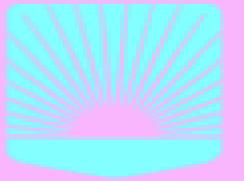




UNIVERSITÉ DE
VERSAILLES
ST-QUENTIN-EN-YVELINES



université PARIS-SACLAY

.....



#4

12/12/2025

jean-michel.batto@cea.fr

cea

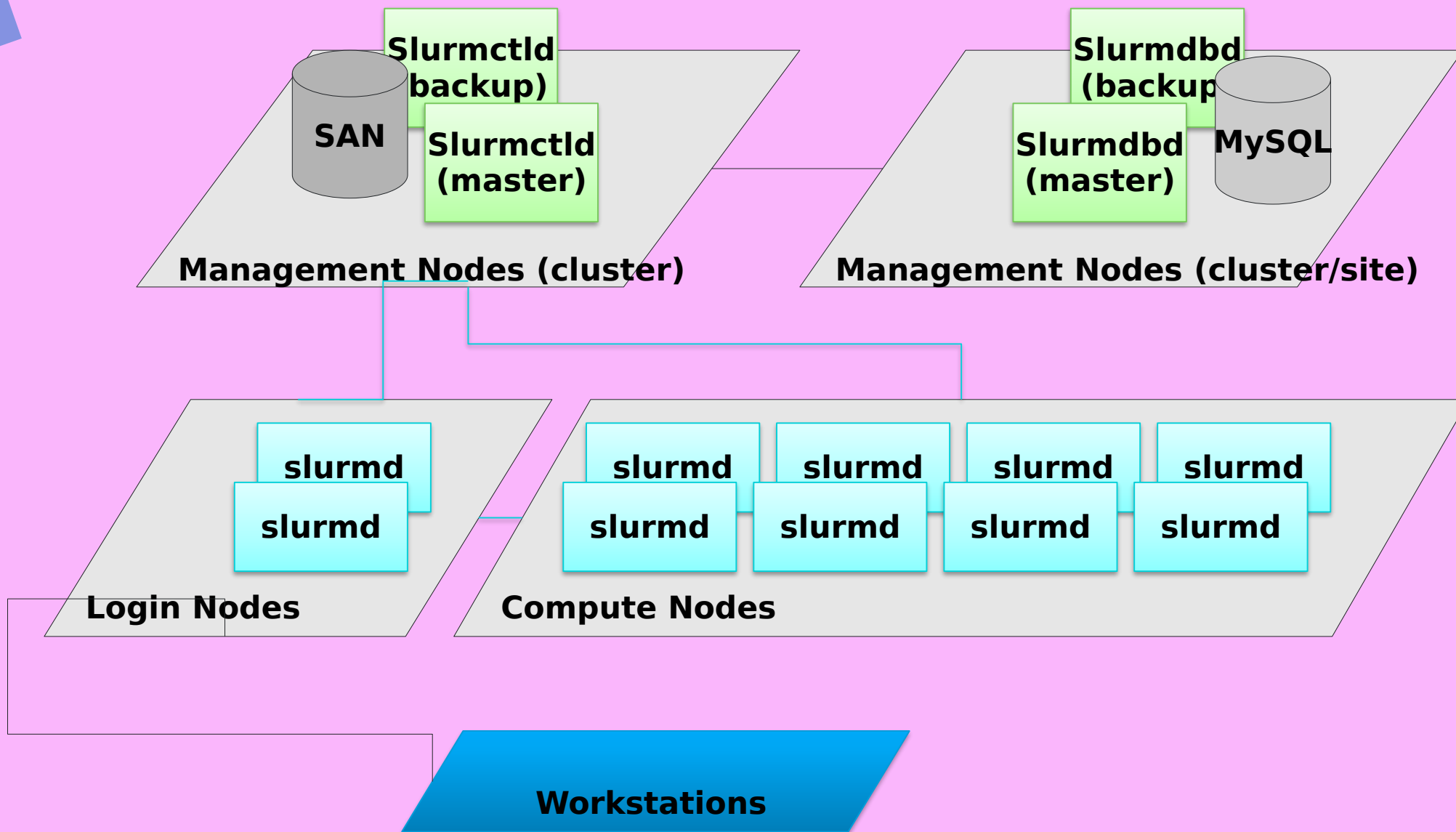
https://gogs.eldarsoft.com/M2_IHPS

JOUR / MOIS / ANNEE



Organisation physique

RECAP





❖ Sous Windows, activer l'option Enable host networking

Resources

AdvancedFile sharingProxies**Network**WSL integration

Configure the way Docker containers interact with the network

Docker subnet

192.168.65.0/24

default: 192.168.65.0/24

☒ **Enable host networking**

Host networking allows containers that are started with `--net=host` use localhost to connect to TCP and UDP services on the host. It automatically allows software on the host to use localhost to connect to TCP and UDP services in the container.

Problème de Docker SWARM → bridge et non overlay

❖ Faites : `docker network ls`

NETWORK ID	NAME	DRIVER	SCOPE
1fd28b9e7e85	bridge	bridge	local
b427088ec3d3	docker_gwbridge	bridge	local
a394d9d8be8d	host	host	local
yai4wmqyb2x2	ingress	overlay	swarm
09a46692cef5	none	null	local
7no4upxag9v3	yml_mpinet	overlay	swarm

Si `yml_mpinet` est « overlay » il faut le changer en « bridge »

`docker network rm ylm_mpinet`

//il faut le recréer sans « overlay » l'intérêt d'overlay est le multi-hôte avec SWARM

`docker network create --attachable yml_mpinet`

NETWORK ID	NAME	DRIVER	SCOPE
1fd28b9e7e85	bridge	bridge	local
b427088ec3d3	docker_gwbridge	bridge	local
a394d9d8be8d	host	host	local
m1gljp8vv77t	ingress	overlay	swarm
09a46692cef5	none	null	local
895bb87477c1	yml_mpinet	bridge	local



INTERACTIF

- ❖ Swarm init → déjà fait dans les TD précédents
- ❖ → Docker crée un contexte préfixé par le nom du répertoire...
- ❖ avec le docker-compose.yml, installation de
 - ❖ 1 db mysql pour le nœud slurmdbd,
 - ❖ 1 nœud slurmdbd,
 - ❖ 1 nœud de contrôle,
 - ❖ 2 nœuds de calcul, le tout sous MPI
- ❖ **docker compose up -d**
- ❖ **Dans cette nouvelle version on ajoute 2 nœuds et 1 conteneur avec vscode**
- ❖ Ouvrir 2 shells (un pour le nœud de contrôle, un pour le nœud C1)

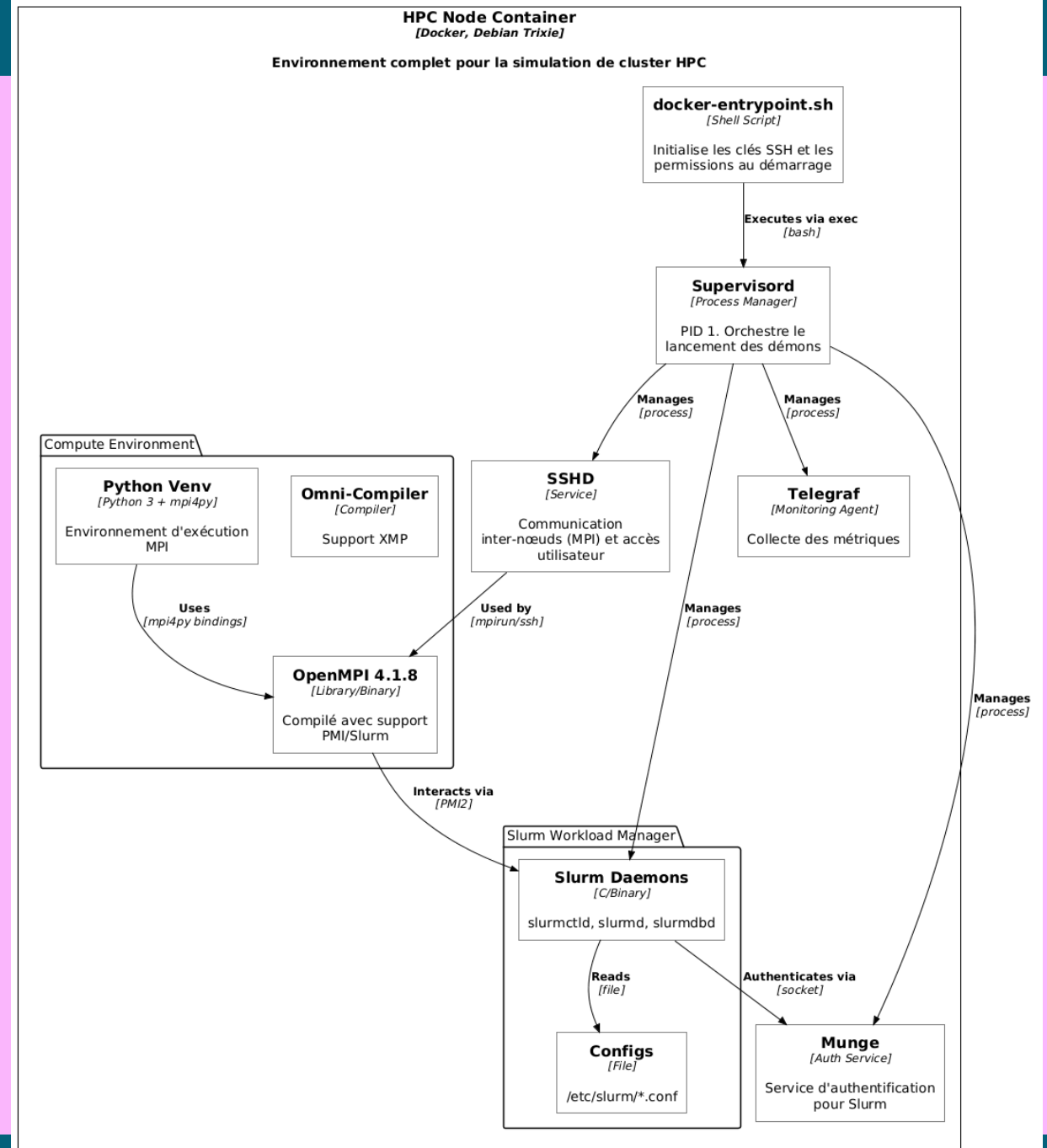
Ce que contient l'image jmbatto/m2chps-mpi41-slurm

Slurm (3 instances possibles du nœuds)

Démarrées avec Supervisord (et non avec tini)

➔ plusieurs daemon sont démarrés

- sshd
- telegraf
- slurm (avec un choix selon le paramétrage docker-compose)
- munge



3 étapes (step0/1/2) de découvertes par rapport aux contraintes

On travaille dans le conteneur slurmctld

❖ on vérifie s'il y a une partition pour root

```
sacctmgr show association -p user=root
```

```
scontrol show partition
```

```
scontrol show nodes
```

```
sinfo -Nel
```

❖ Sinon

```
sacctmgr --immediate add cluster name=linux
```

❖ Puis un restart des images

❖ Dans le répertoire /usr/local/var/mpishare faire

❖ `git clone http://gogs.eldarsoft.com/M2\_IHPS/glcs\_slurm.git`

Dans les répertoires test1/test2

```
mpicc -g3 -o elementary elementary.c
```



❖ Etapes du TD

- ❖ Explorer les répertoires step0, step1, step2

`salloc -n 2 mpirun ./elementary` → attention il faut que le binaire soit visible des nœuds (pb de partage du binaire – **modifier les scripts...**)

`sbatch script0.sh`

`sacct -j %jobid` obtenu au moment du `sbatch`

- ❖ Dans le répertoire step0, modifier le script pour avoir un code retour différent de 0

- ❖ Vérifier le résultat du batch

`sbatch -n 2 xxx.sh //(selon le répertoire)`

`squeue -s -j %jobid`

`sacct -j%jobid`

`squeue -s -i 30 -j %jobid`

`sacct -j %jobid`



❖ sbatch

- ❖ -N, --nodes=⟨minnodes⟩[-maxnodes]⟨size_string⟩
- ❖ Request that a minimum of minnodes nodes be allocated to this job.
- ❖ -n, --ntasks=⟨number⟩
- ❖ sbatch does not launch tasks, it requests an allocation of resources and submits a batch script. This option advises the Slurm controller that job steps run within the allocation will launch a maximum of number tasks and to provide for sufficient resources. The default is one task per node, but note that the --cpus-per-task option will change this default.

```
sbatch -n2 -N2 -exclusive xxx.sh
```

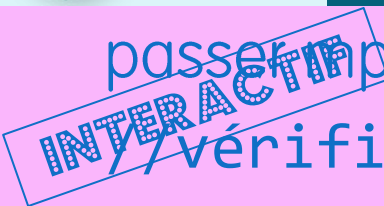
```
squeue -O jobid,state,qos,timeused
```



INTERACTIF

- ❖ SLURM « envoie » un signal SIGTERM avant la limite du temps du batch
- ❖ On veut observer la fin du batch → raison du répertoire step2

```
sacctmgr modify qos normal set MaxWall=00:00:02
```



passer et puis sur le noeud slurmctld (mais ça fonctionne aussi pour root)

//vérifie l'état de la queue

squeue

//puis allocation de 2 nœuds (il s'agit de c1 et c2)

salloc --time=05:00 -N 2

//vérification des noeuds

srun hostname

de la queue

squeue

//permet d'avoir les informations sur les jobs

sacct --format=JobID,elapsed,ncpus,ntasks,state,node



INTERACTIF

- ❖ `scontrol show node`
- ❖ Ou alors
- ❖ `sinfo -Nel`

Pour voir l'historique de la conso

- ❖ `sacct`

scontrol show node

Champ	Valeur Obtenue	Signification et Contexte
NodeName	c2	Le nom du nœud tel qu'il est défini dans la configuration Slurm (slurm.conf). C'est l'identifiant unique du nœud.
Arch	x86_64	L' architecture du processeur du nœud. Important pour la compatibilité des travaux.
CoresPerSocket	1	Le nombre de cœurs par socket du processeur.
ThreadsPerCore	1	Le nombre de threads par cœur (indique si l'hyperthreading est activé ou non). Ici, il est désactivé ou non visible.
CPUTot	1	Le nombre total de CPU (cœurs logiques) disponibles sur ce nœud pour les travaux Slurm.
CPUAlloc	0	Le nombre de CPU actuellement alloués (utilisés) pour l'exécution des travaux.
CPUEfctv	1	Le nombre de CPU effectifs disponibles pour les travaux. Peut être inférieur à CPUTot si des ressources sont réservées à l'OS ou à l'administration.
CPULoad	0.05	La charge moyenne actuelle du CPU (au moment de l'exécution de la commande). Une valeur faible indique que le nœud est peu sollicité.
RealMemory	1000	La mémoire physique totale disponible sur le nœud, en Mo (MegaBytes).
AllocMem	0	La mémoire totale actuellement allouée pour l'exécution des travaux (en Mo).
FreeMem	2799	La mémoire libre vue par Slurm (en Mo). <i>Note : La valeur peut être supérieure à RealMemory si le nœud n'est pas configuré pour limiter l'utilisation de la mémoire.</i>
TmpDisk	0	L'espace disque temporaire disponible pour les travaux Slurm (en Mo).
Partitions	docker	Les partitions de calcul auxquelles ce nœud est affecté. Les utilisateurs soumettent des travaux à ces partitions.
State	IDLE	L' état actuel du nœud . IDLE signifie que le nœud est libre et prêt à accepter de nouveaux travaux. (Autres états courants : ALLOCATED, DRAIN, DOWN).
Version	23.02.6	La version du démon Slurmd qui tourne sur ce nœud.
OS	Linux 5.15...WSL2	Le système d'exploitation et la version du noyau. L'indication WSL2 confirme que ce nœud est virtualisé dans un environnement Windows Subsystem for Linux 2.
BootTime	2025-12-10T12:51:03	L' heure de démarrage du nœud (système d'exploitation).
SlurmdStartTime	2025-12-10T15:15:26	L' heure de démarrage du démon Slurmd (le service Slurm sur ce nœud).
CfgTRES	cpu=1,mem=1000M,billing=1	Les ressources agrégées (TRES) configurées pour ce nœud. Indique la capacité maximale (1 CPU, 1000 Mo de mémoire).
AllocTRES	(vide)	Les ressources agrégées (TRES) actuellement allouées (vide car CPUAlloc=0).
AvailableFeatures	(null)	Fonctionnalités spécifiques que le nœud pourrait offrir (GPU, haute-vitesse Interconnect, etc.) pour le ciblage des travaux.



❖ 1/ tester en mode multi-nœud MPI un code XMP

❖ <https://omni-compiler.org/>

XcalableMP, XMP for short, is a directive-based language extension which allows users to develop parallel programs for distributed memory systems easily and to tune the performance by having minimal and simple notations.

Support typical parallelization under "global-view model"

XMP enables parallelizing the original sequential code using minimal modification with simple directives.

Support coarray to use one-sided communication easily under "local-view model"

Programmer can use coarray syntax in both XMP/Fortran and XMP/C. In particular, XMP/Fortran is designed to be compatible with Coarray Fortran.

Combination of MPI and OpenMP

In order to call an MPI program from an XMP program, XMP provides the MPI programming interface. Moreover, OpenMP directives can be combined into XMP as a hybrid programming.



- ❖ 2/test d'un code XMP, lancement du code
- ❖ 3/adaptation du code pour afficher le nom du noeud

```
mpiuser@62e939a516f3:~/YMLEnvironment/test-spawn-xmp$ cd /usr/local/var/mpishare/  
mpiuser@62e939a516f3:/usr/local/var/mpishare$ mpirun --mca orte_base_help_aggregate 0 --mca btl_tcp_if_inclu  
de 10.0.1.0/24 -n 4 -host 10.0.1.7,10.0.1.4,10.0.1.5,10.0.1.6 worker_program  
rank = 0 ; Processeur 0 - Nom : 62e939a516f3 (Longueur du nom : 12)  
  
(0, 0, 0) Processeur 2 - Nom : 21cdccdc88b1 (Longueur du nom : 12)  
rank = 2 ;  
(2, 2, 0)  
(2, 2, 1)  
(2, 3, 0)  
(2, 3, 1) rank = 3 ; Processeur 1 - Nom : f387910ddb9b (Longueur du nom : 12)  
Processeur 3 - Nom : 3fca87lee6fd (Longueur du nom : 12)  
  
(3, 2, 2)  
(3, 2, 3)  
(3, 3, 2)  
(3, 3, 3)  
(0, 0, 1)  
(0, 1, 0)  
(0, 1, 1) rank = 1 ;  
(1, 0, 2)  
(1, 0, 3)  
(1, 1, 2)  
(1, 1, 3) mpiuser@62e939a516f3:/usr/local/var/mpishare$
```



INTERACTIF

Dans `/YMLEnvironment/test-spawn-xmp` → faire un `make`

- ❖ ~~`sudo curl --unix-socket /var/run/docker.sock`~~
~~`http://localhost/containers/json | jq -r`~~
~~`'map(.NetworkSettings[]."yml_mpinet"."IPAMConfig"."IPv4Addr`~~
~~`ess") []'`~~

`mpirun --mca orte_base_help_aggregate 0 --mca btl_tcp_if_include`
`10.0.1.0/24 -n 4 -host c1,c2,c3,c4 worker_program`



```
#include <stdio.h>
#include "Matrix.xmptype.h"
#pragma xmp nodes p(2,2)
#pragma xmp template t(0:3,0:3)
#pragma xmp distribute t(block,block) onto p
XMP_Matrix A[4][4];
#pragma xmp align A[i][j] with t(j,i)
XMP_Matrix B[4][4];
#pragma xmp align B[i][j] with t(j,i)
XMP_Matrix C[4][4];
#pragma xmp align C[i][j] with t(j,i)
int main(int argc, char ** argv){
    int rank;
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    int i,j,n;
    n=4;
    fprintf(stderr,"rank = %d ; ",rank);

    #pragma xmp loop (i,j) on t(j,i)
    for (i=0;i<n;i++){
        for(j=0;j<n;j++){
            fprintf(stderr,"\n(%d, %d, %d) ",rank,i,j);
            C[i][j] = 0;
            A[i][j] = 1;
            B[i][j] = i*n+j+1;
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
}
```



INTERACTIF

```
// Allouer de la mémoire (dans cet exemple, pour une chaîne de caractères)
char* processor_name = malloc(256 * sizeof(char)); // Alloue de l'espace
pour le nom du processeur
int name_len;
MPI_Get_processor_name(processor_name, &name_len);

// Afficher le résultat
printf("Processeur %d - Nom : %s (Longueur du nom : %d)\n", rank,
processor_name, name_len);

// Libérer la mémoire allouée
free(processor_name);
```



Le code modifié

INTERACTIF

```
mpiuser@62e939a516f3: ~/YMLEnvironment/test-spawn-xmp
#pragma xmp nodes p(2,2)
#pragma xmp template t(0:3,0:3)
#pragma xmp distribute t(block,block) onto p

XMP_Matrix A[4][4];
#pragma xmp align A[i][j] with t(j,i)

XMP_Matrix B[4][4];
#pragma xmp align B[i][j] with t(j,i)

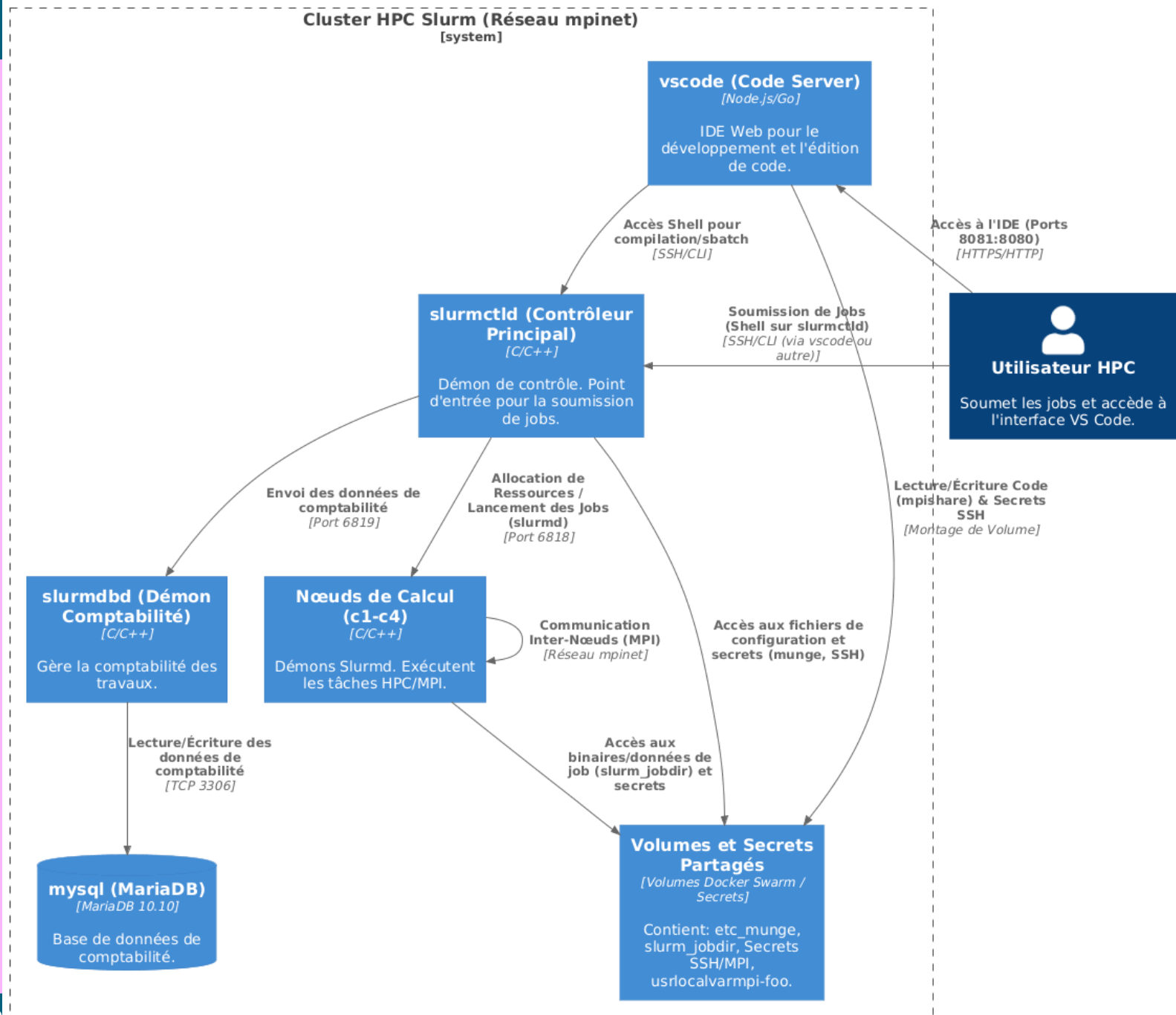
XMP_Matrix C[4][4];
#pragma xmp align C[i][j] with t(j,i)

int main(int argc, char ** argv){
    int rank;
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    int i,j,n;
    n=4;
    fprintf(stderr,"rank = %d ; ",rank);
    // Allouer de la mémoire (dans cet exemple, pour une chaîne de caractères)
    char* processor_name = malloc(256 * sizeof(char)); // Alloue de l'espace pour le nom du processeur
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Afficher le résultat
    printf("Processeur %d - Nom : %s (Longueur du nom : %d)\n", rank, processor_name, name_len);

    // Libérer la mémoire allouée
    free(processor_name);
#pragma xmp loop (i,j) on t(j,i)
    for (i=0;i<n;i++){
        for(j=0;j<n;j++){
            fprintf(stderr,"\n(%d, %d, %d) ",rank,i,j);
            C[i][j] = 0;
            A[i][j] = 1;
            B[i][j] = i*n+j+1;
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
}
```

mpiuser@62e939a516f3:~/YMLEnvironment/test-spawn-xmp\$





❖ Vous devez adapter le script docker-compose.yml

environment:

- PASSWORD=**1234**
- DOCKER_USER=**mpiuser**
- CODER_ACCESS_URL="http://localhost:8081"

user: "1001:1001"

➔ pour vous permettre de vous connecter à l'image

Installation du plugin SSH FS

INTERACTIF

Dans code-server on veut pouvoir se connecter à un nœud

❖ Avant : mettre à jour les certificats dans l'image vscode ➔

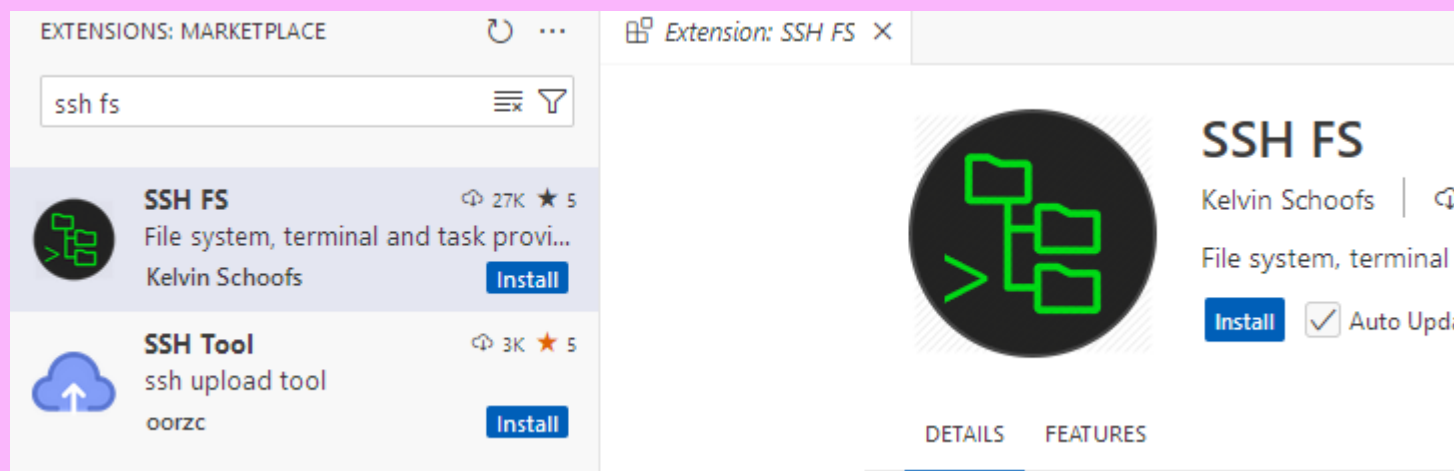
❖ Ouvrir un terminal, aller dans le dossier .ssh-source, faire un

```
>sh certif.sh
```

Cela va déployer les certificats dans le dossier «utilisateur docker» .ssh

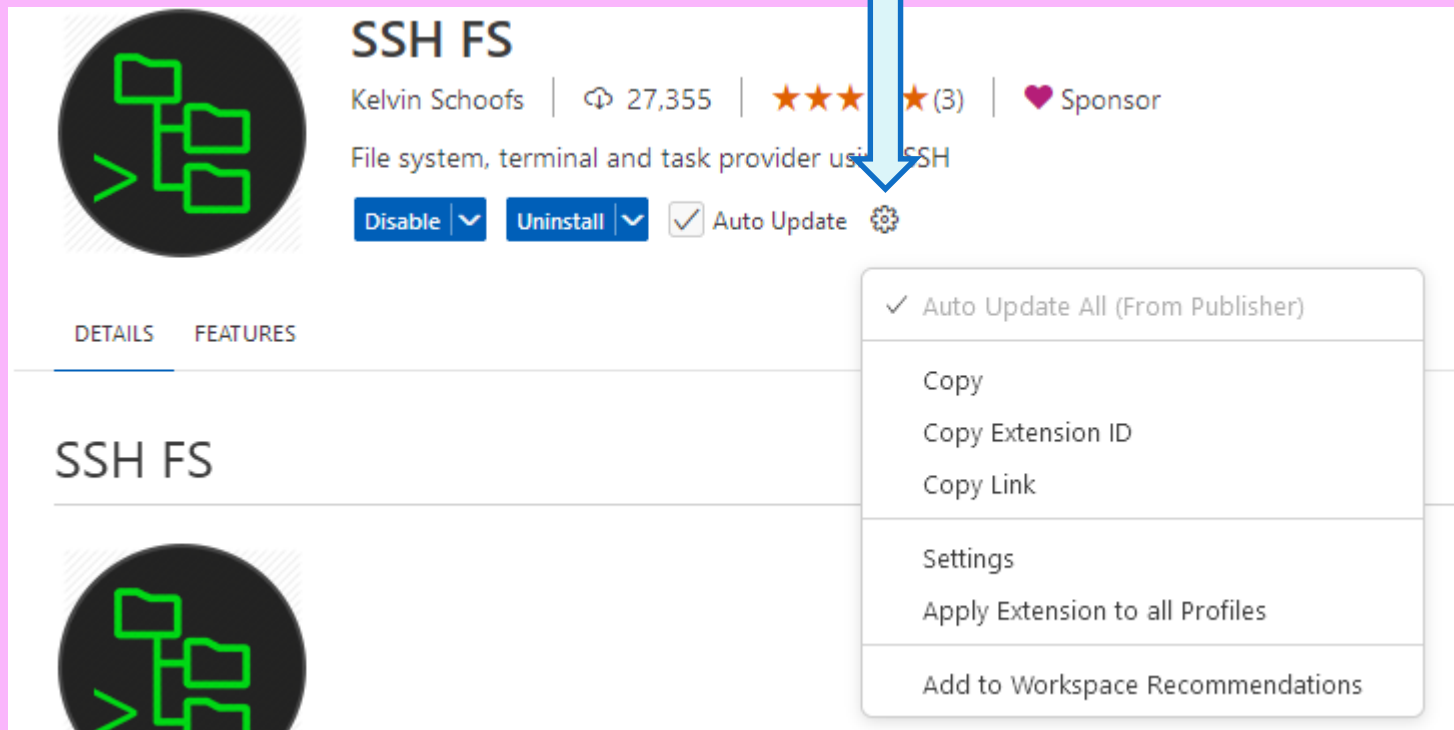
On vérifie en faisant un `ssh mpiuser@mpihead`

//mpiuser : compte non-root associé au login des nœuds et head



INTERACTIF

❖ L'extension a son paramétrage



SSH FS
Kelvin Schoofs | 27,355 | ★★★★★ (3) | Sponsor
File system, terminal and task provider using SSH

Disable Uninstall ☒ Auto Update 

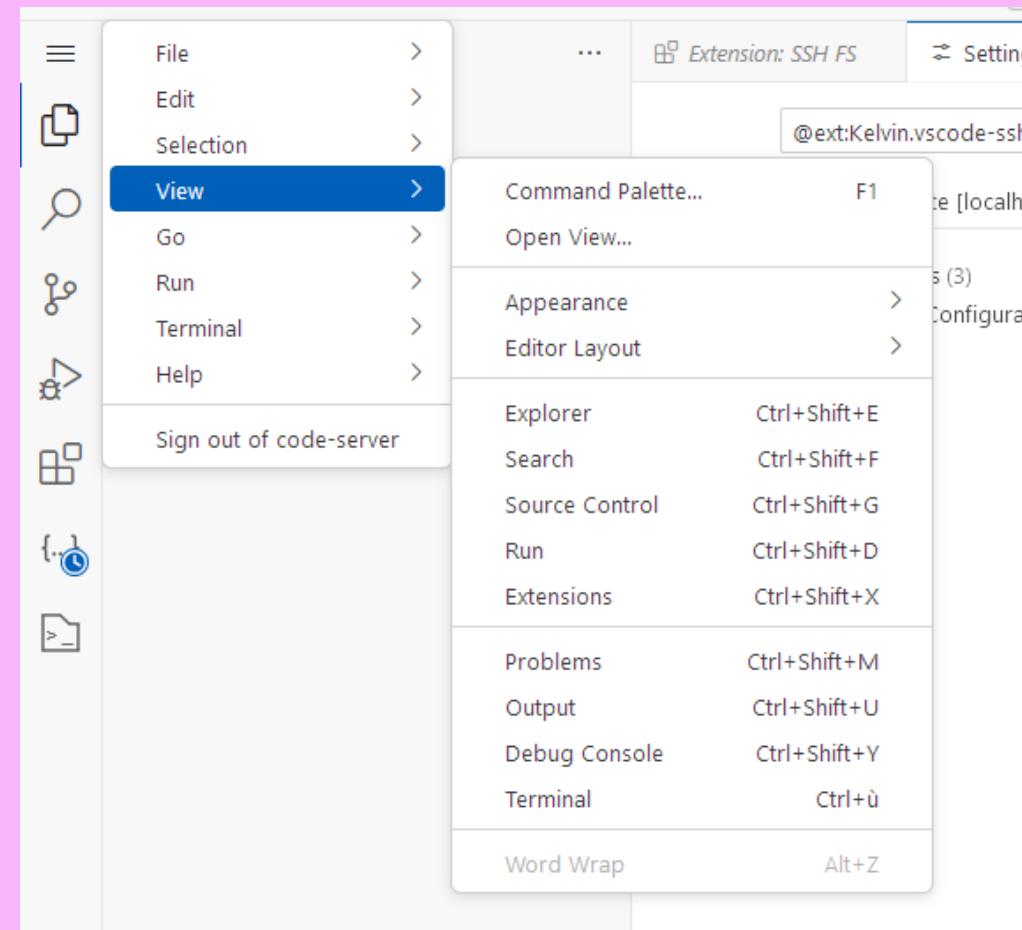
DETAILS FEATURES

SSH FS

- ☒ Auto Update All (From Publisher)
- Copy
- Copy Extension ID
- Copy Link
- Settings
- Apply Extension to all Profiles
- Add to Workspace Recommendations

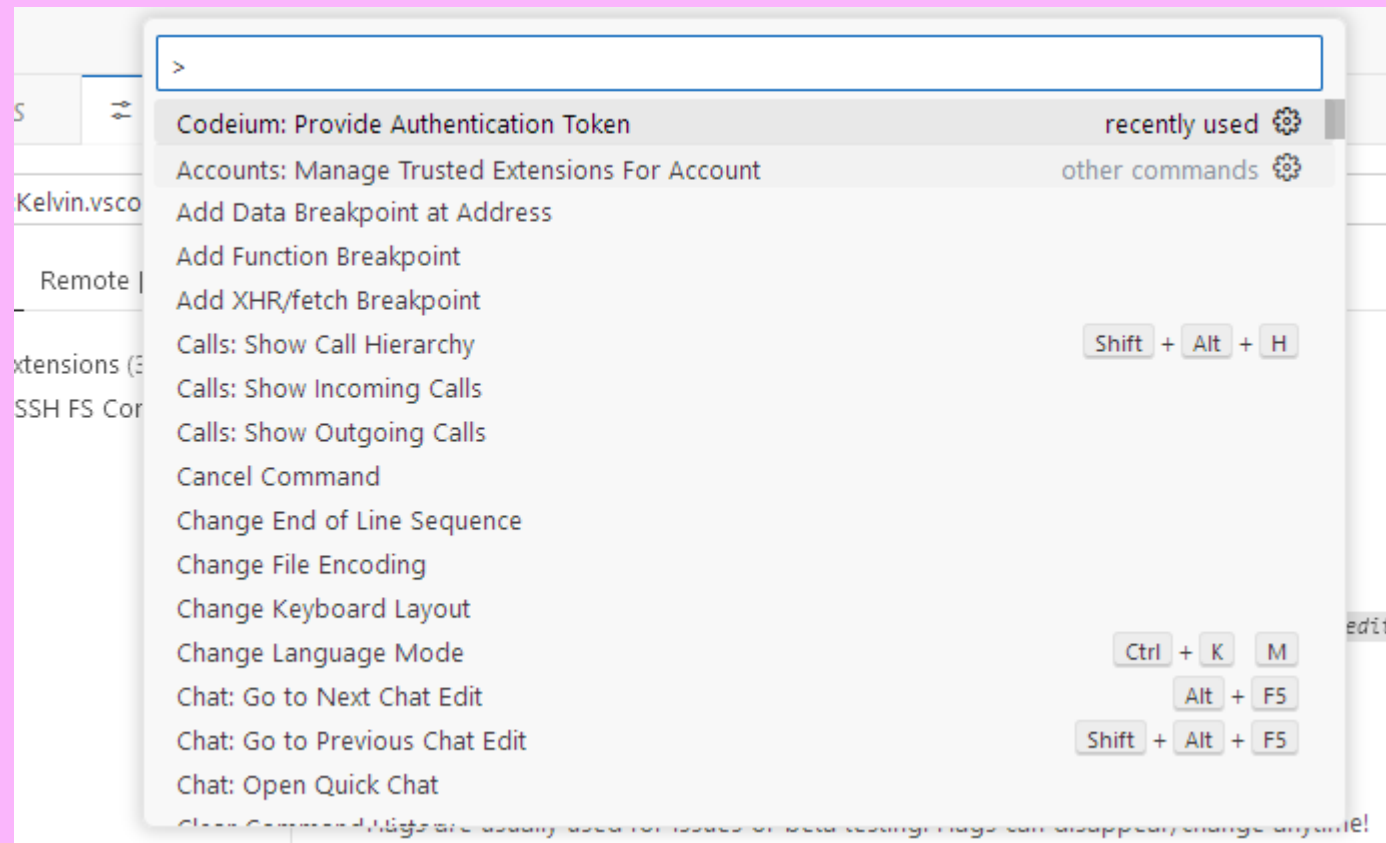
INTERACTIF

❖ On peut atteindre le paramétrage via la « Command Palette »



INTERACTIF

❖ Command palette



INTERACTIF

- ❖ La configuration
 - ❖ Name : mpihead

Create new configuration

Name
Name of the config. Accepted characters: [0-9a-z_+-.@]

mpihead

Location
The file or Settings file to add the new configuration to

Global settings.json ▼

Cancel

Save

INTERACTIF

❖ Configuration de ssh fs



CONFIGURATIONS

⚙️ mpihead mpihead

+

⌵

⚙️

↺

Back

mpihead
Global settings.json

Name

Name of the config. Can only exists of lowercase alphanumeric characters, slashes and any

mpihead

Label Optional

Label to display in some UI places (e.g. popups)

mpihead

Group Optional

Group for this config to group configs together in some UI places. Allows subgroups in the

- ❖ Host : slurmctld
- ❖ Root : /usr/local/var/mpishare
- ❖ Username : mpiuser
- ❖ Private key : /home/coder/.ssh/id_rsa

Username Optional

Username for authentication. Supports environment variables, e.g. \$USERNAME

Password Optional

Password for password-based user authentication. Supports env variables. This gets saved in plaintext! Using pr is recommended!

Private key Optional

A path to a private key. Supports environment variables, e.g. ` \$USERPROFILE/.ssh/myKey.ppk ` or ` \$HOME/.ssh/myKey `

Host Optional

Hostname or IP address of the server. Supports environment variables, e.g. \$HOST

Port Optional

Port number of the server. Supports environment variables, e.g. \$PORT

Root Optional

Path on the remote server that should be opened by default when creating a terminal or using the command/button. Defaults to ` / `

Agent Optional

Path to ssh-agent's UNIX socket for ssh-agent-based user authentication. Supports 'pageant' for variables, e.g. \$SSH_AUTH_SOCK

Username Optional

Username for authentication. Supports environment variables, e.g. \$USERNAME



❖ Installation de clang-format

Dans le dossier .ssh-source

```
sh install_clang_format_go.sh
```

➔ Installation du package et en plus il y a le compilateur go

❖ Installation de l'extension depuis vscode



RECAP

- exécutions parallèles de commandes avec démarrage synchrone et arrêt synchrone (au dernier élément)
- ❖ Echanges entre processus avec des E/S élémentaires via des **channels** bloquants
- ❖ Ceux-ci ont des actions déterminées en E/S sur l'état du processus
- ❖ Le concept de barrière non-déterministe
- ❖ Commande d'E/S avec barrière
- ❖ Commande d'E/S dans les boucles
- ❖ Capacité à choisir l'action en fonction du message



❖ Quels sont les avantages de prendre en charge les tests?

- ❖ Les fichiers `_test.go` exposent les tests !
- ❖ On conserve l'historique des tests (ça n'est plus dans le main)
- ❖ Apporte de la documentation
- ❖ Permet de faire de la non-régression
- ❖ Permet de faire de la validation fonctionnelle (ie. driver)
- ❖ ➔ fait parti des LOC (lines of code)

- ❖ Quels sont les tests ?
- ❖ Dans le langage Go, les tests sont pris en charge par le compilateur
- ❖ Pour tester un package (dans l'exemple c'est le package main):
 - ❖ Importer le package testing
 - ❖ Donner un nom de fichier en `_test.go`
 - ❖ Mettre des fonctions de signature `Test[Majuscule](t* testing.T)`
 - ❖ Par exemple : `func TestInsertHisto(t *testing.T) {`
- ❖ <https://play.golang.org/p/oSDZY0p0YC>

❖ Que décrit la variable `t*testing.T` ?

```
func TestMemoCds (t *testing.T) {  
    var s memoCds  
    s.lenCds = 1  
    s.rawCds = "something"  
    s.reset()  
    fmt.Printf("s %+v\n", s)  
    s.appendR('A')  
    fmt.Printf("s %+v\n", s)  
}
```

<https://play.golang.org/p/2RbN69wBMmS>



- ❖ Objet t: 2 méthodes – Log,Error
- ❖ Un test est pensé pour une utilisation « automatique » → l'idée d'un PASS/FAIL

```
func TestMemoCdsReset(t *testing.T) {  
    var s memoCds  
    s.lenCds = 1  
    s.rawCds = "something"  
    //s.reset()  
    if (s != memoCds{}) {  
        t.Error("s.reset doesn't work!")  
    } else {  
        t.Log("s.reset works")  
    }  
}
```

RECAP

❖ Présentation de go.mod = gestion des modules

❖ go mod init, go mod tidy

module ExtractCDS

go 1.16

```
require (  
    configExtract v0.0.0-00010101000000-000000000000  
    gonum.org/v1/plot v0.10.0  
)
```



VERSION

```
replace configExtract => ./configExtract
```



- ❖ Les workers pour faire du HPC
- ❖ Le calcul HPC repose sur la parallélisation des traitements

Un exemple de traitement : la somme de 2 matrices

Une matrice = un tableau 1 dimension qui est adressable

//Seq function for addition of two matrix

```
func addMat(mat1, mat2 []float64) {  
    n := len(mat1)  
    for i := 0; i < n; i++ {  
        mat1[i] = mat1[i] + mat2[i]  
    }  
}
```

<https://play.golang.org/p/7RWoK-gPjsT>

❖ Une version parallèle serait...

```
func addMatWorker(mat1, mat2 []float64) {  
    var wg sync.WaitGroup  
    n := len(mat1)  
    for i := 0; i < Workers; i++ {  
        wg.Add(1)  
        go func() {  
            defer wg.Done()  
            for i := 0; i < n; i++ {  
                mat1[i] = mat1[i] + mat2[i]  
            }  
        }()  
    }  
    wg.Wait()  
}
```

<https://play.golang.org/p/ZT7Ilhfxkqa>



le worker — basé sur Waitgroup

```
func worker(id int, wg *sync.WaitGroup) {  
    defer wg.Done()  
    fmt.Printf("Worker %d starting\n", id)  
    time.Sleep(time.Second)  
    fmt.Printf("Worker %d done\n", id)  
}
```

```
func main() {  
    var wg sync.WaitGroup  
    for i := 1; i <= 5; i++ {  
        wg.Add(1)  
        go worker(i, &wg)  
    }  
    //wg.Add(5)  
    wg.Wait()  
}
```

<https://play.golang.org/p/pdvJt72GkG2>



On définit un pool de worker → pour paralléliser le traitement

```
func worker(id int, wg *sync.WaitGroup, jobChannel <-chan Job,
    resultChannel chan JobResult) {
    defer wg.Done()
    for job := range jobChannel {
        resultChannel <- DistanceFasta36(job.idi, job.idj, job.b1, job.b2,
        job.o)
    }
}
```



Les structures pour les workers

```
type Job struct {
    b1 string
    b2 string
    idi int
    idj int
}

type JobResult struct {
    d float64
    idi int
    idj int
}

func DistanceFasta36(i int, j int) (res JobResult) {
    res.idi = i
    res.idj = j
    res.d = float64(i) + float64(j)
    return
}

func worker(id int, wg *sync.WaitGroup, jobChannel <-chan Job,
    resultChannel chan JobResult) {
    defer wg.Done()
    for job := range jobChannel {
        resultChannel <- DistanceFasta36(job.idi, job.idj)
    }
}
```



Version avec worker 1/2

```
var countIteration int
func CalculateDistances(bacterias []string) [][]float64 {
    n := len(bacterias)
    var matrix = make([][]float64, n)
    for rowid := range matrix {
        matrix[rowid] = make([]float64, n)
    }
    var wg sync.WaitGroup
    totalJobChannel := 0
    for idi := range bacterias {
        for idj := 0; idj <= idi; idj++ {
            totalJobChannel++
        }
    }
    jobChannel := make(chan Job)
    jobResultChannel := make(chan JobResult, totalJobChannel)
    const NumberOfWorkers = 8
    wg.Add(NumberOfWorkers)
    for i := 0; i < NumberOfWorkers; i++ {
        go worker(i, &wg, jobChannel, jobResultChannel)
    }
}
```



Version avec worker 2/2

```
for idi := range bacterias {  
    for idj := 0; idj <= idi; idj++ { //=  
        jobChannel <- Job{bacterias[idj], bacterias[idi],  
        idi, idj}  
        countIteration++  
    }  
}  
close(jobChannel)  
wg.Wait()  
close(jobResultChannel)  
// Receive job results from workers  
for result := range jobResultChannel {  
    fmt.Printf("idi %d, idj %d, d %f\n", result.idi,  
result.idj, result.d)  
    matrix[result.idi][result.idj] = result.d  
    matrix[result.idj][result.idi] = result.d  
}  
return matrix  
}
```

<https://play.golang.org/p/2AtBnDQVhIF>



❖ On doit partitionner le travail

➔ chaque Worker doit connaître sa partition `//begin,end`

```
func addMatWorker(mat1, mat2 []float64) {  
    var wg sync.WaitGroup  
    n := len(mat1)  
    for i := 0; i < Workers; i++ {  
        wg.Add(1)  
        go func() {  
            defer wg.Done()  
            for i := 0; i < n; i++ {  
                mat1[i] = mat1[i] + mat2[i]  
            }  
        }()  
    }  
    wg.Wait()  
}
```

<https://play.golang.org/p/txazepceYys>

```
❖ //compute indices (start, end) for threads
func GetInfoThread(startTotal, endTotal int, workers,
    i int) (begin int, end int) {
    workTotal := endTotal - startTotal
    work := workTotal / workers
    res := workTotal % workers
    if i < res {
        begin = ((work + 1) * i) + startTotal
        end = begin + (work + 1)
    } else {
        begin = ((work+1)*res + (work)*(i-res)) +
startTotal
        end = begin + (work)
    }
    return
}
```

<https://play.golang.org/p/dXHSFrQIb0O>

❖ Benchmark

- ❖ Le compilateur réalise les benchmarks dans une approche inversée (fenêtre de temps)
- ❖ L'objectif du benchmark – avoir le moins d'artefact dans la mesure
- ❖ Dans le fichier `_test.go`, signature `BenchmarkXxx(b *testing.B)`

```
func BenchmarkAddMat100(b *testing.B) {  
    for i := 0; i < b.N; i++ {  
        b.StopTimer()  
        rand.Seed(time.Now().UnixNano())  
        mat1 := createMat(100, rand.Float64())  
        mat2 := createMat(100, rand.Float64())  
        b.StartTimer()  
        addMat(mat1, mat2)
```

❖ Résultat du benchmark

goos: windows

goarch: amd64

pkg: coursM2/cm6worker

BenchmarkAddMat100-8 **5806759** **202** ns/op

➔ le traitement considéré dure 202ns

➔ il a été testé environ 6 millions de fois (en 2 sec – attention au stopTimer)

❖ On peut faire des benchmarks paramétrables :

```
func benchmarkAddWorker(dim int, b *testing.B) {  
    for i := 0; i < b.N; i++ {  
        b.StopTimer()  
        rand.Seed(time.Now().UnixNano())  
        mat1 := createMat(dim, rand.Float64())  
        mat2 := createMat(dim, rand.Float64())  
        b.StartTimer()  
        addMatWorker(mat1, mat2)  
    }  
}
```



BenchmarkAddMat100-8 5806759 202 ns/op
BenchmarkAddMatWorker100-8 359226 3806 ns/op
BenchmarkAdd1000-8 1528250 738 ns/op
BenchmarkAddWorker1000-8 231120 5096 ns/op
BenchmarkAdd10000-8 139465 8696 ns/op
BenchmarkAddWorker10000-8 87777 13494 ns/op
BenchmarkAdd20000-8 78723 16471 ns/op
BenchmarkAddWorker20000-8 48614 26756 ns/op
BenchmarkAdd50000-8 25012 45813 ns/op
BenchmarkAddWorker50000-8 31179 39287 ns/op
BenchmarkAdd100000-8 10100 113439 ns/op
BenchmarkAddWorker100000-8 16635 71449 ns/op

➔conclusions?

❖ Benchmark

- ❖ Le compilateur réalise les benchmarks dans une approche inversée (fenêtre de temps)
- ❖ L'objectif du benchmark – avoir le moins d'artefact dans la mesure
- ❖ Dans le fichier _test.go, signature **BenchmarkXxx(b *testing.B)**

```
func BenchmarkAddMat100(b *testing.B) {  
    for i := 0; i < b.N; i++ {  
        b.StopTimer()  
        rand.Seed(time.Now().UnixNano())  
        mat1 := createMat(100, rand.Float64())  
        mat2 := createMat(100, rand.Float64())  
        b.StartTimer()  
        addMat(mat1, mat2)  
    }  
}
```

❖ Résultat du benchmark

goos: windows

goarch: amd64

pkg: coursM2/cm6worker

BenchmarkAddMat100-8 **5806759** **202** ns/op

➔ le traitement considéré dure 202ns

➔ il a été testé environ 6 millions de fois (en 2 sec – attention au stopTimer)

❖ On peut faire des benchmarks paramétrables :

```
func benchmarkAddWorker(dim int, b *testing.B) {  
    for i := 0; i < b.N; i++ {  
        b.StopTimer()  
        rand.Seed(time.Now().UnixNano())  
        mat1 := createMat(dim, rand.Float64())  
        mat2 := createMat(dim, rand.Float64())  
        b.StartTimer()  
        addMatWorker(mat1, mat2)  
    }  
}
```



BenchmarkAddMat100-8 5806759 202 ns/op
BenchmarkAddMatWorker100-8 359226 3806 ns/op
BenchmarkAdd1000-8 1528250 738 ns/op
BenchmarkAddWorker1000-8 231120 5096 ns/op
BenchmarkAdd10000-8 139465 8696 ns/op
BenchmarkAddWorker10000-8 87777 13494 ns/op
BenchmarkAdd20000-8 78723 16471 ns/op
BenchmarkAddWorker20000-8 48614 26756 ns/op
BenchmarkAdd50000-8 25012 45813 ns/op
BenchmarkAddWorker50000-8 31179 39287 ns/op
BenchmarkAdd100000-8 10100 113439 ns/op
BenchmarkAddWorker100000-8 16635 71449 ns/op

➔conclusions?

- ❖ Une utilisation efficace des inline (vectorisation)
- ❖ Une représentation des variables concise et vectorisée dès que possible

```
type Location struct {  
    X,Y,Z float64  
}
```

$8 \times 3 = 24$ octets

```
var Locations [1000]Location
```

→ stockage séquentiel du tableau



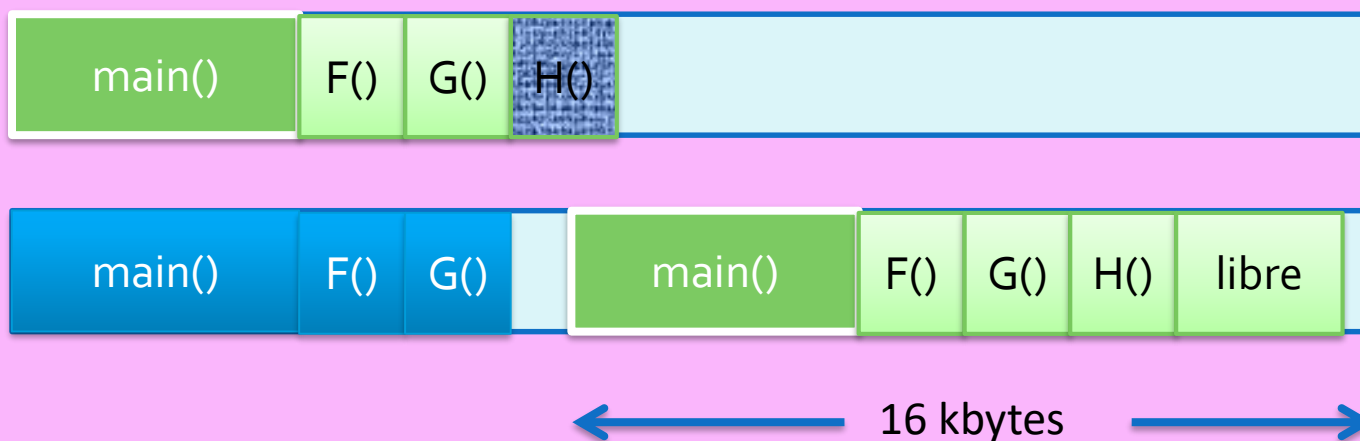
❖ Pthread :

- ❖ pas d'interaction avec un ramasse-miette
- ❖ pas d'interaction avec le compilateur (pas de vérification statique, pas d'optimisation sur les registres)

❖ Goroutines

- ❖ Sont ordonnancées de manière coopérative
- ❖ Les points de commutation sont pré-choisis (sur les syscall bloquants, channel, ramasse-miette)
- ❖ Le compilateurs connaît les registres utilisés et sait les sauvegarder à moindre cout

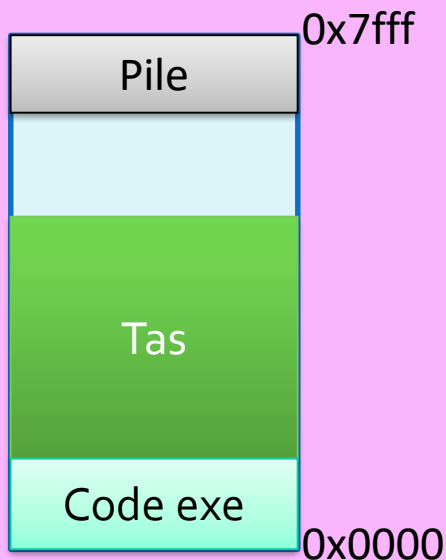
La pile qui se recopie



- ❖ Une optimisation possible par la gestion des goroutines via le loader : la recopie de la pile en cas d'espace insuffisant pour une nouvelle goroutine (H)

❖ Escape analysis // analyse d'échappement

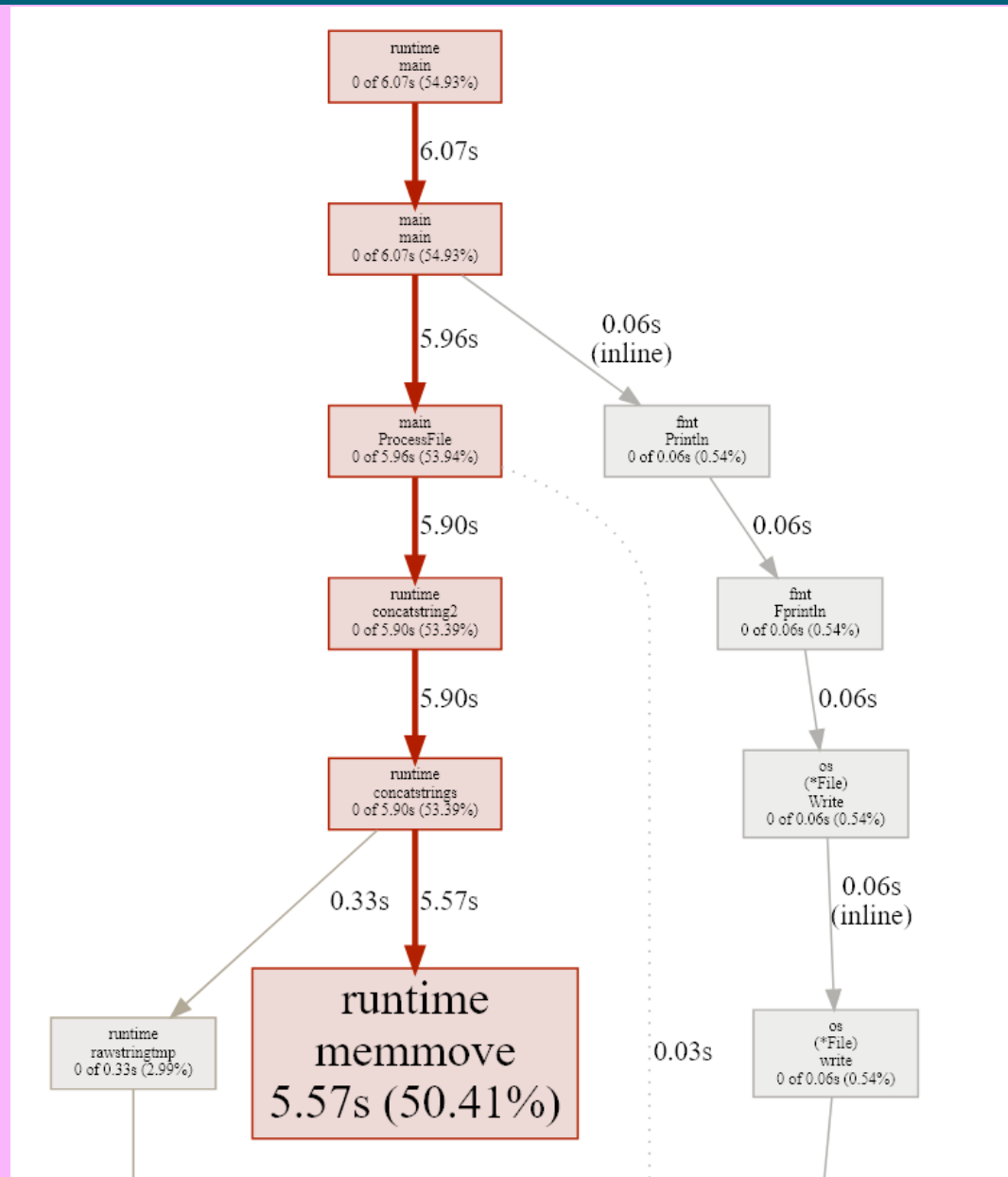
- ❖ → analyse statique qui vise à allouer les variables sur la pile (=économie en cas de changement de contexte, pas de gestion de la désallocation)
 - Contrairement à C qui force à choisir entre le Tas (malloc) et la pile (déclaration statique) et qui laisse le codeur faire son optimisation
 - → Go fait de l'analyse d'échappement



❖ Version non optimisée

12,8 secondes

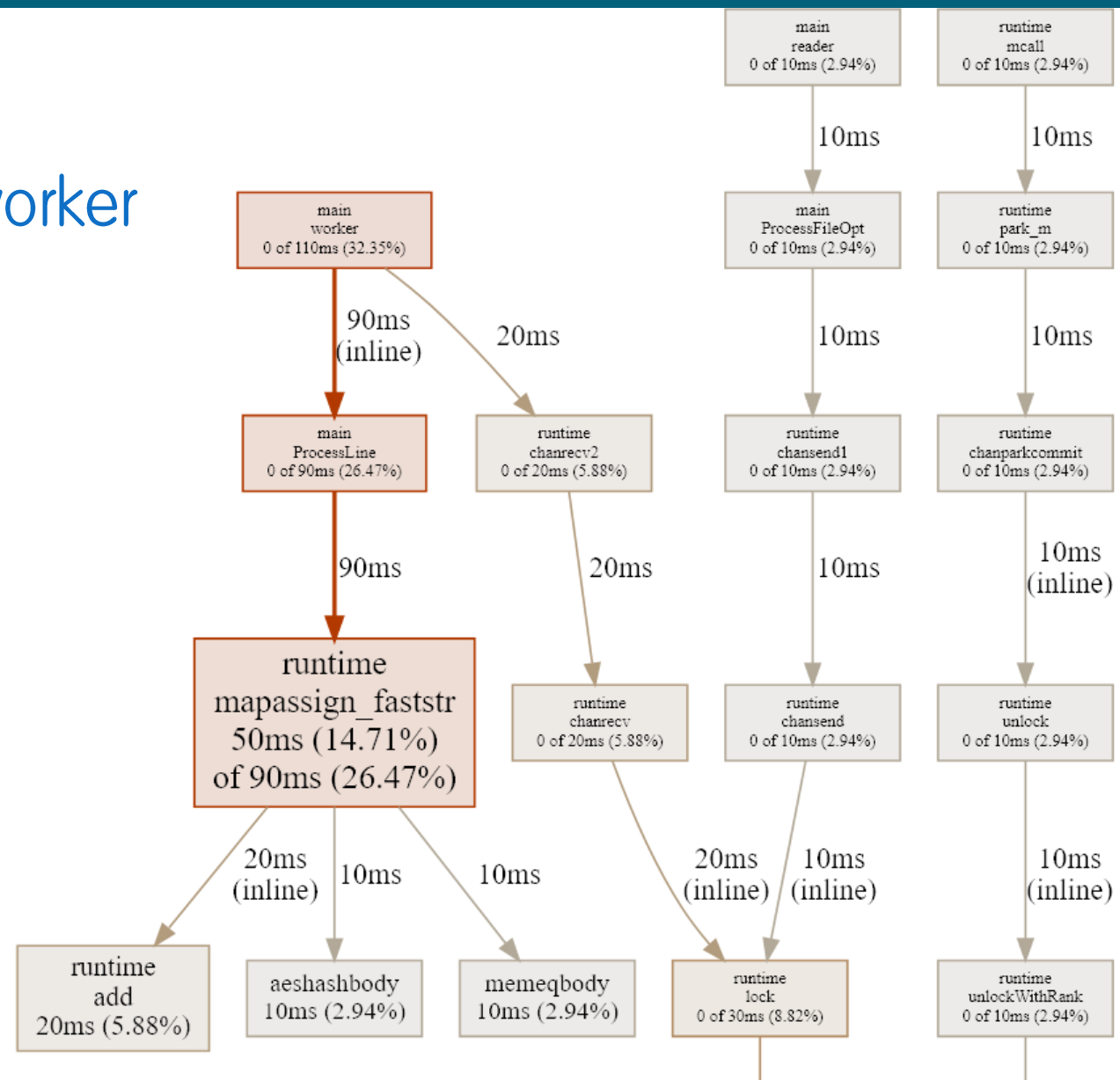
Le point chaud est la lecture du fichier



❖ Version optimisée avec worker

❖ ➔ la cible

292 millisecondes



- ❖ L'héritage nous habitue à un formalisme
- ❖ Que souhaite-t-on?

```
type packet struct {  
    targetData []byte  
    loaderData func(icount int) error  
}
```

- ❖ ➡ avoir un objet et sa méthode de remplissage des données

<https://play.golang.org/p/BTrrZGypnD1>



```
type packet struct {  
    targetData []byte  
    loaderData func(icontains int) error  
}  
  
func NewPacket() (obj *packet) {  
    obj = &packet{}  
    obj.loaderData = func(icontains int) (err  
error) {  
        obj.targetData, err =  
compute_arraybyte(int64(icontains))  
        return  
    }  
    return  
}
```

❖ Pour spécialiser notre classe :

```
func NewPacketRandom() (obj *packet) {  
    obj = &packet{}  
    obj.loaderData = func(icount int) (err error) {  
        r := rand.New(rand.NewSource(99))  
        obj.targetData = make([]byte, icount)  
        r.Read(obj.targetData)  
        return  
    }  
    return  
}
```



Comment faire un objet qui peut être enrichi? (un ersatz d'héritage)

- ❖ Il doit suivre un contrat d'interface
 - ❖ → l'objet enrichi 'se comporte comme'
- ❖ Un champ qui contient une méthode modifiable
- ❖ Un constructeur qui réalise la modification – et évidemment, valide le contrat d'interface



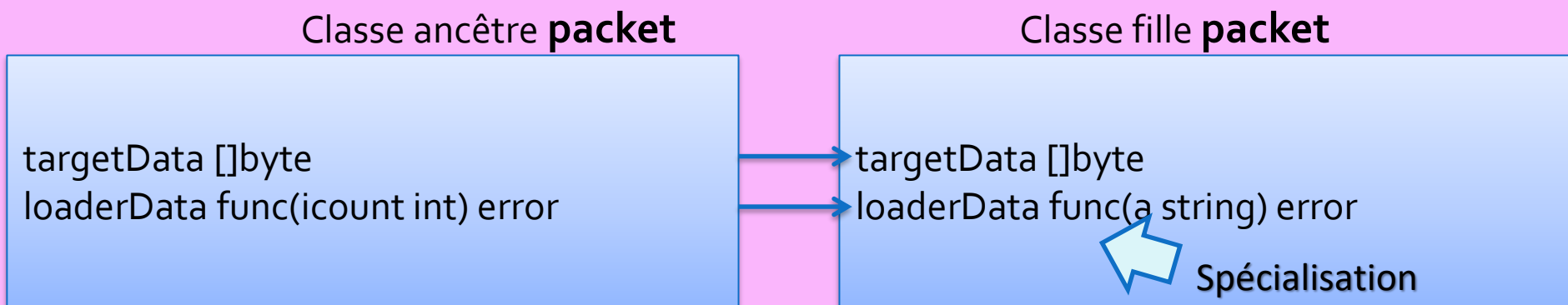
On veut enrichir → type dérivé par composition

```
type packet struct {  
    targetData []byte  
    loaderData func(icontains int) error  
}  
  
func NewPacket() (obj *packet) {  
    obj = &packet{}  
    obj.loaderData = func(icontains int) (err error) {  
        obj.targetData, err = compute_arraybyte(int64(icontains))  
        return  
    }  
    return  
}
```



```
func (obj *packet) EraseAbove64() {  
    for i, a := range obj.targetData {  
        if a > 64 {  
            obj.targetData[i] = 0  
        }  
    }  
    return  
}
```

La méthode **EraseAbove64** est commune aux 2 objets



- ❖ On veut hériter pour améliorer le type – en ajoutant une nouvelle fonction.

```
type packetBetter struct {
    *packet
    packetLen func() int
}

func NewPacket2() (obj *packetBetter) {
    obj = &packetBetter{packet: NewPacket()}
    //obj.packet = NewPacket()
    obj.packetLen = func() int {
        return len(obj.targetData)
    }
    return
}
```

Encapsulation
*packet



Classe **packetBetter**

targetData []byte
loaderData func(a string) error
packetLen() int

- ❖ Go ne propose pas un vrai héritage – par contre, il propose un contrat d'interface

```
type greatPacket interface {  
    CreatePacket()  
    EraseAbove64()  
}
```

L'idée est que se définissent comme greatPacket, les objets qui respectent ce contrat d'interface.



```
func (obj *packet) CreatePacket() {  
    if obj == nil {  
        obj = NewPacket()  
    }  
}
```

```
func (obj *packetBetter) CreatePacket() {  
    if obj == nil {  
        obj = NewPacket2()  
    }  
}
```

```
type packetBetter struct {  
    *packet  
    packetLen func() int  
}
```

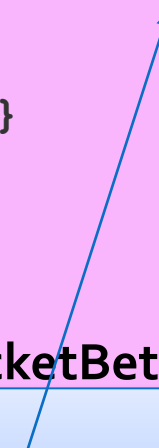
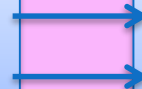
Doit-on créer une fonction EraseAbove64() ?

Classe **packet**

EraseAbove64()
CreatePacket() //newPacket

Classe **packetBetter**

EraseAbove64() //hérite de packet
CreatePacket() //newPacket2

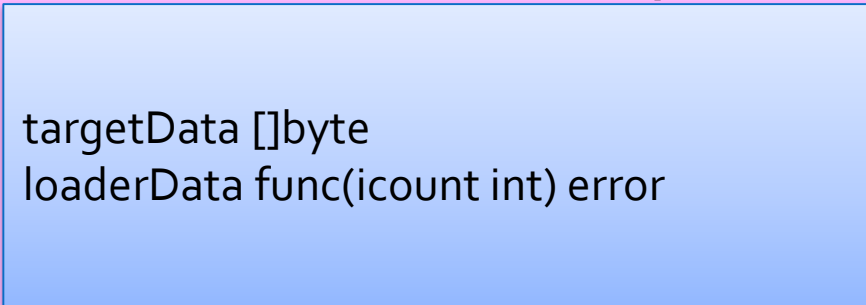


Comment fait-on de l'héritage en Go?

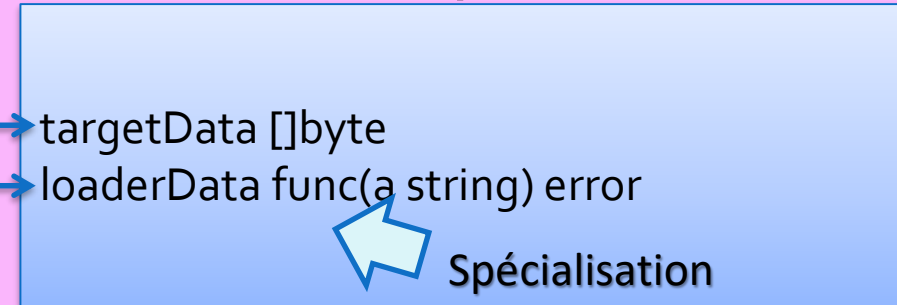
Héritage

Champs

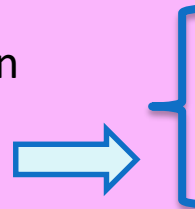
Classe ancêtre **packet**



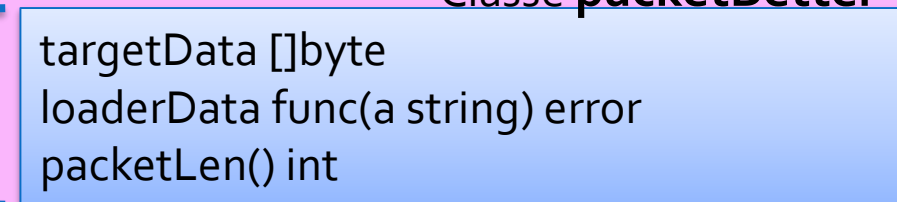
Classe fille **packet**



Encapsulation
*packet

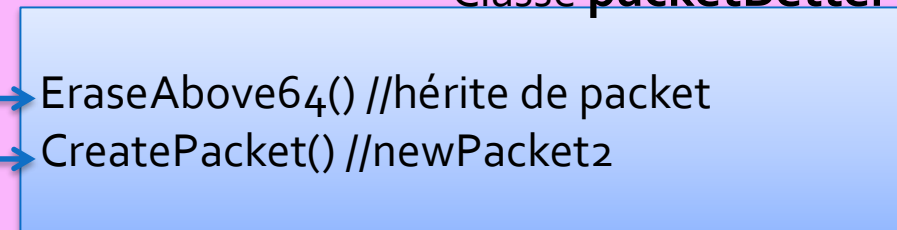
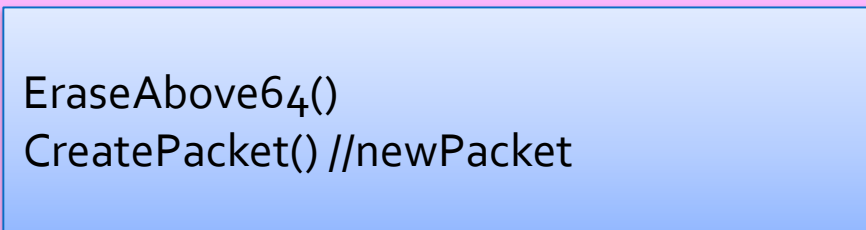


Classe **packetBetter**



Méthodes publiques (par le type **greatPacket interface{}**)

Classe **packetBetter**



Sens de recherche des méthodes

<https://play.golang.org/p/QDgpxMTIaWw>

❖ On veut savoir si une classe respecte le contrat d'interface

```
var _ greatPacket = (*packet) (nil)
var _ greatPacket = (*packetBetter) (nil)
```



Langage de classe sans héritage

- ❖ Un concept classique : l'usine à composants
 - ❖ Une classe virtuelle ancêtre (=usine)
 - ❖ Des classes composants qui héritent
- ❖ Go implémente un type "interface" qui est un contrat

```
type DNA interface {  
    initID()  
    DecritFonction() string }
```

```
type gene struct {  
    typeName string }  
//The DNA struct implements the ID() function  
func (g *gene) initID() {  
    g.typeName = " DNA inconnu " }  
//The DNA struct implements the DecritFonction() function  
func (sv *gene) DecritFonction() string {  
    return "Fonction " + sv.typeName + " : code une protéine " }
```



La classe "usine à composants"

```
type mRNA struct {
    typeName string }
//The DNA struct implements the ID() function
func (g *mRNA) initID() {
    g.typeName = " Messenger RNA " }
//The DNA struct implements the DecritFonction() function
func (sv *mRNA) DecritFonction() string{
    return "Fonction " + sv.typeName + " : code un ARN messenger " }

type CDS struct {
    typeName string }
//The DNA struct implements the ID() function
func (g *CDS) initID() {
    g.typeName = " Coding Data Sequence " }
//The DNA struct implements the DecritFonction() function
func (sv *CDS) DecritFonction() string {
    return "Fonction " + sv.typeName + " : code une region traduite " }
```



La classe "usine à composants"

```
const (  
  t_gene = iota  
  t_mRNA  
  t_CDS )  
func InstancieDNA(t int) (DNA, error) {  
  switch t {  
  case t_gene:      return new(gene), nil  
  case t_mRNA:      return new(mRNA), nil  
  case t_CDS:       return new(CDS), nil  
  default:          return nil, errors.New("Invalid DNA Type")  
  }  
}
```



La classe "usine à composants"

```
dna,_ := InstancieDNA(t_gene)
```

```
dna.initID()
```

```
fmt.Printf("%s\n",dna.DecritFonction())
```

Résultat :

›Fonction : code une protéine

➤ Le principe de Go : `interface{}` = se comporte comme

<https://play.golang.org/p/kWrnpnTa2nL>

❖ Comment occulter une interface?

→ il suffit d'avoir un comportement de type façade

```
type Facade interface {  
    Public() interface{}  
}  
  
func Public(o interface{}) interface{} {  
    if p, ok := o.(Facade); ok { ← L'objet possède une méthode "façade"  
        return p.Public()  
    }  
    return o  
}
```



❖ Vérifier qu'une classe est complète

```
var gene1 DNA = (*s_gene)(nil)
```

```
.\main.go:110: cannot use (*s_gene)(nil) (type *s_gene)  
as type DNA in assignment:
```

```
*s_gene does not implement DNA (missing  
DecritFonction method)
```

Ce truc ne coute rien et permet de vérifier la
validité des classes



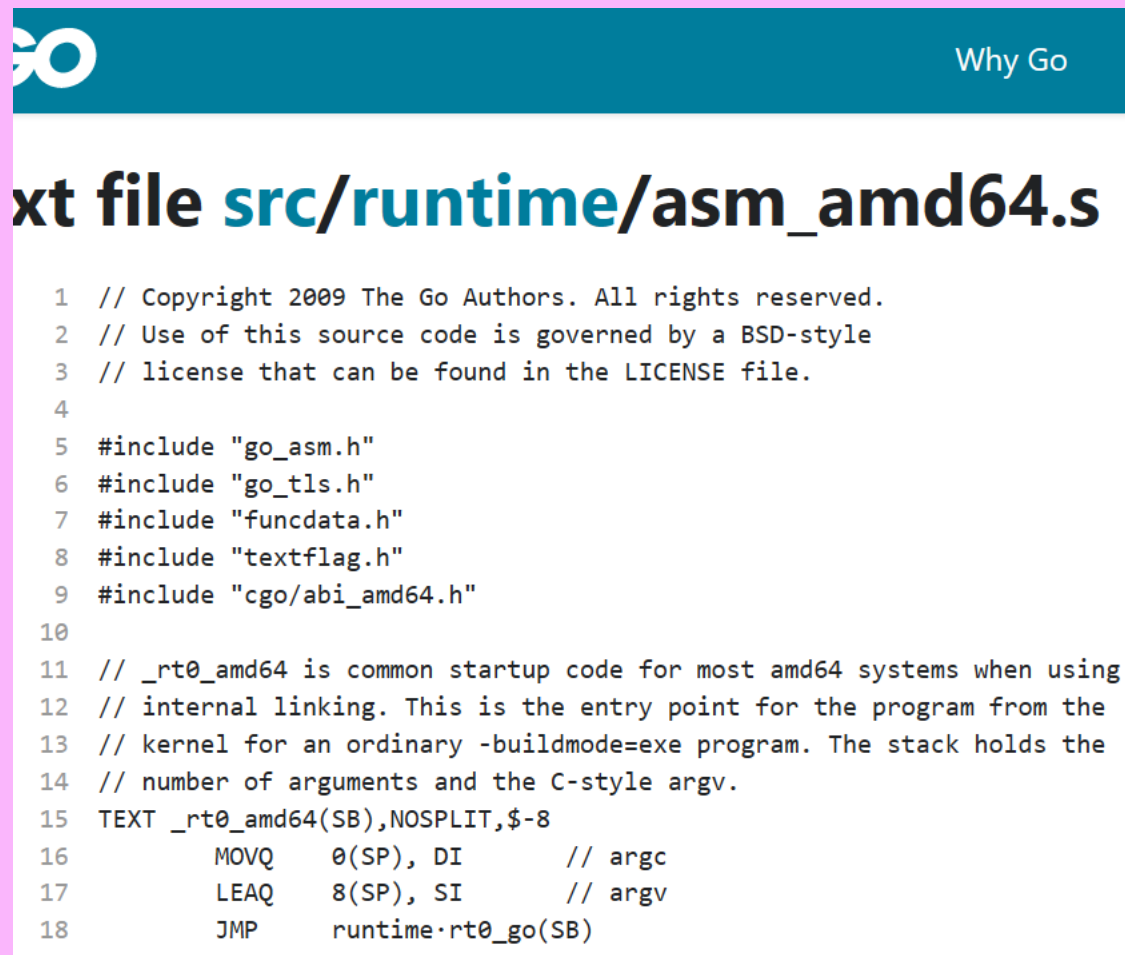
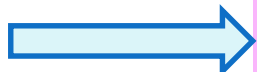
- ❖ Le langage Go permet de faire des appels ASM sans utiliser un outil spécifique.
- ❖ Le «problème» - l'assembleur du langage Go est celui de Plan9 (...68020...) - différent de l'assembleur Intel/ATT
- ❖ Plusieurs tactiques pour obtenir un code assembleur.



- ❖ L'assembleur est installé avec le compilateur Golang, compatible avec de nombreux processeurs
- ❖ Règle du suffix pour code assembleur : nom_amd64.s → version amd64
nom_arm64.s → version arm, 64bits
- ❖ Exemple : <https://go.dev/src/runtime/>

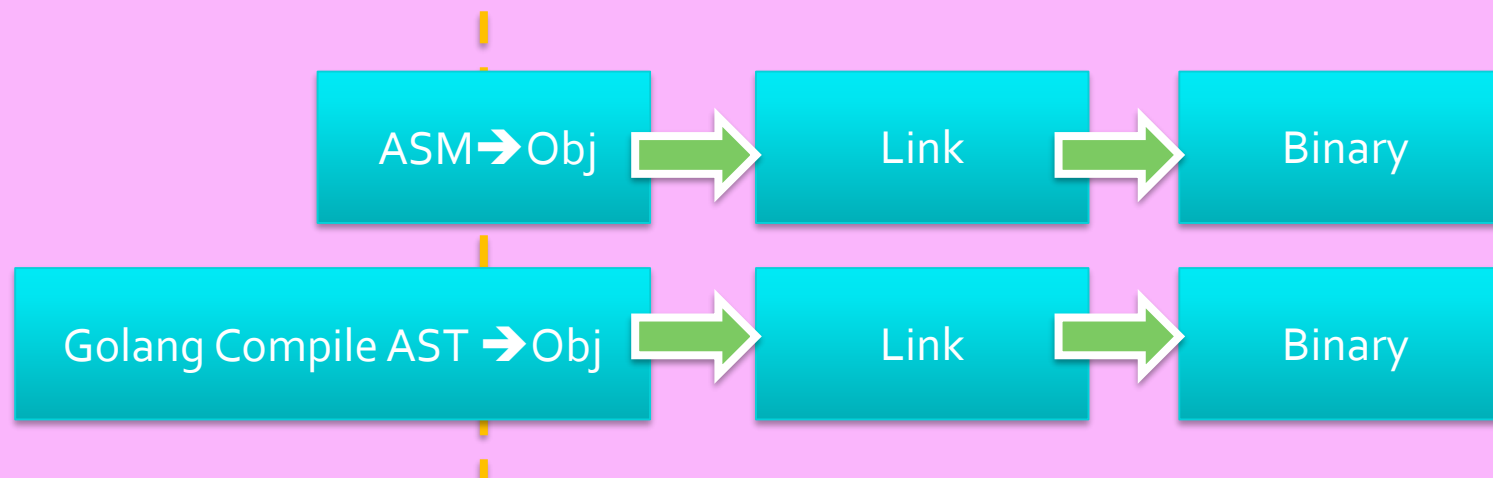


asm_386.s
asm_amd64.s
asm_arm.s
asm_arm64.s
asm_mips64x.s
asm_mipsx.s
asm_ppc64x.h
asm_ppc64x.s
asm_riscv64.s
asm_s390x.s
asm_wasm.s
atomic_arm64.s
atomic_mips64x.s
atomic_mipsx.s
atomic_pointer.go
atomic_ppc64x.s
atomic_riscv64.s



L'assembleur du langage Go

❖ Le système Plan9 améliore l'étape de compilation.



Évolution du loader de Plan9 qui sait optimiser les accès

❖ Le format Obj est plus «évolué» que ELF/COFF



A Manual for the Plan 9 assembler

Rob Pike

rob@plan9.bell-labs.com

- ❖ 1986: Rob Pike écrit un assembleur pour Plan9
- ❖ <https://9p.io/sys/doc/asm.html>
- ❖ Le flot de donnée est de gauche à droite
- ❖ L'instruction connaît la largeur du mot (Q, L)
- ❖ L'objectif de cet assembleur est la portabilité !
- ❖ <https://talks.golang.org/2016/asm.slide#1>
- ❖ Aperçu de l'assembleur :
 - ❖ FP: Frame pointer: arguments and locals.
 - ❖ PC: Program counter: jumps and branches.
 - ❖ SB: Static base pointer: global symbols.
 - ❖ SP: Stack pointer: top of stack.

Un exemple simple : la somme

```
func Sum(a []uint64) uint64 {  
    var sum uint64  
    for i := 0; i < len(a); i++ {  
        sum += a[i]  
    }  
    return sum  
}
```

<https://godbolt.org/z/YKsW9P61T> : le code avec utilisation



Traduction du code Go en ASM Go

amd64 gc 1.16		✓	Compiler options...
A ▾ ⚙ Output... ▾ 🔍 Filter... ▾ 📖 Libraries + Add new... ▾ 🖌 Add			
1	TEXT	"".Sum(SB), NOSPLIT ABIInternal, \$0-32	
2	FUNCDATA	\$0, gcllocals·1a65e721a2ccc325b	
3	FUNCDATA	\$1, gcllocals·69c1753bd5f81501d	
4	MOVQ	"".a+16(SP), AX	
5	MOVQ	"".a+8(SP), CX	
6	XORL	DX, DX	
7	XORL	BX, BX	
8	JMP	Sum_pc32	
9	Sum_pc16:		
10	LEAQ	1(DX), SI	
11	MOVQ	(CX)(DX*8), DI	
12	ADDQ	DI, BX	
13	MOVQ	SI, DX	
14	NOP		
15	Sum_pc32:		
16	CMPQ	DX, AX	
17	JLT	Sum_pc16	
18	MOVQ	BX, "".~r1+32(SP)	
19	RET		

SP+16 → AX → longueur

SP+8 → CX → pointeur s

DX : contient l'avancem

BX : la somme // le résult

LEA adresse DX+1 dans

CX+[DX*8] → pointeur s

Addition contenu vers B

Déplacement mémoire

Positionnement résult

NOSPLIT= 4
SP = pseudo SP, Plang
symbol+offset(SP)

SP+16 → AX → longueur
SP+8 → CX → pointeur sur contenu
DX : contient l'avancement
BX : la somme // le résultat

LEA adresse DX+1 dans SI → taille inc
CX+[DX*8] → pointeur sur contenu avec incrément
Addition contenu vers BX
Déplacement mémoire

Positionnement résultat sur la pile

Le produit scalaire – entier32, sans dépassement

→ en Golang

```
func DotProduct(a []int32, b []int32, N int32) (sum int32) {  
    //N := len(a)  
    for i := int32(0); i < N; i++ {  
        sum += a[i] * b[i]  
    }  
    return  
}
```

→ en C

```
#include <stdint.h>  
void dp_int32(int32_t *a, int32_t *b, int32_t *len, int32_t *res)  
{  
    int32_t N = *len;  
    int32_t reslocal = 0;  
    for(int32_t i = 0; i < N; i++) {  
        reslocal = reslocal + a[i]*b[i];  
    }  
    *res = reslocal; return ;  
}
```



- ❖ <https://godbolt.org/z/K7747xGEc>
- ❖ `-Os -mno-red-zone -mstackrealign -mllvm -inline-threshold=1000 -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti`
- ❖ `-Os` → code compact
- ❖ → il existe une possibilité de génération de code ASM automatique vers Golang avec :



Résultat de Clang

C source #1 X

A Save/Load + Add new... Vim C

```
1 // Type your code here, or load an example.
2 #include <stdint.h>
3
4 void dp_int32(int32_t *a, int32_t *b, int32_t *len, int32_t *res) {
5     int32_t N = *len;
6     int32_t reslocal = 0;
7
8     for(int32_t i = 0; i < N; i++) {
9         reslocal = reslocal + a[i]*b[i];
10    }
11    *res = reslocal;
12    return;
13 }
```

x86-64 clang 13.0.0 (C, Editor #1, Compiler #1) X

x86-64 clang 13.0.0 -Os -mno-red-zo

A Output... Filter... Libraries + Add new...

```
1 dp_int32:                                # @d
2     push    rbp
3     mov     rbp, rsp
4     and     rsp, -8
5     mov     r8d, dword ptr [rdx]
6     test    r8d, r8d
7     jle     .LBB0_1
8     xor     eax, eax
9     xor     r9d, r9d
10 .LBB0_4:                                # =>
11     mov     edx, dword ptr [rsi + 4*rax]
12     imul    edx, dword ptr [rdi + 4*rax]
13     add     r9d, edx
14     inc     rax
15     cmp     r8, rax
16     jne     .LBB0_4
17     jmp     .LBB0_2
18 .LBB0_1:
19     xor     r9d, r9d
20 .LBB0_2:
21     mov     dword ptr [rcx], r9d
22     mov     rsp, rbp
23     pop     rbp
24     ret
```

Output (0/0) x86-64 clang 13.0.0 - 405ms (188688) ~393 lines

- ❖ Générer un code ASM avec Clang – qui sait faire de l'autovectorisation
- ❖ Convertir le code de l'ASM Gnu vers l'ASM Go ➔ outil c2goasm
 - ❖ git clone <https://github.com/minio/asm2plan9s>
 - ❖ git clone <https://github.com/klauspost/asmfmt>
 - ❖ git clone <https://github.com/minio/c2goasm>
 - ❖ Mettre les go.mod, mettre les binaires dans le path
 - ❖ \$ c2goasm -a -f ...

❖ Utilisation de c2goasm

- ❖ 1/obtenir un code C qui compile avec Clang
- ❖ `clang -S -mavx2 -masm=intel -mllvm -force-vector-width=4 -fno-asynchronous-unwind-tables -fno-exceptions -fno-omit-frame-pointer -fno-rtti -c -O3 dotproduct.c`
- ❖ 2/fabriquer un code en Go pour avoir le header
- ❖ `_dp_int32(int32_t *a, int32_t *b, int32_t *len, int32_t *res)`

```
#include <stdint.h>

void dp_int32(int32_t *a, int32_t *b, int32_t
    *len, int32_t *res) {
    int32_t N = *len;
    int32_t reslocal = 0;
    for(int32_t i = 0; i < N; i++) {
        reslocal = reslocal + a[i]*b[i];}
    *res = reslocal; return ; }
```



```
>dotproduct_amd64.go
package Dotproduct
import ( "unsafe")
//go:noescape
func _dp_int32(a, b, N, res unsafe.Pointer)
func DotProduct(a []int32, b []int32, N int32) int32 {
    var res int32
    _dp_int32(unsafe.Pointer(&a[0]),unsafe.Pointer(&b[0]),
    unsafe.Pointer(&N),unsafe.Pointer(&res))
    return res
}
```

- Ajout du '_' en préfix
- Utilisation de unsafe
- go:noescape → pas d'analyse d'échappement



- ❖ `clang -S -mavx2 -masm=intel -mllvm -force-vector-width=4 -fno-asynchronous-unwind-tables -fno-exceptions -fno-omit-frame-pointer -fno-rtti -c -O3 dotproduct.c`
- ❖ `-O3 -mavx512f -mavx512dq -mavx512bw -mavx512vbmi -mavx512vbmi2 -mavx512vl`
- ❖ Clang a 2 optimiseurs pour l'autovectorisation – on peut agir sur la taille de la vectorisation
- ❖ ➔ génère une version assembleur
- ❖ `./c2go -a -f dotproduct.s dotproduct_amd64.s`
- ❖ Il s'agit de positionner le fichier `dotproduct_amd64.go`



- ❖ Problème : le code généré ne fonctionne pas sous windows.

→ il y a du code «en binaire»

...

```
LONG $0x646ffac5; WORD $0x1086// vmovdqu xmm4, oword [rsi + 4*rax]
LONG $0x6c6ffac5; WORD $0x2086// vmovdqu xmm5, oword [rsi + 4*rax + 16]
LONG $0x746ffac5; WORD $0x3086 // vmovdqu xmm6, oword [rsi + 4*rax + 32]
LONG $0x7c6ffac5; WORD $0x4086// vmovdqu xmm7, oword [rsi + 4*rax + 48]
```

- ❖ La voie du méta assembleur:

- ❖ PeachPy <https://github.com/Maratyszczka/PeachPy> - 1815 étoiles

- ❖ → assembleur codé en Python avec un métalangage

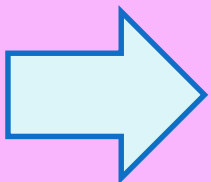
- ❖ Avo <https://github.com/mmcloughlin/avo> - 2465 étoiles

- ❖ → assembleur codé en Go avec un métalangage



Exemple de meta assembleur (PeachPy)

Meta ASM



```
from peachpy.x86_64 import *

# Lets write a function float DotProduct(const float* x, const float* y)

# If you want maximum cross-platform compatibility, arguments must have names
x = Argument(ptr(const_float_), name="x")
# If name is not specified, it is auto-detected
y = Argument(ptr(const_float_))

# Everything inside the `with` statement is function body
with Function("DotProduct", (x, y), float_,
    # Enable instructions up to SSE4.2
    # PeachPy will report error if you accidentally use a newer instruction
    target=uarch.default + isa.sse4_2):

    # Request two 64-bit general-purpose registers. No need to specify exact names.
    reg_x, reg_y = GeneralPurposeRegister64(), GeneralPurposeRegister64()

    # This is a cross-platform way to load arguments. PeachPy will map it to something proper later.
    LOAD.ARGUMENT(reg_x, x)
    LOAD.ARGUMENT(reg_y, y)

    # Also request a virtual 128-bit SIMD register...
    xmm_x = XMMRegister()
    # ...and fill it with data
    MOVAPS(xmm_x, [reg_x])
    # It is fine to mix virtual and physical (xmm0-xmm15) registers in the same code
    MOVAPS(xmm2, [reg_y])

    # Execute dot product instruction, put result into xmm_x
    DPPS(xmm_x, xmm2, 0xF1)

    # This is a cross-platform way to return results. PeachPy will take care of ABI specifics.
    RETURN(xmm_x)
```



Exemple de meta assembleur (AVO)

```
package main

import . "github.com/mmcloughlin/avo/build"

func main() {
    TEXT("Add", NOSPLIT, "func(x, y uint64) uint64")
    Doc("Add adds x and y.")
    x := Load(Param("x"), GP64())
    y := Load(Param("y"), GP64())
    ADDQ(x, y)
    Store(y, ReturnIndex(0))
    RET()
    Generate()
}
```

```
#include "textflag.h"

// func Add(x uint64, y uint64) uint64
TEXT ·Add(SB), NOSPLIT, $0-24
    MOVQ x+0(FP), AX
    MOVQ y+8(FP), CX
    ADDQ AX, CX
    MOVQ CX, ret+16(FP)
    RET
```

go:generate go run asm.go -out add.s -stubs stub.go



Alternative : fabrication du code 'à la main'

```
//func _dp_int32(a *int32, b *int32, gN *int32, res *int32)
TEXT    ·_dp_int32(SB),4, $0-32
        MOVQ a+0(FP), DI
        MOVQ b+8(FP), SI
        MOVQ gN+16(FP), DX
        MOVQ res+24(FP), CX
        MOVQ (DX),R8                // mov    r8d, dword [rdx]
        CMPQ R8,$0                  // test   r8d, r8d
        JLE  LBB0_1
        XORQ AX, AX                  // xor    eax, eax
        XORQ R9, R9                  // xor    r9d, r9d
LBB0_4:
        MOVQ (SI)(AX*4),DX           // mov    edx, dword [rsi + 4*rax]
        IMULQ (DI)(AX*4),DX          // imul    edx, dword [rdi + 4*rax]
        ADDQ DX,R9                   // add    r9d, edx
        INCQ AX                      // inc    rax
        CMPQ AX, R8                  // cmp    r8, rax
        JNE  LBB0_4
        JMP  LBB0_2
LBB0_1:
        XORQ R9, R9                  // xor    r9d, r9d
LBB0_2:
        MOVQ R9,(CX)                 // mov    dword [rcx], r9d
        RET
```

Vectorisation AVX du produit scalaire

8 int32 : a[]

8 int32 : b[]



VMOVDQU



VMOVDQU



Y1: 256 bits

Y2 : 256 bits



VPMULLD

Vxxx : Vectorisé



Y3



VPADDD

Y4



VMOVDQU → pile (32 bytes)

AVX : 2011

Attention : la taille a[]/b[] est un multiple de 8

Avec AVX512 - registres Z : 512 bits, VZERoupper



Le code vectorisé

```
TEXT _dpavx_int32(SB),4, $32-32
```

```
MOVQ a+0(FP), DI
```

```
MOVQ b+8(FP), SI
```

```
MOVQ gN+16(FP), CX
```

```
MOVQ res+24(FP), DX
```

```
MOVQ (CX),R8 // value of gN
```

```
MOVQ DX,R9 // return address
```

```
VPXOR Y4, Y4, Y4
```

```
XORQ AX,AX
```

```
start:
```

```
VMOVDQU (SI), Y1
```

```
VMOVDQU (DI), Y2
```

```
VPMULLD Y1, Y2, Y3
```

```
VPADDD Y3, Y4, Y4
```

```
ADDQ $32, SI
```

```
ADDQ $32, DI
```

```
ADDQ $8, AX
```

```
CMPL AX, R8
```

```
JNE start
```

```
VMOVDQU Y4, d0-32(SP) // vector result to stack
```

```
LEAQ d0-32(SP), BX
```

```
MOVQ $8, CX // array length 8 int32
```

```
XORQ SI, SI // clean SI
```

```
redux: //8 bytes => 8 int32 reduction to 1 int32
```

```
ADDL (BX), SI
```

```
ADDQ $4, BX // handle 4 int32
```

```
DECQ CX
```

```
JNZ redux
```

```
MOVL SI,(R9)
```

```
RET
```

`_dpavx_int32(a *int32, b *int32, gN *int32, res *int32)`

Attention : la taille a[]/b[] est un multiple de 8

- ❖ Cible gcc/clang, SSE4.1
- ❖ Les intrinsics sont «des ajouts» au compilateur
- ❖ <https://godbolt.org/z/aW6dhrxqE>
- ❖ Utilisation du format m128 bits (XMM) // 4 int32
 - _mm_setzero_si128 : mise à zéro
 - _mm_load_si128 : chargement //, hint sur le cache (nocache)
 - _mm_mullo_epi32 : multiplication int32, cast int32
 - _mm_add_epi32 : addition int32
 - _mm_srli_si128 : décalage à droite
 - _mm_cvtsi128_si32 : récupère la partie basse int32



Attention : la taille a[]/b[] est un multiple de 4

```
#include <smmintrin.h>

// the actual loop body itself is still fine for runtime-variable N
[[gnu::target("sse4.1")]] int32_t dp(int32_t a[], int32_t b[], int32_t N)
{
    int32_t sum = 0;
    __m128i temp_sum = _mm_setzero_si128();
    for(int i=0;i<N;i=i+4){ // 4 int32 processed
        //Load the 4 values from x
        __m128i temp_1 = _mm_load_si128(reinterpret_cast<__m128i*>(&a[i])); // add cast
        //Load the 4 values from y
        __m128i temp_2 = _mm_load_si128(reinterpret_cast<__m128i*>(&b[i])); // add cast
        //Multiply x[0] and y[0], x[1] and y[1] etc
        __m128i temp_products = _mm_mullo_epi32(temp_1, temp_2);
        //Sum temp_sum
        temp_sum = _mm_add_epi32(temp_sum, temp_products);
    }
    // take horizontal sum of temp_sum - reduction
    temp_sum = _mm_add_epi32(temp_sum, _mm_srli_si128(temp_sum, 8));
    temp_sum = _mm_add_epi32(temp_sum, _mm_srli_si128(temp_sum, 4));
    sum = _mm_cvtsi128_si32(temp_sum); // convert
    // assign after the loop so compiler knows it doesn't alias
    return sum;
}
```

Point d'optimisations possibles

- ❖ GOMAXPROCS = donne la limite du nombre de processus, équivaut au nombre de cœur
- ❖ `runtime.GOMAXPROCS(n int) int` → positionne le nombre de processus.
- ❖ GOGC = action du ramasse miette
- ❖ GOGC = 100 par défaut, décrit la fraction de mémoire qui est considérée pour déclencher le ramasse miette. 100 = doublement (100%)
- ❖ → peut être mis à OFF, ou à des valeurs élevées
25000 → rend le ramasse miette agressif
- ❖ `runtime.GC()` : invoque le GC, bloque le programme