Utilisation d'OCaml dans le contexte du (HPC)

Anthony Scemama

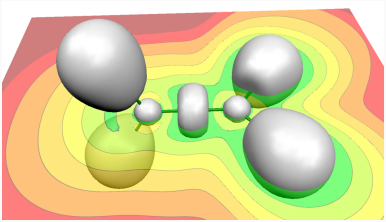
17/01/2023

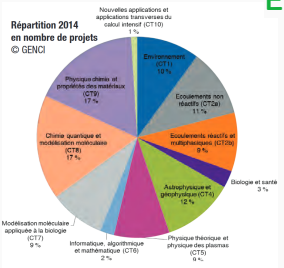
Laboratoire de Chimie et Physique Quantiques, UPS/CNRS Toulouse

Contexte

Chimie Quantique

- Description de la matière par la mécanique quantique (Eq. Schrödinger)
- Utilisateurs: chimistes théoriciens





Enjeux sociétaux

- Industrie pharmaceutique	Drug design
- Électronique	Nano- et micro-électronique
- Matériaux	Nanotubes de carbone, graphène
- Catalyse	Réactions enzymatiques, pétrole

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

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

E_0 : Énergie correspondante

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

Codes de calcul de structure electronique

Fortran

- La chimie quantique est depuis longtemps sur les supercalculateurs (cartes perforées, etc)
- Language très facile à prendre en main
- Compilateurs très efficaces: code compilé rapide & instructions SIMD
- Programmation multi-thread avec OpenMP (directives)

C++

- Effet de mode: le Fortran est vieillot
- Templates
- L'industrie (Intel) pousse pour le remplacement de Fortran par C++
- Utilisation d'accélérateurs (GPUs) mieux supportée qu'en Fortran

Aujourd'hui

- Ecrire un code tout en Fortran ou C++ est difficile à maintenir
- On voit arriver beaucoup de codes mixtes Python/C++/Fortran
- Il y a des tentatives avec Rust et Julia
- OCaml est toujours quasi inexistant dans cette communauté (mais il y a QCaml: `https://gitlab.com/scemama/qcaml`)

Parallelisme

Multi-thread: OpenMP

- Directives (pragmas)
- Parallélisation de boucles et/ou par tâches
- Extension aux accélérateurs GPU

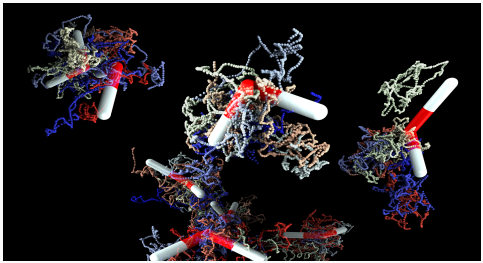
Distribué: Message Passing Interface (MPI)

- Standard absolu (algèbre linéaire, décomposition de domaines, etc)
- Paradigme Single Program Multiple Data (SPMD): tous les processus executent le même code, mais la fonction `MPI_RANK` renvoie un rang différent pour chacun
- Utilise the hardware à faible latence (Infiniband, etc)
- Pas de tolérance aux pannes: si un process meurt, tout le monde est tué
- Mais les évolutions de MPI tendent à le rendre plus général

QMC=Chem

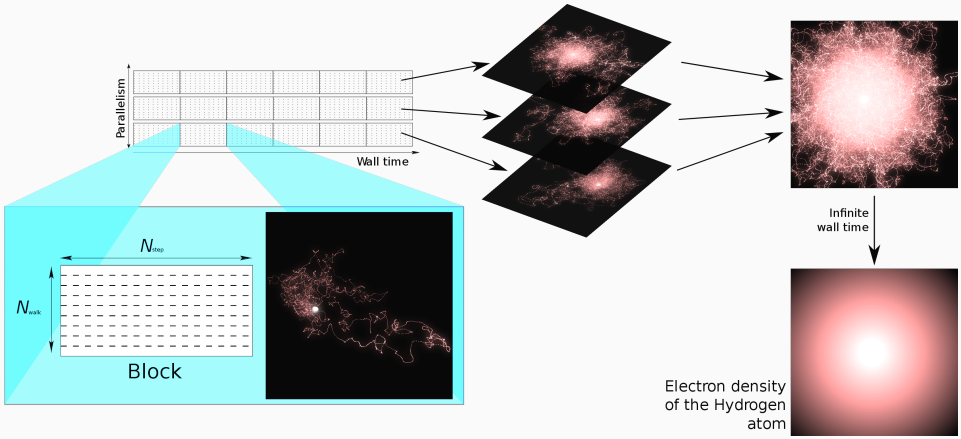
Monte Carlo Quantique (QMC)

- Calculs de structure électronique: problème à N-corps quantique
- Résolution de l'Eq. de Schrödinger en utilisant des marches aléatoires
- Calcul d'un grand nombre de trajectoires en parallèle
- Le résultat est la moyenne des énergies calculées le long des trajectoires



  - Méthode parmi les plus précises dans le domaine
  - Très coûteuse en CPU, peu en mémoire et en stockage
  - Embarrasingly parallel: parfaitement adapté aux supercalculateurs

Algorithme


Parallelism
Wall time
N_{step}
N_{walk}
Block
Infinite
wall time
Electron density
of the Hydrogen
atom

Particularités liées à la méthode

- Très peu de RAM par processus
- Tolérance aux pannes possible: si une machine plante, on perd de la statistique mais le résultat n'est pas faux
- Les processus n'ont pas besoin d'être synchronisés
- Les ressources n'ont pas besoin d'être fixes

Conséquences

- Le meilleur speedup est obtenu en mettant au moins une trajectoire par CPU: paralléliser au niveau des boucles sera moins bon
- OpenMP n'est pas necessaire
- MPI n'est pas le meilleur outil pour le parallélisme distribué
- Certains centres de calcul disent que ce n'est pas du "vrai" HPC

Design du programme

Fortran

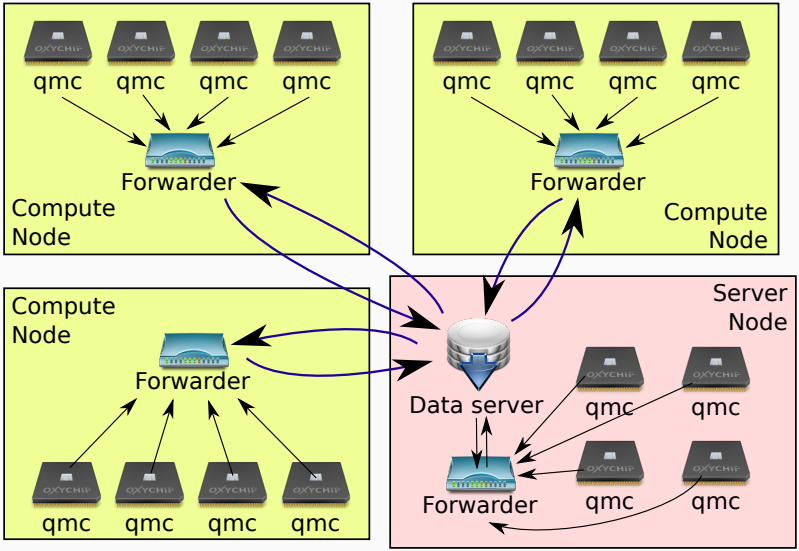
- Programme séquentiel équivalent à

```
while not !stop do
  let e = compute_block () in
  stop := send_to_dataserver(e)
done
```

OCaml:

- Data server:
  - Gestion des données d'input/output
  - Collecte periodiquement les résultats des processus Fortran
  - Arrête le calcul quand la condition d'arrêt est atteinte
- Forwarder: Un processus par noeud qui redirige les comm.
- Outils de pré/post traitement

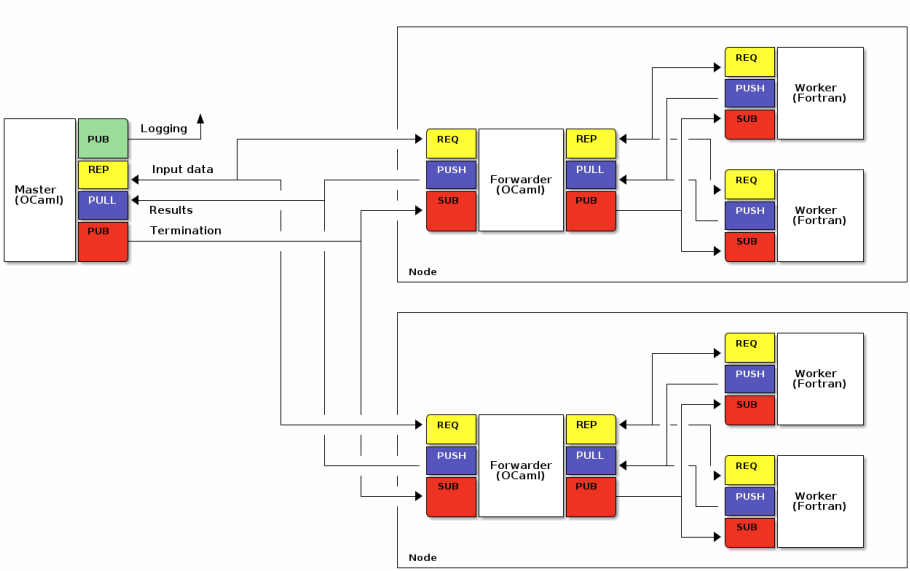
Design du programme



Communications

- ZeroMQ est une bibliothèque de communication basée sur l'échange de messages asynchrone
- Contrairement à MPI, ZeroMQ utilise TCP, donc la latence est beaucoup plus importante
- Moins facile à prendre en main que MPI
- Permet de faire communiquer facilement des processus différents, comme avec des sockets $\longrightarrow$ Multiple Program Multiple Data (MPMD)
- L'idée est de cacher la latence avec le côté asynchrone
- Permet d'implémenter la tolérance aux pannes et la gestion dynamique des ressources
- Bien adapté à un grand nombre de petits messages

Design du programme



OCaml

Types de messages échangés entre Fortran et OCaml:

```
(* Message.mli *)
type t =
  | Register    of Compute_node.t * int
  | Unregister  of Compute_node.t * int
  | Walkers     of Compute_node.t * int * (float array) array
  | GetWalkers  of Strictly_positive_int.t
  | Test
  | Property    of Block.t
  | Ezfio       of string
  | Error       of string

val of_string_list : string list -> t
val to_string      : t -> string
```

Fortran

```
use f77_zmq
integer(ZMQ_PTR) :: msg
msg  = f77_zmq_msg_new()  ! Buffer

call zmq_register_worker(msg)
call get_running(do_run)
do while (do_run == t_Running)
   block_id = block_id+1
   call compute_block()

   call zmq_send_header(msg,'e_loc',block_id)
   call zmq_send_scalar_prop(msg,block_weight,e_loc_block_walk)
   ! Sends "e_loc %s %d %d %f %f"
   !       hostname pid block_id block_weight e_loc_block_walk
   call zmq_send_header(msg,'elec_coord',block_id)
   call zmq_send_real(msg,elec_coord_full,size(elec_coord_full))
   call get_running(do_run)
end do
call zmq_unregister_worker(msg)
```

OCaml

```
let of_string_list m =
  try
    match m with
    | [ "register"   ; c ; pid  ] -> Register   (Compute_node.of_string c, int_of_string pid)
    | [ "unregister" ; c ; pid  ] -> Unregister (Compute_node.of_string c, int_of_string pid)
    | "elec_coord" :: c :: pid :: _ :: n :: walkers ->
       Walkers (Compute_node.of_string c, int_of_string pid, walkers)
    |  [ prop ; c ; pid ; b ; w ; v ] ->
        let open Block in
        Property
          { property     = Property.of_string prop;
            value        = Sample.of_float (float_of_string v);
            weight       = Weight.of_float (float_of_string w);
            compute_node = Compute_node.of_string c;
            pid          = int_of_string pid;
            block_id     = Block_id.of_int (int_of_string b);
          }
    | ...
```

OCaml

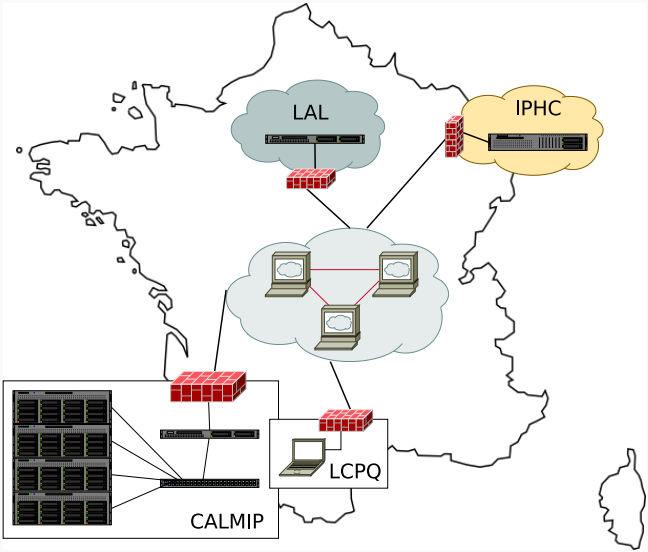
- Le typage du message permet de tester à la réception que le message est OK
- Une fois qu'on a une variable de type `Message.t`, on est sûr que le message est valide

Puis pattern-matching sur les messages pour décider de ce qu'il faut en faire.

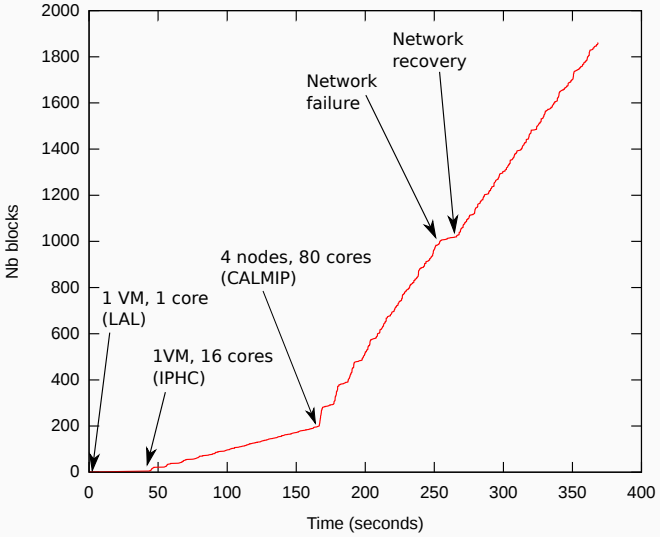
Particularités de QMC=Chem

- 76800 coeurs sur Curie (TGCC/CEA) en 2011, 0.96 PFlops/s
- 98.4% d'efficacité parallèle sur 16 000 coeurs (run de 3h)
- Tolérant aux pannes: on peut supprimer n'importe quel noeud sans arrêter le calcul
- On peut débrancher le réseau, puis le rebrancher sans interrompre le calcul
- On peut ajouter/supprimer des ressources à la demande
- Les processus de calcul peuvent tourner sur du hardware différent, tournant à des vitesses différentes

Calcul distribué sur Centre HPC et Cloud (2015)



Calcul distribué sur Centre HPC et Cloud (2015)



Quantum Package

Interaction de configurations

Méthode complètement différente de QMC.

$$\hat{H}\,\Psi(\mathrm{r}) = E\,\Psi(\mathrm{r}); \Psi(\mathrm{r}) = \sum_{i=1}^{N_{\mathrm{det}}} c_i \mathrm{D}_i(\mathrm{r})$$

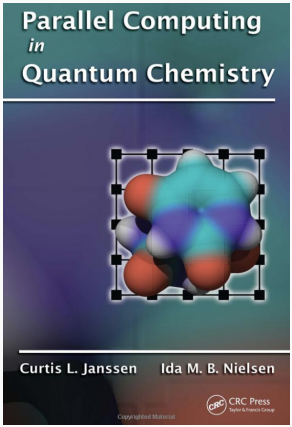
- $N_{\mathrm{det}} \sim 10^8$, $\mathrm{r} \in \mathbb{R}^{3N}$ pour N électrons
- On cherche les c_i qui minimisent l'énergie
- On calcule exactement les intégrales $H_{ij} = \int D_i(\mathrm{r})\hat{H}D_j(\mathrm{r})d\mathrm{r}$
- Algèbre linéaire: recherche de vecteur propre avec la méthode de Davidson

En bref
On fait des produits matrice-vecteur de dimension $\sim 10^8$

Parallelisme

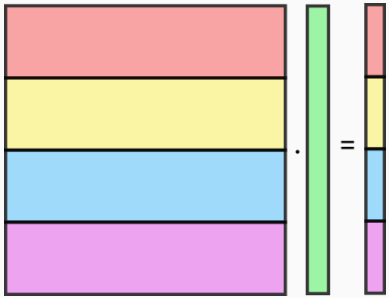
A grid network's performance, however, is too low (the bisection width is too small and the latency too large) to be of direct interest for the quantum chemistry applications discussed in this book.

- Ici, il serait naturel d'utiliser MPI
- Challenge: Peut-on montrer qu'ils se trompent?



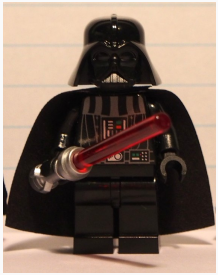
Produit Matrice-vecteur

- $AX = B$
- $N = 21691814$
- Parallélisation par tâches
- Chaque tâche produit un morceau de vecteur
- Chaque processus est capable de calculer n'importe quel élément de matrice (mémoire répliquée)
- La matrice est construite *sur la mouche*[1]



[1] "on the fly"

Design du programme: MPMD



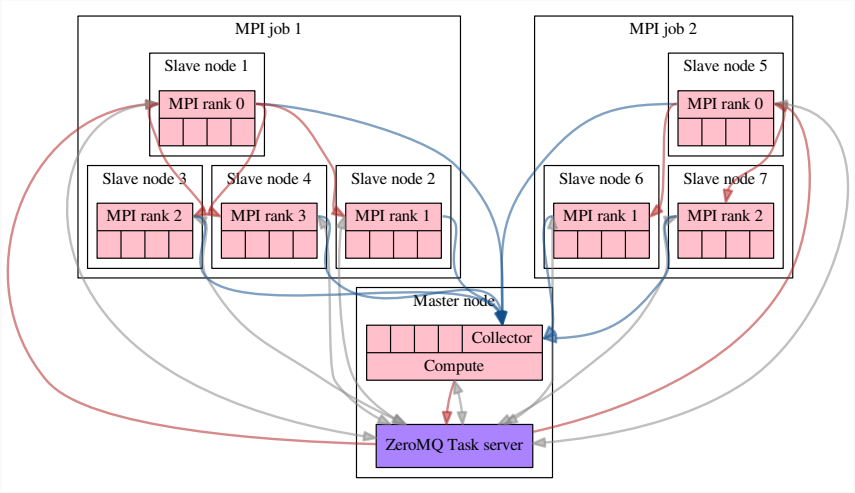
Master

Slave

Tunnel

- 1 executable OCaml: Gestionnaire de tâches
- 1 executable Fortran/OpenMP: Processus maître
- ≥ 0 executable(s) Fortran/MPI/OpenMP: Processus esclave
- Executables tunnels pour interconnecter des réseaux locaux
- MPI est utilisé pour faire des broadcasts rapides des vecteurs
- Le tout est interconnecté avec ZeroMQ

Design du programme: MPMD



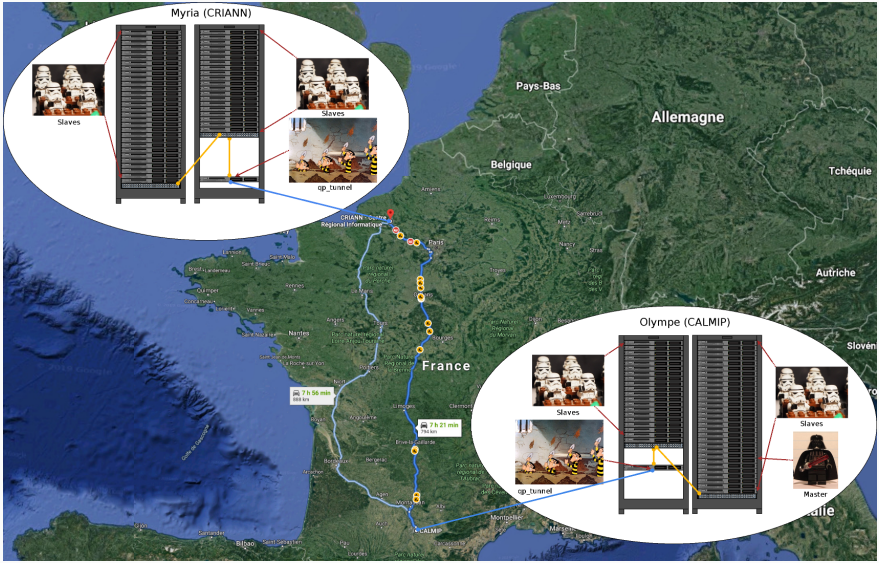
OCaml

```
(* TaskServer.mli *)
type t =
{
    queue              : Queuing_system.t ;
    state              : Message.State.t option ;
    address_tcp        : Address.Tcp.t option ;
    address_inproc     : Address.Inproc.t option ;
    progress_bar       : Progress_bar.t option ;
    running            : bool;
    accepting_clients  : bool;
    data               : (string, string) Hashtbl.t ;
}

(** Create/Finish a new job *)
val new_job : Message.Newjob_msg.t -> t -> [> `Req ] Zmq.Socket.t -> [> `Pair] Zmq.Socket.t -> t
val end_job : Message.Endjob_msg.t -> t -> [> `Req ] Zmq.Socket.t -> [> `Pair] Zmq.Socket.t -> t
```

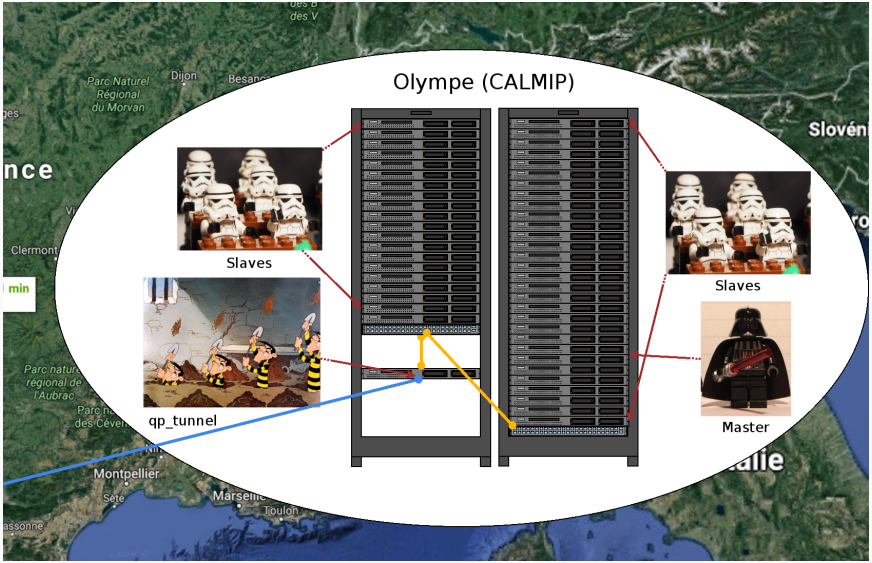
OCaml

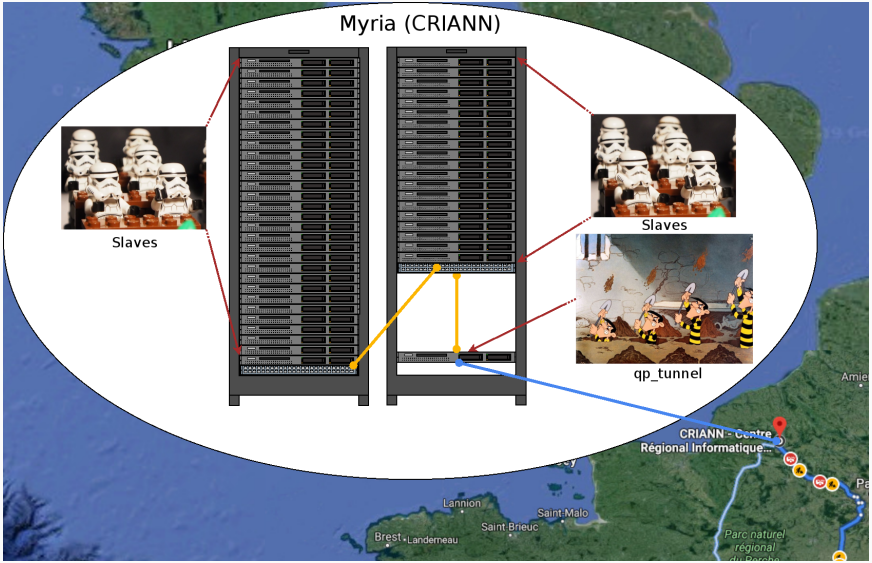
```
(** Add a task to the pool *)
val add_task: Message.AddTask_msg.t -> t -> [> `Req ] Zmq.Socket.t -> t

(** Connect/Disconnect a client *)
val connect: Message.Connect_msg.t -> t -> [> `Req ] Zmq.Socket.t -> t
val disconnect: Message.Disconnect_msg.t -> t -> [> `Req ] Zmq.Socket.t -> t

(** The client gets a new task to execute *)
val get_task: Message.GetTask_msg.t -> t -> [> `Req ] Zmq.Socket.t ->  [> `Pair] Zmq.Socket.t -> t

(** Mark the task as done by the client *)
val task_done: Message.TaskDone_msg.t -> t -> [> `Req ] Zmq.Socket.t -> t

(** Delete a task when it has been pulled by the collector *)
val del_task: Message.DelTask_msg.t -> t -> [> `Req ] Zmq.Socket.t -> t

(** Terminate server *)
val terminate : t -> [> `Req ] Zmq.Socket.t -> t
```

Exemple de calcul distribué



CALMIP


Olympe (CALMIP)
Slaves
qp_tunnel
Slaves
Master
Montpellier
Sete
Toulon
Dijon
Clermont
Slovéni


Myria (CRIANN)
Slaves
Slaves
qp_tunnel
CRIANN - Centre Régional Informatique...
Lannion
Saint-Malo
Saint-Brieuc
Brest
Landerneau
Parc naturel régional

Benchmark

Bande passante

CALMIP login ↔ CALMIP compute	IB EDR 100Gb/s
CRIANN login ↔ CALMIP login	Renater : 74.1 MB/s
CRIANN login ↔ CRIANN compute	Omnipath 100GiB/s

Latence (ping)

CALMIP login ↔ CALMIP compute	0.09 ms
CRIANN login ↔ CALMIP login	16.72 ms
CRIANN login ↔ CRIANN compute	0.23 ms

Résultats

- $N = 21\,691\,814$, 109 tâches
- 412 MiB envoyés à chaque groupe MPI au départ
- 165 MiB envoyés à chaque groupe à chaque itération
- 1.5 MiB de résultat par tâche
- 17 iterations

Configuration	coeurs	Temps
40 noeuds Olympe	1440	36:51
40 noeuds Myria	1120	44:10
20 noeuds Myria, 20 noeuds Olympe	1280	43:48