



# Génie Logiciel pour le Calcul Scientifique



#9

03/02/2025

jean-michel.batto@cea.fr

cea

[https://gogs.eldarsoft.com/M2\\_IHPS](https://gogs.eldarsoft.com/M2_IHPS)



INTERACTIF

❖ Vous allez faire fonctionner 3 codes « élémentaires » de la dev à la prod:

❖ sur les nœuds de dev : mpihead, mpinode1, mpinode2, mpinode3

❖ → la commande mpirun

❖ sur les nœuds de prod : en mode interactif avec salloc et srun

Vous allez tester dans les 2 environnements. On doit faire un docker attach pour accéder au nœud de contrôle. slurmctld

```
docker exec -it slurmctld /bin/bash
```

1/exploration du paramétrage dynamique des nœuds dans XMP (histogramme)

4 nœuds en dev, puis 2 nœuds en prod

2/openmp : entre la dev et la prod. La commande mpirun permet de propager une variable aux nœuds

3/protection-reprise : un exemple de code

INTERACTIF

❖ 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 yml_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

- ❖ En utilisant le répertoire de ce CM9 (GLCS-CM9-2024/GLCS-CM5-2024)
- ❖ Cela permet de conserver le paramétrage des volumes docker du CM5
- ❖ Démarrage de docker
  - ❖ `docker compose up -d`



INTERACTIF

- ❖ Reprenons le code
- ❖ <https://gogs.eldarsoft.com/jmbatto/GLCS-CM6-TDXMP>
- ❖ //avec cette commande on a le mode « dynamique »
- ❖ `#pragma xmp nodes p(*)`
- ❖ Vérifier le résultat pour 4 nœuds (head, node1, node2, node3).
- ❖ `Makefile run: histo`  
`mpirun --mca orte_base_help_aggregate 0 -host mpihead,mpinode1,mpinode2,mpinode3 -n 4 histo`  
→ vérifier si cela fonctionne pour 4 nœuds de calcul
- ❖ N'oubliez pas la réduction sur le résultat



Comment faire la réduction entre les data\_out associés aux « partitions » XMP ?

Le code doit être indépendant du nombre de partition.

```
for (int i = 0; i < 20; i++) {  
    int tmp = data_out[i];  
    #pragma xmp reduction(+ : tmp)  
    data_out[i] = tmp;  
}
```



INTERACTIF

passer mpiuser 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
```

```
//à la place de mpirun on lance la commande histo
```

```
srun histo
```

**Que voyons nous ?**

```
//on regarde l'état de la queue
```

```
squeue
```

```
//permet d'avoir les informations sur les jobs
```

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

INTERACTIF

- ❖ Commande neofetch
- ❖ commande shell pour avoir une description du nœud

```


root@c1:/# neofetch
  __,met$$$$$gg.
 ,g$$$$$$$$$$$$$$P.
,g$$$P"      ""Y$$.".
,$$P'          `$$$'.
',$$P          ,ggs.   `$$b:
`d$$'          ,P"'.   $$$
 $$P           d$'     ,   $$P
 $$:           $$-    ,d$$'
 $$;           Y$b._   _dP'
 Y$$           `."Y$$$$P"'
`$$b          "._._
`Y$$
`Y$$.
`$$b.
`Y$$b.
`"Y$b._
`"

```

```

root@c1
-----
OS: Debian GNU/Linux 12 (bookworm) on Windows 10 x86_64
Kernel: 5.15.167.4-microsoft-standard-WSL2
Uptime: 4 hours, 29 mins
Packages: 625 (dpkg)
Shell: bash 5.2.15
CPU: Intel i5-8250U (7) @ 1.799GHz
GPU: 8b47:00:00.0 Microsoft Corporation Basic Render Driver
Memory: 1048MiB / 3860MiB

```







INTERACTIF

❖ `scontrol show node`

❖ Ou alors

❖ `sinfo -Ne1`

Pour voir l'historique de la consommation

❖ `sacct`



INTERACTIF

❖ [http://gogs.eldarsoft.com/M2\\_IHPS/GLCS-CM9-TDXMP.git](http://gogs.eldarsoft.com/M2_IHPS/GLCS-CM9-TDXMP.git)

❖ On veut évaluer OpenMP dans notre image Docker

❖ `salloc --time=05:00 -N 2`

❖ `srun para_hello_world`

Comparez le test avec un `mpirun` (cf `Makefile`) sur les nœuds head

❖ N'oubliez pas de faire un `sacct` à la fin de la session



INTERACTIF

❖ On veut évaluer la protection-reprise (en anglais checkpoint)

❖ `git clone`

[http://gogs.eldarsoft.com/M2\\_IHPS/glcs\\_slurm.git](http://gogs.eldarsoft.com/M2_IHPS/glcs_slurm.git)

On veut travailler sur la branche `supportCM8`

```
git checkout supportCM8
```

```
make run → pour vérifier le fonctionnement
```

```
salloc --time=05:00 -N 2
```

```
srun
```

```
scancel
```

→ que voit-on au niveau de la protection/reprise ?



❖ Quelle modification introduire pour permettre un bon fonctionnement?



- ❖ EB : écrire un programme qui fait le tri d'un fichier `ascii` avec 2 colonnes – séparateur tabulation – trier la valeur de la première colonne qui est un nom de 5 caractères. Le résultat est un fichier.
- ❖ SFD : - en C
  - ❖ (A) Ouvrir le fichier
  - ❖ (B) Lire le fichier et convertir les valeurs texte (avec vérification si c'est un nom de 5 caractères) – fermer le fichier
  - ❖ (C) Faire un tri
  - ❖ (D) Ecrire le résultat avec une tabulation

Etape	Estimation LOC	JH
A	10	1 JH
B	20	
C	5	
D	15	

1 JH : écrire l'EB / SFD / Recette

1 JH : écrire le code et test (50 LOC)

Etape	Estimation LOC	JH
A	10	2 JH
B1	20	
B2	5	
C1	30	
C2	5	
D	15	

1 JH : écrire l'EB / SFD / Recette

2 JH : écrire le code et test (85 LOC > 1J travail)



- ❖ nvidia/llama-3.1-nemotron-70b-instruct
- ❖ Peux-tu estimer le temps de développement en heures pour écrire un programme qui fait le tri d'un fichier ascii avec 2 colonnes – séparateur tabulation – trier la valeur de la première colonne qui est un nom de 5 caractères. Le résultat est un fichier. Le programme est écrit en C. La réponse est concise en une phrase.
- ❖ **Temps de Développement Estimé** : 14 heures (conception compris), réparties entre lecture/traitement des données (4h), tri (3h), écriture du fichier de sortie (1h) et gestion des erreurs/tests (5h).
- ❖ → 14 heures en C = 2JH
- ❖ → 8 heures en Node.JS
- ❖ → 10 heures en Golang



## ❖ Build

- ❖ SFD - Recette

- ❖ Livraisons Dev / Préprod / Prod

## ~~❖ Run~~

- ~~❖ PRA~~

## ~~❖ MCO~~

## ~~❖ MCS~~

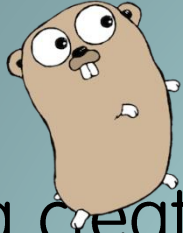


# Projet cuicui - chiffrage

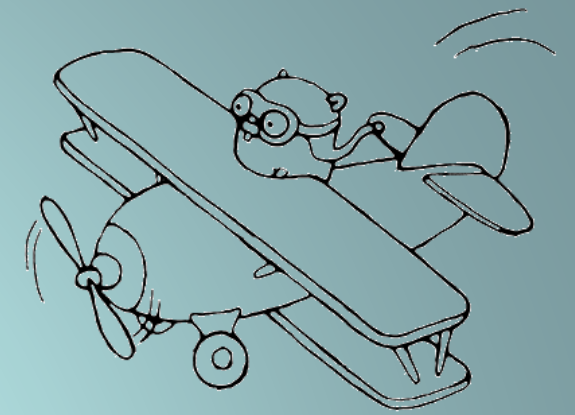
Rubrique	fonction	difficulté	
gestion utilisateur	userstate	8	
	user	11	
	inscription	4	
	abonnement	5	
	suspension de compte	8	
	audit compte		
	administrateur	5	
	compte maitre	8	
	page utilisateur	6	
Total utilisateur			
gestion message	page message	24	
	fil de message	message	3
		message decorator	1
		message thread	3
		image message	3
		text message	3
		admin message decorator	3
		opérations utilisateur	8
		page administrateur	opérations administrateur
notifications	7		
6 fonctions à valider avec le client	6		
Recette			

The Go programming language and environment, avril 2022 <http://dx.doi.org/10.1145/3488716>

- ❖ 2007 : Robert Griesemer, Rob Pike, Ken Thompson démarrent le projet d'un nouveau langage (Pike et Thomson sont co-auteurs du B –avant le C- et de l'UTF-8)
- ❖ 2009 : projet opensource publique
- ❖ mars 2012 : Go 1.0
- ❖ 12 janvier 2016 : une nouvelle incroyable !  
**"All Systems z are Go: IBM ports Google language to mainframes"**
- ❖ Docker (2013), Kubernetes (2014), Prometheus (2016) : en Go



# Le langage Go



- La création des langages est une activité continue  
50 nouveaux langages tous 10 ans

(source [https://en.wikipedia.org/wiki/Timeline\\_of\\_programming\\_languages](https://en.wikipedia.org/wiki/Timeline_of_programming_languages))

Un langage est une représentation de l'espace du problème

Un langage n'est pas universel (tour de Babel) → idée de spécialisation (en 1996, 500 langages spécialisés)

(source <http://www.cs.bsu.edu/homepages/dmz/cs697/langtbl.htm>)

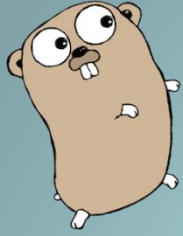
Il faut environ 7 ans pour qu'un langage soit populaire

PHP4.2 (2002) → 2009

Python2.3 (2003) → 2010

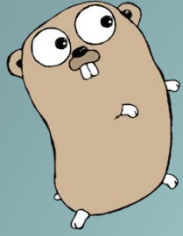
Go1.x(2009) → 2016 // langage de l'année sur tiobe.com

Un langage "à la mode apparait" en fonction des pbs à "la mode"



# Les problèmes de 2025

- Many-cores
  - amd threadripper à 64 cœurs
  - 8 cœurs – 16 threads, 300\$
- SIMD et vectorisation
  - GPU
- Enveloppe thermique <5W
  - IoT
- Cycle de développement en mode “superhéro” (=pas de cycle en V)
  - Agile, Git



# Typologie d'un langage

- Les paradigmes sous jacents au delà du pure langage
  - Tout est liste (lisp, ruby, smalltalk) et rien d'autre
    - Smalltalk conceptualise l'objet à travers une vision organisée
  - Tout est objet (C++) y compris le clavier
    - C++ reprend le C et branche des objets sur tout (header, opérateur, template, ...)
  - Tout est règle/prédicat (SQL, Prolog)
- L'abstraction dans l'abstraction :
  - mettre une VM
  - mettre un ramasse miette
  - mettre des références à la place des pointeurs



❖ break, default, func, interface, select, case, defer, go, map, struct, chan, else, goto, package, switch, const, fallthrough, if, range, type, continue, for, import, return, var

- 2009 : Go 1.0 – avec un compilateur
- **Go : 25 mots clés / pour ANSI C : 32 / pour Brainfuck : 8**
- **Approche CSP → les mêmes pbs que ceux du HPC**
- Langage Opensource
  - Licence du langage : type BSD
- Aujourd'hui 4 compilateurs : llgo (llvm), gcc, gc, tinygo (IoT)
- Processeurs cibles  
ARM, ARM64, x86-32 et AMD64, Mips32, Mips64, s390, PPC64, RiscV, RiscV64



## ❖ Typage et gestion des chaînes de texte

- ❖ Postfixé // gestion de l'utf-8 – Rob Pike & K. Thompson

```
var b rune = 'a'
```

```
var kanji_r rune = '門' // kanji pour dire porte E99680
```

```
var kanji_s string = "\xE9\x96\x80"
```

```
fmt.Printf("%c \n%s et le caractère a, ascii = %+v\n",  
    kanji_r, kanji_s, b)
```

<https://play.golang.org/p/XEVoMmK5mXw>





## ❖ Les variables et les constantes

```
var i = 'a'
```

```
i := 'a'
```

```
const a int = 10
```

```
const (
```

```
    Red = (1 << iota)
```

```
    Green = (1 << iota)
```

```
    Blue, ColorMask (1 << iota, (1 << (iota + 1)) - 1) )
```

→ ramasse-miette

→ référence &r, \*r



## ❖ tableaux

Itérateur

```
var s =[]string{2:"aa","bb"}  
for a,c :=range s  
{    fmt.Printf("%d,%s\n",a,c)    }  
  
0,  
1,  
2,aa  
3,bb
```

## ❖ ... = variadic



❖ Slices → un type plus intéressant

```
var s =[]string{2:"aa","bb"}
```

```
var t=[]string{4:"cc","dd"}
```

```
s=append(s,t...)
```

```
for a,c :=range s {    fmt.Printf("%d,%s\n",a,c)    }
```

```
0,
```

```
1,
```

```
2,aa
```

```
3,bb
```

```
4,
```

```
5,
```

```
6,
```

```
7,
```

```
8,cc
```

```
9,dd
```

<https://play.golang.org/p/jXS5Rn0U8UR>



```
var s =[]string{2:"aa","bb"}  
var t=[]string{4:"cc","dd"}  
s=append(s[2:4],t[4:6]...)  
for a,c :=range s {   fmt.Printf("%d,%s\n",a,c)   }  
0,aa  
1,bb  
2,cc  
3,dd
```

<https://play.golang.org/p/1eyKjt-3lkr>



❖ `make()`

❖ `new()`

`A:=new(int) // *A = 0`

→ retourne un pointeur

`B:=make([int,1]) // B[0]=0`

→ retourne une référence et fait l'allocation

❖ `&LocalVar, &T{...}`

→ retourne une référence

## ❖ Switch

❖ Permet de choisir – plus élégant que l’accumulation de if

```
switch extension {
```

```
  case
```

```
    “.svg”,
```

```
    “.png”,
```

```
    “.pdf”,
```

```
    “.tif”,
```

```
    “.tiff”,
```

```
    “.jpeg”,
```

```
    “.jpg”:
```

```
  return
```

```
  default : fmt.Printf(“inconnu\n”)
```

```
}
```

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



❖ for i:=0; i<10;i++{

❖ for a,c := range {

❖ break

❖ continue (continue l'itération ou va à la condition d'arrêt)

<https://play.golang.org/p/61F0QcH3h4d>

❖ goto

```
var t string =
```

```
"0.61803398874989484820458683436563811772030917980576286  
2135448622705260462818"
```

```
for a,i := range t {
```

```
    fmt.Printf("a %d %c\n",a,i)
```

```
    if i == '9' {
```

```
        goto Fin
```

```
    }
```

```
}
```

```
Fin:
```

```
    fmt.Printf("fin")
```

<https://play.golang.org/p/1J-Hzb0PAga>



# Les mots réservés du langage

- ❖ break
- ❖ default
- ❖ func
- ❖ interface
- ❖ select
- ❖ case
- ❖ defer
- ❖ go
- ❖ map
- ❖ struct
- ❖ chan
- ❖ else
- ❖ goto
- ❖ package
- ❖ switch
- ❖ const
- ❖ fallthrough
- ❖ if
- ❖ range
- ❖ type
- ❖ continue
- ❖ for
- ❖ import
- ❖ return
- ❖ var
- ❖ new
- ❖ make
- ❖ close
- ❖ <-
- ❖ ...





## ❖ Le transtypage

❖ Le langage est contraint

❖ `var x int = 300`

❖ `var y int = 400`

❖ `var mul float32`

❖ `mul = float32(x) * float32(y)`

❖ On peut transtyper un int en byte puis en rune

❖ `rune(byte(val))`

```
type fenetre_2char struct {
    est_ascii bool
    val_ascii rune
}
```

Création d'un objet : struct

```
const t string =
"0.61803398874989484820458683436563811772030917980576286213544862270526
0462818"
```

<https://oeis.org/A001622>



```
func test_struct() {
```

```
    sizeString := len(t) - 2
```

```
    for i := 2; i < sizeString; i = i + 2 {
```

```
        val, err := strconv.Atoi(t[i : i+2])
```

```
        if err == nil && val > 65 && val < 90 {
```

```
            a_rune := []rune(t[i : i+2])
```

```
            a := fenetre_2char{est_ascii: true, val_ascii: a_rune[0]}
```

```
            fmt.Printf("iteration i: %d variable a %+v\n", i, a)
```

```
        }
```

```
    }
```

```
}
```

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

Pb! Quelle est la  
bonne conversion?



[https://fr.wikibooks.org/wiki/Les\\_ASCII\\_de\\_0\\_%C3%A0\\_127/La\\_table\\_ASCII](https://fr.wikibooks.org/wiki/Les_ASCII_de_0_%C3%A0_127/La_table_ASCII)



## ❖ Fonction

```
func (self fenetre_2char) Print() (rune, int) {  
    fmt.Printf("Print() rune %c\n", self.val_ascii)  
    a := byte(self.val_ascii)  
    return self.val_ascii, int(a)  
}
```

→ peut retourner plusieurs valeurs à la fois !

→ le receveur (=self) peut être une variable ou un pointeur sur une variable



- ❖ Mécanisme de finalisation // contrat sur le futur (quand l'objet est détruit)
- ❖ defer = finally en java = local destructeur en C++
- ❖ Permet de faire du code 'propre'
- ❖ Utile pour les DB, les filesystem, les IO (avec un close)



<https://play.golang.org/p/4uo6CiZSINC>

```
func test_struct() (total int) {
    var i int

    defer func() { total = i }()

    sizeString := len(t) - 2
    for i = 2; i < sizeString; i = i + 2 {
        val, err := strconv.Atoi(t[i : i+2])
        fmt.Printf("val %+v\n", val)
        if (err == nil) && (val > 65) && (val < 90) {
            a_rune := rune(byte(val))
            a := fenetre_2char{est_ascii: true, val_ascii: a_rune}
            fmt.Printf("iteration i: %d variable a %+v\n", i, a)
            a.Print()
        }
    }
    return
}
```

Comment faire sans le «defer» ?



```
def sum(k):  
    def helper(n):  
        if n == 0:  
            return 0  
        return n + helper(n-1)  
    return helper(k)  
  
def main():  
    print(sum(3))  
  
if __name__ == '__main__':  
    main() # print 6
```



# La même fonction en Golang

```
package main
import (
    "fmt"
)
func sum(k int) int {
    helper := func(n int) int {
        if n == 0 {
            return 0
        }
        return n + sum(n-1)
    }(k)
    return helper
}
func main() {
    fmt.Println(sum(3))
}
```

<https://play.golang.org/p/ba3O1Ev42yt>



# Ceci n'est pas du GO mais du C

```
var WeekDays = map[string]time.Weekday{"lundi": time.Monday, "monday": time.Monday,  
    "mardi": time.Tuesday, "tuesday": time.Tuesday,  
    "mercredi": time.Wednesday, "wednesday": time.Wednesday,  
    "jeudi": time.Thursday, "thursday": time.Thursday,  
    "vendredi": time.Friday, "friday": time.Friday,  
    "samedi": time.Saturday, "saturday": time.Saturday,  
    "dimanche": time.Sunday, "sunday": time.Sunday}
```

```
func DaysValidation(DaysList *[]string) error {  
    for idx, day := range *DaysList {  
        //      Cleaning input string  
        (*DaysList)[idx] = strings.ToLower(strings.Replace(day, " ", "", -1))  
        //      String verification  
        _, is_valid := WeekDays[(*DaysList)[idx]]  
        if !is_valid {  
            return errors.New("Only English and French days are known")  
        }  
    }  
    //      If all day are well writen, return nil error  
    return nil  
}
```





```
var daysValidation = func() map[string]time.Weekday {
    return map[string]time.Weekday{"lundi": time.Monday,
        "monday": time.Monday,
        "mardi": time.Tuesday,
        "tuesday": time.Tuesday,
        "mercredi": time.Wednesday,
        "wednesday": time.Wednesday,
        "jeudi": time.Thursday,
        "thursday": time.Thursday,
        "vendredi": time.Friday,
        "friday": time.Friday,
        "samedi": time.Saturday,
        "saturday": time.Saturday,
        "dimanche": time.Sunday,
        "sunday": time.Sunday}
}

func printDay(key string) {
    a, err := daysValidation()[key]
    fmt.Println(err)
    fmt.Println(a)
}

func main() {
    printDay("lundi")
    printDay("lunday")
}
```

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



# Ce qui nous reste à voir

- ❖ break
- ❖ default
- ❖ func
- ❖ interface
- ❖ select
- ❖ case
- ❖ defer
- ❖ go
- ❖ map
- ❖ struct
- ❖ chan
- ❖ else
- ❖ goto
- ❖ package
- ❖ switch
- ❖ const
- ❖ fallthrough
- ❖ if
- ❖ range
- ❖ type
- ❖ continue
- ❖ for
- ❖ import
- ❖ return
- ❖ var
- ❖ new
- ❖ make
- ❖ close
- ❖ <-
- ❖ ...



```
func test_fallthrough() {  
    i := 45  
    switch {  
    case i < 10:  
        fmt.Println("i plus petit que 10")  
        fallthrough  
    case i < 50:  
        fmt.Println("i plus petit que 50")  
        fallthrough  
        fmt.Println("bug")  
    case i < 100:  
        fmt.Println("i plus petit que 100")  
    }  
}
```

<https://play.golang.org/p/l2i0vZtItNP>

Fallthrough = cascade d'exécution



- ❖ Fondamental dans les machines multicoeurs
- ❖ Notion de canal de liaison (channel) comme en MPI
- ❖ Permet de faire des actions 'en tâche de fond'



Programming  
Techniques

S. L. Graham, R. L. Rivest  
Editors

---

## Communicating Sequential Processes

C.A.R. Hoare  
The Queen's University  
Belfast, Northern Ireland

---

**This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.**

**Key Words and Phrases:** programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays

**CR Categories:** 4.20, 4.22, 4.32

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if..then..else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (Fortran), procedures (Algol 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (Concurrent Pascal [2]), clusters (CLU [13]), forms (ALPHARD [19]), actors (Hewitt [1]).

The traditional stored program digital computer has been designed primarily for deterministic execution of a single sequential program. Where the desire for greater speed has led to the introduction of parallelism, every attempt has been made to disguise this fact from the programmer, either by hardware itself (as in the multiple function units of the CDC 6600) or by the software (as in an I/O control package, or a multiprogrammed operating system). However, developments of processor technology suggest that a multiprocessor machine, constructed from a number of similar self-contained processors (each with its own store), may become more powerful, capacious, reliable, and economical than a machine which is disguised as a monoprocessor.

In order to use such a machine effectively on a single task, the component processors must be able to communicate and to synchronize with each other. Many methods of achieving this have been proposed. A widely adopted method of communication is by inspection and updating of a common store (as in Algol 68 [18], PL/I, and many machine codes). However, this can create severe problems in the construction of correct programs and it may lead to expense (e.g. crossbar switches) and



- ❖ Executions 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



# CSP : Communicating Sequential Process

## par Tony Hoare, 1978

- ❖ Communicating : les "channel" (chan)  
make (chan string)
- ❖ Sequential process → objets concurrents (go)  
go func
- ❖ Sequential process → blocage sur lecture/écriture  
dans une file // buffer  
Symbole ←-



- ❖ chan : bidirectionnel
- ❖ chan <- : en écriture seulement
- ❖ <- chan : en lecture seulement





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

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

$8*3 = 24$  octets

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

→ stockage séquentiel du tableau

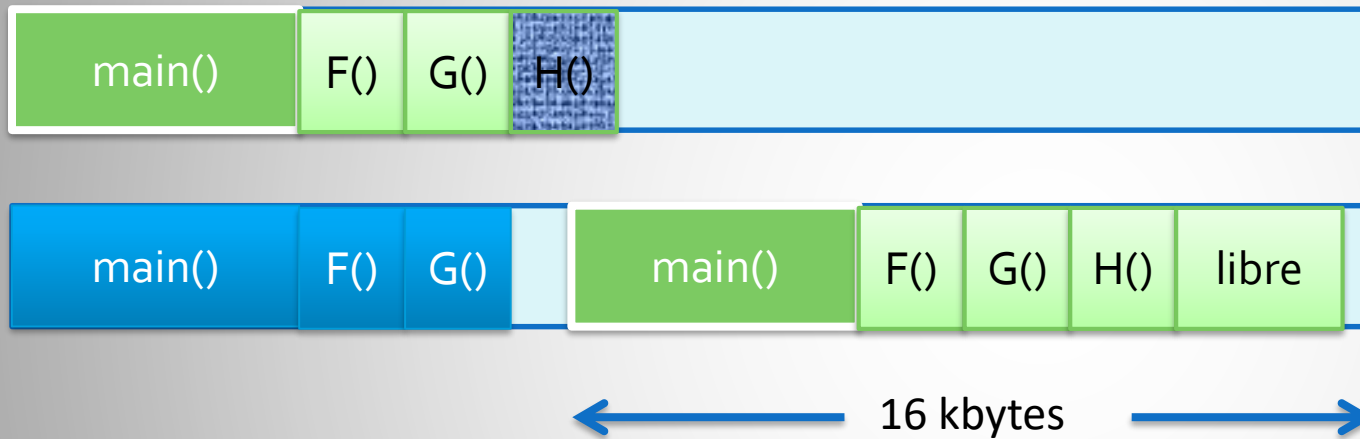


## ❖ Pthread :

- ❖ 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 connait les registres utilisés et sait les sauvegarder à moindre cout



- ❖ 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





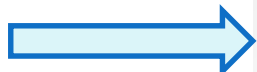
- ❖ Le langage Go permet de faire des appels ASM sans utiliser un outil spécifique.
- ❖ Le «problème» - l'assembleur du langage Go est celui de Plan9 (...68020...) - différent de l'assembleur Intel/ATT
- ❖ Plusieurs tactiques pour obtenir un code assembleur.



- ❖ 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

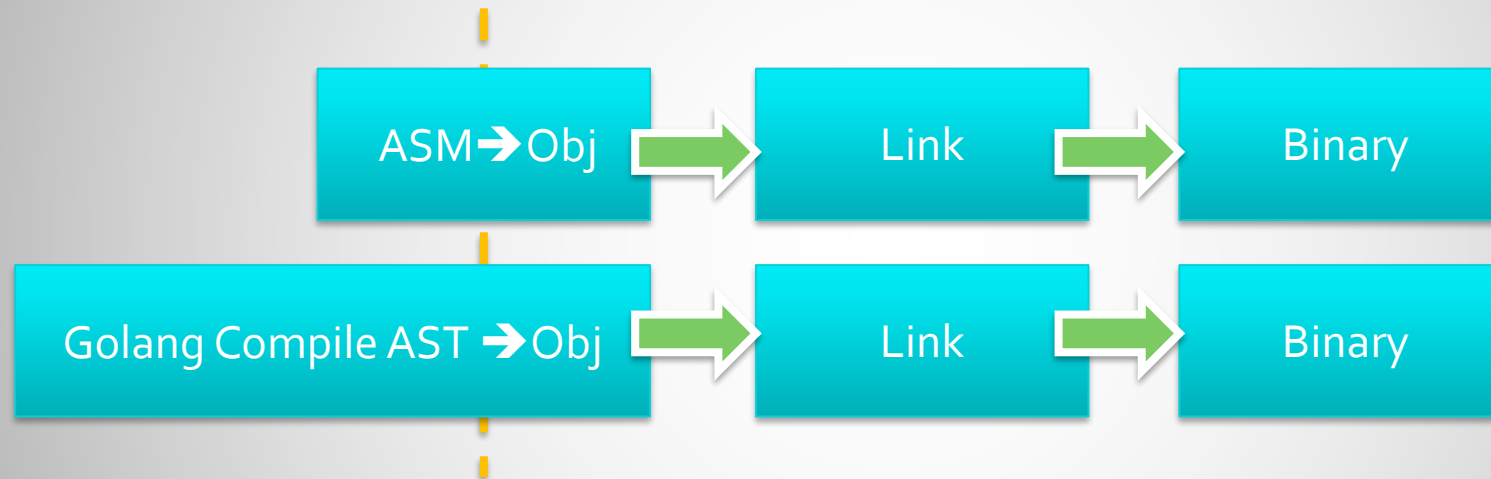


Why Go

## xt file `src/runtime/asm_amd64.s`

```
1 // Copyright 2009 The Go Authors. All rights reserved.
2 // Use of this source code is governed by a BSD-style
3 // license that can be found in the LICENSE file.
4
5 #include "go_asm.h"
6 #include "go_tls.h"
7 #include "funcdata.h"
8 #include "textflag.h"
9 #include "cgo/abi_amd64.h"
10
11 // _rt0_amd64 is common startup code for most amd64 systems when using
12 // internal linking. This is the entry point for the program from the
13 // kernel for an ordinary -buildmode=exe program. The stack holds the
14 // number of arguments and the C-style argv.
15 TEXT _rt0_amd64(SB),NOSPLIT,$-8
16     MOVQ    0(SP), DI    // argc
17     LEAQ   8(SP), SI    // argv
18     JMP    runtime·rt0_go(SB)
```

- ❖ 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





```
func Example() {  
    ch := make(chan string)  
    go func() { ch <- "Hello, world" }()  
    fmt.Println(<-ch)  
    // Output: Hello, world  
}
```

<https://play.golang.org/p/q-ugPxPk8d2>



<https://play.golang.org/p/QxzMmS5QZXR>

❖ Ils sont typés

```
func test_chan() {
    ch := make(chan int)
    fmt.Println("Sending value 1 to channel")
    go send(ch, 1)
    fmt.Println("Receiving from channel")
    go receive(ch)
    time.Sleep(time.Second * 1)
}

func send(ch chan int, i int) {
    ch <- i
}

func receive(ch chan int) {
    val := <-ch
    fmt.Printf("Value Received=%d in receive\n", val)
}
```



## ❖ Buffered

<https://play.golang.org/p/esBAGSAaJ5g>

```
func test_buffered_chan() {  
    ch := make(chan int, 1)  
    ch <- 1  
    fmt.Println("Sending value to channel complete")  
    val := <-ch  
    fmt.Printf("Receiving Value from channel finished. Value received: %d\n", val)  
}
```

<https://play.golang.org/p/TYjoCF4D0HT>

## ❖ Chan en écriture

```
func write_example() {  
    ch := make(chan int, 3)  
    process(ch)  
    fmt.Println(<-ch)  
}  
  
func process(chWrite chan<- int) {  
    chWrite <- 2  
    //s := <-chWrite  
}
```

<https://play.golang.org/p/yr8TU2hdgxY>

## ❖ Chan en lecture

```
func read_example() {  
    ch := make(chan int, 3)  
    ch <- 2  
    process(ch)  
    fmt.Println()  
}  
func process(chRead <-chan int) {  
    s := <-chRead  
    fmt.Println(s)  
    //chRead <- 2  
}
```



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

## ❖ Len() – effectif des msg, Cap() - capacité

```
func cap_len_example() {  
    ch := make(chan int, 3)  
    ch <- 5  
    fmt.Printf("Len: %d, Cap %d\n\n", len(ch), cap(ch))  
    ch <- 6  
    fmt.Printf("Len: %d, Cap %d\n\n", len(ch), cap(ch))  
    ch <- 7  
    fmt.Printf("Len: %d, Cap %d\n\n", len(ch), cap(ch))  
}
```



# channel et opérateur range

```
func test_range() {  
    ch := make(chan int)  
    go sum(ch)  
    ch <- 2  
    ch <- 2  
    ch <- 2  
    close(ch)  
    time.Sleep(time.Second * 1)  
}  
func sum(chRead <-chan int) {  
    sum := 0  
    for a := range chRead {  
        sum += a  
    }  
    fmt.Printf("Sum: %d\n", sum)  
}
```

[https://play.golang.org/p/wvNIG5cax\\_z](https://play.golang.org/p/wvNIG5cax_z)

Range permet  
de défiler un  
chan





# Fermer un channel : close()

<https://play.golang.org/p/Uvjas1wNQ61>

```
func process_ch() {  
    ch := make(chan int, 3)  
    ch <- 2  
    ch <- 2  
    ch <- 2  
    close(ch) ← Fermeture du channel  
    sum(ch)  
    time.Sleep(time.Second * 1)  
}  
  
func sum(ch chan int) {  
    sum := 0  
    for val := range ch {  
        sum += val  
    }  
    fmt.Printf("Sum: %d\n", sum)  
}
```





<https://play.golang.org/p/MNs0GPJeKkt>

```
func test_status() {  
    ch := make(chan int, 1)  
    ch <- 2  
    val, ok := <-ch  
    fmt.Printf("Val: %d OK: %t\n", val, ok)  
    close(ch)  
    val, ok = <-ch  
    fmt.Printf("Val: %d OK: %t\n", val, ok)  
}
```



Commande	Réaction pour channel sans buffer	Réaction pour channel avec buffer	channel fermé	channel assigné à nil
<b>chan &lt;-</b> <b>// écrire</b>	bloquant si pas de lecteur sinon succès	bloque si le channel est plein	panic	bloque indéfiniment
<b>&lt;- chan</b> <b>// lire</b>	bloque s'il n'y a pas d'écrivain	bloque si le channel est vide	récupère le contenu du channel ou alors un type vide	bloque indéfiniment
<b>len()</b> <b>//effectif des msg restant</b>	0	effectif des msg restant	0 pour les channel sans buffer – sinon effectif msg	0
<b>cap()</b> <b>//capacité</b>	0	capacité du channel	0 pour les channel sans buffer – sinon capacité msg	0
<b>close()</b>	Succès	succès	panic	panic



```
const quit_value=999
func fibonacci(cWrite, cmdRead chan int) {
    fmt.Println("fibonacci started")
    x, y := 0, 1
    for {
        fmt.Printf("task fibo x = %d\n", x)
        select {
            case cWrite <- x:
                x, y = y, x+y
            case cmd_value := <-cmdRead:
                fmt.Printf("quit_value = %d\n", cmd_value)
                return
            //default : fmt.Printf("zzz\n")
        }
    }
}

func main() {
    c := make(chan int)
    q := make(chan int)
    go func() {
        fmt.Println("Goroutine started")
        for i := 0; i < 5; i++ {
            value := <-c
            fmt.Printf("main received %d\n", value)
        }
        q <- quit_value
    }()
    //close(c)
    fibonacci(c, q)
}
```

<https://play.golang.org/p/r92J8kn-rfE>

Permet de filtrer le msg

Le select est bloquant,  
sauf avec **default**



# Ce que nous avons vu

- ❖ break
- ❖ default
- ❖ func
- ❖ interface
- ❖ select
- ❖ case
- ❖ defer
- ❖ go
- ❖ map
- ❖ struct
- ❖ chan
- ❖ else
- ❖ goto
- ❖ package
- ❖ switch
- ❖ const
- ❖ fallthrough
- ❖ if
- ❖ range
- ❖ type
- ❖ continue
- ❖ for
- ❖ import
- ❖ return
- ❖ var
- ❖ new
- ❖ make
- ❖ close
- ❖ <-
- ❖ ...



- ❖ 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/oSDZYo0p0YC>



❖ Que décrit la variable t\* testing ?

```
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")  
    }  
}
```





❖ 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`



```
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)
    }
}
```



```
type Job struct {
    a1 string
    a2 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)
    }
}
```



```
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)
    }
}
```



```
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>



## ❖ Les workers pour faire du HPC (utilisation de sa machine locale)

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>



❖ 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/dXHSFrQib00>



# En rouge : ce qui reste à voir.

Nom de l'opérateur	Fonction (par ordre d'importance)
<b>sync.Mutex</b>	Verrou
sync.WaitGroup	Barrière
sync.RWMutex	Verrou multi-processus
atomic.Add	Opérateur Atomique
<b>sync.Once</b>	Execute 1 seule fois
atomic.Load	Opérateur Atomique
atomic.Store	Opérateur Atomique
<b>sync.Pool</b>	<b>Pool mémoire qui traverse le GC</b>
atomic.CompareAndSwap	Opérateur Atomique
<b>sync.Cond (signal, broadcast, lock)</b>	Moniteur (CSP)
atomic.Value	Opérateur Atomique
atomic.Swap	Opérateur Atomique



## ❖ IA et Golang

❖ <https://sketch.dev/>

Can you propose a synthesis of the 23 OOP patterns from the GoF, the example should be done around the idea of a cheeseburger.

**behavioral\_patterns.go:** Contains patterns for object interaction

Chain of Responsibility: Order processing pipeline

Command: Kitchen orders

Interpreter: Custom order rules

Iterator: Menu traversal

Mediator: Kitchen coordination

Memento: Burger state preservation

Observer: Order status notifications

State: Order processing states

Strategy: Pricing strategies

Template Method: Burger preparation steps

Visitor: Menu item operations

**structural\_patterns.go:** Contains patterns for object composition

Adapter: Payment system adaptation

Bridge: Cooking methods

Composite: Menu items and combos

Decorator: Burger customization

Facade: Order processing

Flyweight: Burger type sharing

Proxy: Order service caching

**creational\_patterns.go:** Contains patterns for object creation

Singleton: Restaurant instance

Abstract Factory: Ingredient creation

Builder: Burger construction

Factory Method: Different burger types

Prototype: Cloning burger configurations