilement parallélisable (communications inter-nœuds importantes)
- Le stockage des intégrales pose problème
- Nécessitent (sauf CAS) la transformation à 4 indices complète ($oo \rightarrow vv$ **et** $vv \rightarrow vv$)
- Schémas itératifs (Davidson), donc beaucoup de barrières de synchronisation

Méthodes très mal adaptées aux machines actuelles

TRANSFORMATION À 4 INDICES

$(pq\|rs)$	:	AOs, donc toujours très creux $\mathcal{O}(N^2)$
$(iq\|rs) = \sum_p c_p^i (pq\|rs)$	:	un peu creux $\mathcal{O}(N^3)$
$(ij\|rs) = \sum_q c_q^j (iq\|rs)$	:	plein $\mathcal{O}(N^4)$
$(ij\|ks) = \sum_r c_r^k (ij\|rs)$	:	plein, ou un peu creux si MO localisées
$(ij\|kl) = \sum_s c_s^l (ij\|ks)$	:	plein, ou assez creux si MO localisées

TRANSFORMATION À 4 INDICES

$(pq|rs)$: AOs, donc toujours très creux $\mathcal{O}(N^2)$

$(iq|rs) = \sum_p c_p^i (pq|rs)$: un peu creux $\mathcal{O}(N^3)$

$(ij|rs) = \sum_q c_q^j (iq|rs)$: plein $\mathcal{O}(N^4)$

$(ij|ks) = \sum_r c_r^k (ij|rs)$: plein, ou un peu creux si MO localisées

$(ij|kl) = \sum_s c_s^l (ij|ks)$: plein, ou assez creux si MO localisées

- $\mathcal{O}(N^4)$ données et $\mathcal{O}(N^5)$ flops

TRANSFORMATION À 4 INDICES

$$
\begin{aligned}
(pq|rs) &: \text{AOs, donc toujours très creux } \mathcal{O}(N^2) \\
(iq|rs) = \textstyle\sum_p c_p^i (pq|rs) &: \text{un peu creux } \mathcal{O}(N^3) \\
(ij|rs) = \textstyle\sum_q c_q^j (iq|rs) &: \text{plein } \mathcal{O}(N^4) \\
(ij|ks) = \textstyle\sum_r c_r^k (ij|rs) &: \text{plein, ou un peu creux si MO localisées} \\
(ij|kl) = \textstyle\sum_s c_s^l (ij|ks) &: \text{plein, ou assez creux si MO localisées}
\end{aligned}
$$

- $\mathcal{O}(N^4)$ données et $\mathcal{O}(N^5)$ flops
- Chaque CPU a besoin de tout à un moment :
  communications all-to-all (épouvantable pour le HPC)

TRANSFORMATION À 4 INDICES

$$
\begin{aligned}
(pq|rs) &: \text{AOs, donc toujours très creux } \mathcal{O}(N^2) \\
(iq|rs) = \textstyle\sum_p c_p^i (pq|rs) &: \text{un peu creux } \mathcal{O}(N^3) \\
(ij|rs) = \textstyle\sum_q c_q^j (iq|rs) &: \text{plein } \mathcal{O}(N^4) \\
(ij|ks) = \textstyle\sum_r c_r^k (ij|rs) &: \text{plein, ou un peu creux si MO localisées} \\
(ij|kl) = \textstyle\sum_s c_s^l (ij|ks) &: \text{plein, ou assez creux si MO localisées}
\end{aligned}
$$

- $\mathcal{O}(N^4)$ données et $\mathcal{O}(N^5)$ flops
- Chaque CPU a besoin de tout à un moment : communications all-to-all (épouvantable pour le HPC)
- Fonctionne très bien à mémoire partagée, mal en distribué

TRANSFORMATION À 4 INDICES

$$
\begin{aligned}
(pq|rs) &: \text{AOs, donc toujours très creux } \mathcal{O}(N^2) \\
(iq|rs) = \textstyle\sum_p c_p^i (pq|rs) &: \text{un peu creux } \mathcal{O}(N^3) \\
(ij|rs) = \textstyle\sum_q c_q^j (iq|rs) &: \text{plein } \mathcal{O}(N^4) \\
(ij|ks) = \textstyle\sum_r c_r^k (ij|rs) &: \text{plein, ou un peu creux si MO localisées} \\
(ij|kl) = \textstyle\sum_s c_s^l (ij|ks) &: \text{plein, ou assez creux si MO localisées}
\end{aligned}
$$

- $\mathcal{O}(N^4)$ données et $\mathcal{O}(N^5)$ flops
- Chaque CPU a besoin de tout à un moment : communications all-to-all (épouvantable pour le HPC)
- Fonctionne très bien à mémoire partagée, mal en distribué

Schémas approchés pour réduire le stockage :

- Résolution de l'identité avec une base auxiliaire
- Décomposition de Cholesky

Stockage des intégrales biélectroniques

- On ne stocke que les intégrales $> \epsilon$
- Adressage $i \rightarrow (pq|rs)$ (symétries de permutation des indices)

Stockage des intégrales biélectroniques

- On ne stocke que les intégrales $> \epsilon$
- Adressage $i \rightarrow (pq|rs)$ (symétries de permutation des indices)

Approches naturelles

1. Table de hachage : Insertion en $\mathcal{O}(1)$, Recherche en $\mathcal{O}(1)$
2. Arbre binaire de recherche : Insertion en $\mathcal{O}(\log(N))$, Recherche en $\mathcal{O}(\log(N))$

Stockage des intégrales biélectroniques

- On ne stocke que les intégrales $> \epsilon$
- Adressage $i \to (pq|rs)$ (symétries de permutation des indices)

Approches naturelles

1. Table de hachage : Insertion en $\mathcal{O}(1)$, Recherche en $\mathcal{O}(1)$
2. Arbre binaire de recherche : Insertion en $\mathcal{O}(\log(N))$, Recherche en $\mathcal{O}(\log(N))$

Mais

- Temps constant mais long : pas de prefetch possible, latences gigantesques. Rehash nécessaire.
- Insertions difficiles en parallèle

Accès aux intégrales biélectroniques

Notre implémentation dans le Quantum Package[2]: Tableau de listes

- $(pq|rs) \rightarrow i$: i sur 64 bits

[2] `https://github.com/LCPQ/quantum_package`

Accès aux intégrales biélectroniques

Notre implémentation dans le Quantum Package[2]: Tableau de listes

- $(pq|rs) \rightarrow i$: i sur 64 bits
- 49 bits de poids fort : Accès direct au premier élément d'une liste (tableau de pointeurs)

[2] `https://github.com/LCPQ/quantum_package`

Accès aux intégrales biélectroniques

Notre implémentation dans le Quantum Package[2]: Tableau de listes

- $(pq|rs) \rightarrow i$: i sur 64 bits
- 49 bits de poids fort : Accès direct au premier élément d'une liste (tableau de pointeurs)
- 15 bits de poids faible : Recherche par dichotomie de i dans $< 2^{15} = 32768$ valeurs possibles

[2] `https://github.com/LCPQ/quantum_package`

ACCÈS AUX INTÉGRALES BIÉLECTRONIQUES

Avantages

- Accès direct au 1er élément de la list (Random access), puis

Avantages

- Accès direct au 1er élément de la list (Random access), puis
- Recherche par dichotomie dans le cache L1 (très rapide : faible latence)

Avantages

- Accès direct au 1er élément de la list (Random access), puis
- Recherche par dichotomie dans le cache L1 (très rapide : faible latence)
- Dichotomie : Tri en $\mathcal{O}(N)$ par radix sort

Avantages

- Accès direct au 1er élément de la list (Random access), puis
- Recherche par dichotomie dans le cache L1 (très rapide : faible latence)
- Dichotomie : Tri en $\mathcal{O}(N)$ par radix sort
- Tri des listes en parallèle

Avantages

- Accès direct au 1er élément de la list (Random access), puis
- Recherche par dichotomie dans le cache L1 (très rapide : faible latence)
- Dichotomie : Tri en $\mathcal{O}(N)$ par radix sort
- Tri des listes en parallèle
- Insertion/Écriture possible en parallèle dans 2 listes distinctes

Avantages

- Accès direct au 1er élément de la list (Random access), puis
- Recherche par dichotomie dans le cache L1 (très rapide : faible latence)
- Dichotomie : Tri en $\mathcal{O}(N)$ par radix sort
- Tri des listes en parallèle
- Insertion/Écriture possible en parallèle dans 2 listes distinctes
- Les éléments consécutifs sont proches physiquement : utilisation efficace des caches

Avantages

- Accès direct au 1er élément de la list (Random access), puis
- Recherche par dichotomie dans le cache L1 (très rapide : faible latence)
- Dichotomie : Tri en $\mathcal{O}(N)$ par radix sort
- Tri des listes en parallèle
- Insertion/Écriture possible en parallèle dans 2 listes distinctes
- Les éléments consécutifs sont proches physiquement : utilisation efficace des caches
- Distribution de la structure de données sur plusieurs nœuds triviale

INTERACTION DE CONFIGURATIONS

RÈGLES DE SLATER-CONDON

Si 2 déterminants diffèrent par une spin-orbitale:

$$\begin{aligned}\langle D|\mathcal{O}_1|D_i^j\rangle &= \langle i|\mathcal{O}_1|j\rangle \\ \langle D|\mathcal{O}_2|D_i^j\rangle &= \sum_{k\in D}\langle ik|\mathcal{O}_2|jk\rangle - \langle ik|\mathcal{O}_2|kj\rangle\end{aligned}$$

Si 2 déterminants diffèrent par 2 spin-orbitales:

$$\begin{aligned}\langle D|\mathcal{O}_1|D_{ik}^{jl}\rangle &= 0 \\ \langle D|\mathcal{O}_2|D_{ik}^{jl}\rangle &= \langle ik|\mathcal{O}_2|jl\rangle - \langle ik|\mathcal{O}_2|lj\rangle\end{aligned}$$

Règles de Slater-Condon

- Transforment des intégrales à $3N$ dimensions en intégrales à 3 ou 6 dimensions
- Condition nécessaire : Orbitales orthogonales
- Dans la base des déterminants, $\mathcal{H}$ est très creux et symétrique

$\rightarrow$ Chaque élément extra-diagonal de $\mathcal{H}$ est soit 0, soit une somme d'intégrales mono- et bi-électroniques

Calcul de l'énergie CI

- Algorithme naïf : Construction de la matrice $\mathcal{H}$ et diagonalisation

Calcul de l'énergie CI

- Algorithme naïf : Construction de la matrice $\mathcal{H}$ et diagonalisation
- Algorithme moins naïf : construction de la matrice $\mathcal{H}$ en format creux et diagonalisation

Calcul de l'énergie CI

- Algorithme naïf : Construction de la matrice $\mathcal{H}$ et diagonalisation
- Algorithme moins naïf : construction de la matrice $\mathcal{H}$ en format creux et diagonalisation
- CISD : nb de déterminants $\sim$ nb d'intégrales. On ne pourra pas stocker $\mathcal{H}$ en RAM, même en format creux.

Calcul de l'énergie CI

- Algorithme naïf : Construction de la matrice $\mathcal{H}$ et diagonalisation
- Algorithme moins naïf : construction de la matrice $\mathcal{H}$ en format creux et diagonalisation
- CISD : nb de déterminants $\sim$ nb d'intégrales. On ne pourra pas stocker $\mathcal{H}$ en RAM, même en format creux.
- $\mathcal{H}$ est diagonal-dominant $\rightarrow$ diagonalisation itérative de Davidson.

Calcul de l'énergie CI

- Algorithme naïf : Construction de la matrice $\mathcal{H}$ et diagonalisation
- Algorithme moins naïf : construction de la matrice $\mathcal{H}$ en format creux et diagonalisation
- CISD : nb de déterminants $\sim$ nb d'intégrales. On ne pourra pas stocker $\mathcal{H}$ en RAM, même en format creux.
- $\mathcal{H}$ est diagonal-dominant $\rightarrow$ diagonalisation itérative de Davidson.
- au-delà de CISD, le nb de déterminants est plus grand que le nb d'intégrales. On ne pourra pas non plus stocker les vecteurs.

CALCUL DE L'ÉNERGIE CI

- Algorithme naïf : Construction de la matrice $\mathcal{H}$ et diagonalisation
- Algorithme moins naïf : construction de la matrice $\mathcal{H}$ en format creux et diagonalisation
- CISD : nb de déterminants $\sim$ nb d'intégrales. On ne pourra pas stocker $\mathcal{H}$ en RAM, même en format creux.
- $\mathcal{H}$ est diagonal-dominant $\rightarrow$ diagonalisation itérative de Davidson.
- au-delà de CISD, le nb de déterminants est plus grand que le nb d'intégrales. On ne pourra pas non plus stocker les vecteurs.
- Donc, écriture des vecteurs sur disque : cauchemar pour le HPC.

```
do j=1,N_determinants
  do i=1,j
    H(i,j) = SlaterRules( D(i), 'H', D(j) )
  end do
end do
```

- Algorithme naturel
- Lecture en continu des déterminants
- Écriture en continu de `H`
- Lectures aléatoires des intégrales
- Très creux, donc `SlaterRules( D(i), 'H', D(j) ) = 0` presque toujours
- Inefficace tel quel

```
do n=1,N_integrals
  call get_H_indices(ijkl(:,n), pq, npq)
  do m=1,npq
    p = pq(1,m)
    q = pq(2,m)
    H(p,q) = H(p,q) + c(p,q) * integral_value(n)
  end do
end do
```

- On saute implicitement tous les $\langle i|\mathcal{H}|j\rangle = 0$
- Lecture/Écriture aléatoire dans `H`
- Nécessite une fonction d'adressage `get_H_indices`

- On saute implicitement tous les $\langle i|\mathcal{H}|j\rangle = 0$
- Lecture/Écriture aléatoire dans `H`
- Nécessite une fonction d'adressage `get_H_indices`

Conséquences

- Espaces de déterminants complets et sélectionnés *a priori* : CISD, CISDT, CISDTQ, CAS, CAS+SD, Full-CI
- Difficulté : croissance de ces espaces avec la taille du problème

Conclusion sur les méthodes post-Hartree-Fock

- Communications all-to-all à cause de la transformation à 4 indices
- Gros besoins en RAM (intégrales, vecteurs d'IC)
- Éventuel besoin en disque
- Patterns d'accès irréguliers à la RAM
- Efficacité CPU faible (loin de la performance crête)
- Méthodes itératives donc beaucoup de barrières de synchronisation

Conclusion sur les méthodes post-Hartree-Fock

- Communications all-to-all à cause de la transformation à 4 indices
- Gros besoins en RAM (intégrales, vecteurs d'IC)
- Éventuel besoin en disque
- Patterns d'accès irréguliers à la RAM
- Efficacité CPU faible (loin de la performance crête)
- Méthodes itératives donc beaucoup de barrières de synchronisation

On peut difficilement trouver pire pour les machines actuelles...

NOTRE APPROCHE DANS LE *QUANTUM PACKAGE*

APPROCHE CIPSI

- Revenir à une approche *determinant-driven* dans des espaces plus petits

- Revenir à une approche *determinant-driven* dans des espaces plus petits
- Sélection des déterminants au cours du calcul par leur contribution à l'énergie

$$\frac{\langle \Psi^{(n)} | \mathcal{H} | i \rangle^2}{E(n) - \langle i | \mathcal{H} | i \rangle} > \tau$$

- Revenir à une approche *determinant-driven* dans des espaces plus petits
- Sélection des déterminants au cours du calcul par leur contribution à l'énergie

$$\frac{\langle \Psi^{(n)} | \mathcal{H} | i \rangle^2}{E(n) - \langle i | \mathcal{H} | i \rangle} > \tau$$

- Calcul du reste en perturbation

- Revenir à une approche *determinant-driven* dans des espaces plus petits
- Sélection des déterminants au cours du calcul par leur contribution à l'énergie

$$\frac{\langle \Psi^{(n)} | \mathcal{H} | i \rangle^2}{E(n) - \langle i | \mathcal{H} | i \rangle} > \tau$$

- Calcul du reste en perturbation
- Si on laisse tourner le calcul indéfiniment, on atteint le Full-CI

RÈGLES DE SLATER

- En 2013 : Algorithme hyper-efficace pour calculer les règles de Slater[3]
- Utilise des instructions hardware (SSE4.2, $>$ 2008)
- Calcul de degré d'excitation en 5–10 cycles CPU ($\sim$ accès L2)
- Calcul des opérateurs d'excitation < 100 cycles CPU ($<$ accès RAM)
- Rend efficace les implémentations *determinant-driven*

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0000000011111111 : D_1
0010010010101111 : D_2
0010010001010000 : D_1 xor D_2
0000000001010000 : (D_1 xor D_2) and D_1
0010010000000000 : (D_1 xor D_2) and D_2

[3] A. S., E. Giner, *arXiv*:1311.6244 [physics.comp-ph] (2013)

CONCLUSION SUR LE APPROCHES CIPSI

- Communications all-to-all à cause de la transformation à 4 indices : implémentation openMP pour le moment
- Gros besoins en RAM (intégrales)
- Patterns d'accès réguliers à la RAM
- Efficacité CPU très bonne
- Méthodes itératives mais qui peuvent être rendues asynchrones (thèse de Y. Garniron)
- Très flexibles : Multi-reference Coupled Cluster sélectionné[4]

[4] E. Giner , G. David , A. S., J. P. Malrieu, *arXiv*:1509.03114 [physics.chem-ph] (2015)

CONCLUSION SUR LE APPROCHES CIPSI

- Communications all-to-all à cause de la transformation à 4 indices : implémentation openMP pour le moment
- Gros besoins en RAM (intégrales)
- Patterns d'accès réguliers à la RAM
- Efficacité CPU très bonne
- Méthodes itératives mais qui peuvent être rendues asynchrones (thèse de Y. Garniron)
- Très flexibles : Multi-reference Coupled Cluster sélectionné[4]

On peut espérer avoir une implémentation efficace

[4] E. Giner , G. David , A. S., J. P. Malrieu, *arXiv*:1509.03114 [physics.chem-ph] (2015)

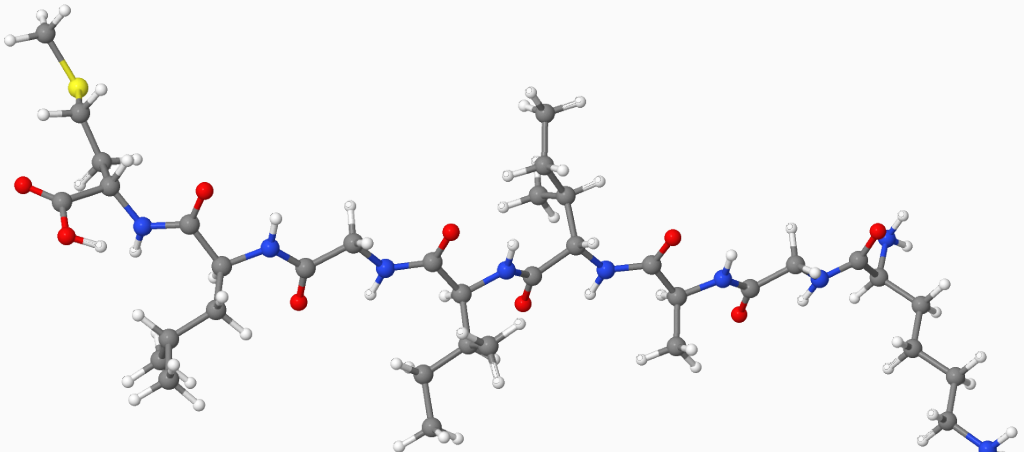
MÉTHODES STOCHASTIQUES

$$
\begin{aligned}
E &= \frac{\int \mathrm{d}\mathbf{r}_1, \ldots, \mathrm{d}\mathbf{r}_N \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N) \mathcal{H} \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N)}{\int \mathrm{d}\mathbf{r}_1, \ldots, \mathrm{d}\mathbf{r}_N \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N) \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_N)} \\
&= \sum \frac{\mathcal{H}\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N)}{\Psi(\mathbf{r}_1, \ldots, \mathbf{r}_N)} \text{ échantillonné avec } (\Psi \times \Phi)
\end{aligned}
$$

- Pas de calcul d'intégrale, donc pas de transformation à 4 indices.
- Nécessite d'évaluer Ψ, $\nabla_i \Psi$ et $\Delta_i \Psi$ à chaque position $\mathbf{r}_1, \ldots, \mathbf{r}_N$
- Toutes les positions $\mathbf{r}_1, \ldots, \mathbf{r}_N$ sont indépendantes $\rightarrow$ parallélisme massif

Code développé au LCPQ (M. Caffarel *et al*)

- Communications et I/O toujours non-bloquantes
- 98.4% d'efficacité parallèle sur 16 000 cœurs
- Communications avec ØMQ : élasticité des ressources, tolérance aux pannes
- Fonctionne sur Grille/Cloud
- En 2011 : 0.96 PFlops/s soutenus sur Curie pendant 24h (76 800 cœurs), 434 électrons, 2960 AOs
- 100 MiB par cœur, CPU/L3-bound



CODE QMC=CHEM : EFFICACITÉ MONO-CŒUR

- Difficulté : manipulation de petites matrices ($< 300 \times 300$)
- Utilisation de la simple précision quand c'est pertinent
- Produit matrice dense $\times$ vecteur creux pour calculer les MOs : atteint 100% du peak quand tout est en L1. 63.6% mesuré sur Sandy Bridge.[5]
- Uniquement FMA vectoriel dans les boucles internes, donc accélère systématiquement sur les nouvelles architecture
- $\mathcal{O}(N^2)$ plus efficace que des splines-3D en $\mathcal{O}(N)$
- Pour les fonctions d'onde multi-déterminantales (post-Full-CI)[6]: $\mathcal{O}(\sqrt{N_{\text{det}}})$

[5] A. S. M. Caffarel, E. Oseret, W. Jalby, *J. Comput. Chem.*, 31:11, 938–951 (2013)

[6] A. S. T. Applencourt, E. Giner, M. Caffarel, *arXiv*:????.???? [physics.comp-ph] (2015)

Cleland, Booth, Alavi *et al* (2009)

- Résolution des équations du Full-CI dans la base des déterminants avec un algorithme stochastique
- Contrairement à Davidson, pas besoin de stocker le vecteur propre dans la base complète
- Énergies Full-CI/cc-pCVQZ des diatomiques de la 1ère ligne
- Échantillonnage de la matrice densité à 2 corps, donc accès à toutes les propriétés
- CAS-SCF stochastique pour traiter des CAS de plusieurs dizaines d'électrons
- Nécessite d'avoir les intégrales moléculaires : transformation à 4 indices

Willow, Kim et Hirata (2012)

$$E = 2\sum_{i,j}^{\text{occ}}\sum_{a,b}^{\text{virt}} \frac{\langle ij|ab\rangle\langle ab|ij\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} - \sum_{i,j}^{\text{occ}}\sum_{a,b}^{\text{virt}} \frac{\langle ij|ab\rangle\langle ab|ji\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}$$

- But : suppression de la transformation à 4 indices et du stockage des intégrales en RAM

Willow, Kim et Hirata (2012)

$$E = 2\sum_{i,j}^{\text{occ}}\sum_{a,b}^{\text{virt}} \frac{\langle ij|ab\rangle\langle ab|ij\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} - \sum_{i,j}^{\text{occ}}\sum_{a,b}^{\text{virt}} \frac{\langle ij|ab\rangle\langle ab|ji\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}$$

- But : suppression de la transformation à 4 indices et du stockage des intégrales en RAM
- Transformée de Laplace, et intégration Monte Carlo dans un espace à 13 dimensions

$$\frac{1}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} = -\int_0^\infty \mathrm{d}\tau \exp\left[(\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b)\tau\right]$$

Willow, Kim et Hirata (2012)

$$E = 2\sum_{i,j}^{\text{occ}}\sum_{a,b}^{\text{virt}} \frac{\langle ij|ab\rangle\langle ab|ij\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} - \sum_{i,j}^{\text{occ}}\sum_{a,b}^{\text{virt}} \frac{\langle ij|ab\rangle\langle ab|ji\rangle}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b}$$

- But : suppression de la transformation à 4 indices et du stockage des intégrales en RAM
- Transformée de Laplace, et intégration Monte Carlo dans un espace à 13 dimensions

$$\frac{1}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} = -\int_0^\infty \mathrm{d}\tau \exp\left[(\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b)\tau\right]$$

$$E = \int \mathrm{d}\mathbf{r}_1 \ldots \int \mathrm{d}\mathbf{r}_4 \int \mathrm{d}\tau \frac{o(\mathbf{r}_1, \mathbf{r}_3, \tau) o(\mathbf{r}_2, \mathbf{r}_4, \tau) v(\mathbf{r}_1, \mathbf{r}_4, \tau) v(\mathbf{r}_2, \mathbf{r}_3, \tau)}{r_{12} r_{34}}$$

CONCLUSIONS

Pour avoir un programme efficace il faut:

- Réutiliser au maximum ce qui peut être dans les caches
- Minimiser les accès aléatoires à la RAM : avoir des accès réguliers
- Minimiser les accès au réseau (communications bloquantes)
- Oublier les I/O sur disque
- Utiliser des bibliothèques optimisées dès que possible (MKL, PETSc, MUMPS, *etc*)

Ce qui fonctionne mal

- Transformation à 4 indices : communications $1 \to N$ ou $N \to 1$
- Synchronisations : latence du réseau
- Accès aléatoires à la mémoire
- Accès au système de fichiers

Ce qui fonctionne mal

- Transformation à 4 indices : communications $1 \to N$ ou $N \to 1$
- Synchronisations : latence du réseau
- Accès aléatoires à la mémoire
- Accès au système de fichiers

Ce qui fonctionne bien

- Algorithmes qui prennent peu de RAM
- Accès régulier à la RAM
- Algorithmes stochastiques : parallélisme massif possible