Evolution Power4 ⟶ Power5 : qu'est-ce qui change sur le cluster IBM ??

Guy Moebs - CRIHAN

CRIHAN

Plan de la présentation

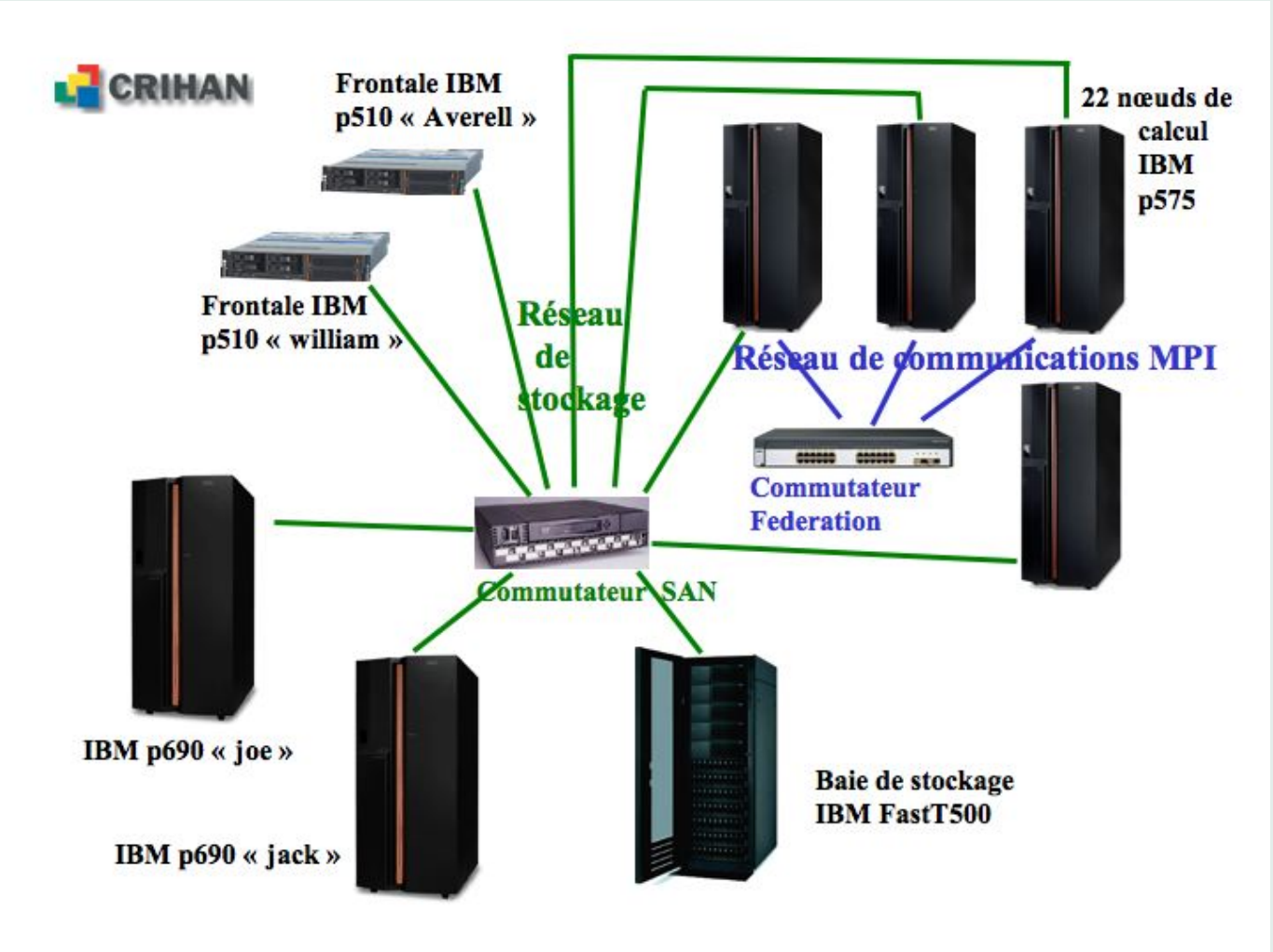
1. Descriptions matérielle
2. Description logicielle
3. Soumission des calculs au travers de LoadLeveler
4. Environnement de compilation XLF

1. Description matérielle

1.1. Vue globale de la solution

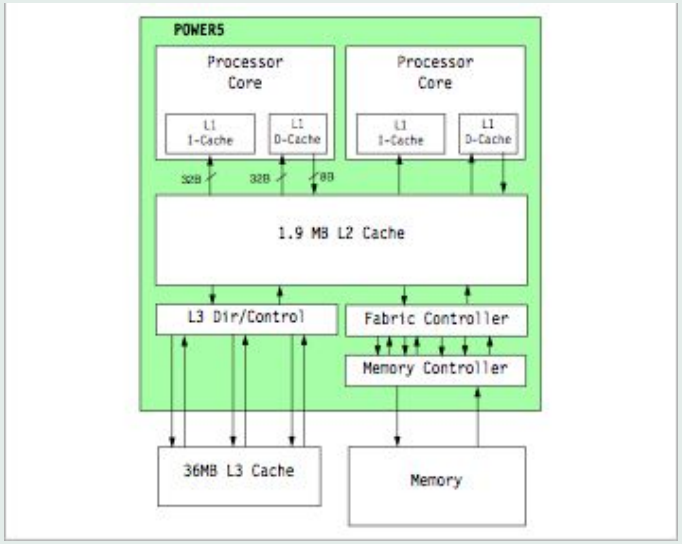
Le cluster IBM est composé de :

- 2 frontales de connexion p510, *william* et *averell*, dotées chacune de :
  - 2 processeurs Power 5 (1.5 GHz) ;
  - 4 Go de mémoire vive ;
- 22 nœuds de calcul SMP p575 dotés chacun de :
  - 8 processeurs Power 5 (1.9 GHz, 7.6 GFlops),
  - 16 Go de mémoire vive ;
- 1 commutateur à faible latence *Federation* qui interconnecte 16 des 22 nœuds de calcul ;
- 1 baie de disques FastT500 (20 To de disque utile), reliée à chacun des nœuds par des liens fibre optique (2Gb/s) ;
- les deux nœuds p690 seront intégrés à la solution dans les prochains jours.


CRIHAN
Frontale IBM
p510 « Averell »
Frontale IBM
p510 « william »
Réseau
de
stockage
22 nœuds de
calcul
IBM
p575
Réseau de communications MPI
Commutateur
Federation
Commutateur SAN
IBM p690 « joe »
IBM p690 « jack »
Baie de stockage
IBM FastT500

1.2. La puce Power 5

- La puce Power5 est à double-cœur mais un seul est activé
- Il dispose d'une hiérarchie de caches L1, L2, L3



1.3. Le Multi Chip Module (MCM)

- Le bloc de base pour un nœud p575 est le MCM (Multi Chip Module)
- Il se compose d'une seule puce Power 5
- Chaque p575 contient donc 8 MCM

1.4. Latences (en cycles d'horloge) et associativité

	Power4	Power5
Registres	1	1
Cache L1	2	2
Cache L2	12	13
Cache L3	123	87
Mémoire	351	235

	Power4	Power5
Cache L1	2 voies associatives, FIFO 32 ko (D) + 64 ko (I) 128 octets	4 voies associatives LRU 32 ko (D) + 64 ko (I) 128 octets
Cache L2	4 voies associatives 1440 ko (par double cœur) 128 octets	10 voies associatives 1920 ko (un seul cœur) 128 octets
Cache L3	8 voies associatives 32 Mo 256 octets	12 voies associatives 36 Mo 256 octets

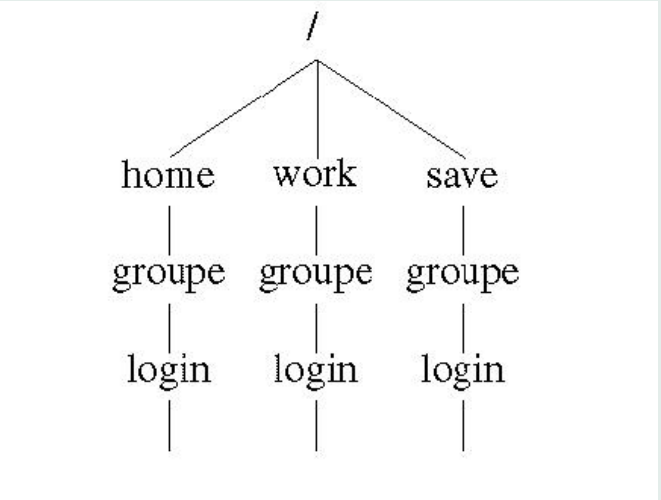
1.5. Configuration mémoire

- Chaque nœud de calcul p575 dispose de 16 Go de mémoire physique
  ⇒ soit environ 12 Go de mémoire disponible pour les applications
- Chaque nœud est configuré en 10 Go de *Large Pages* (LP) et 2 Go de *Small Pages* (SP)
- Une Large Page est une page mémoire de 16 Mo
- Une Small Page est une page mémoire de 4 ko
- Les performances sont nettement meilleures (jusqu'à 60% de gain) lorsque les *Large Pages* sont employées (Data Prefetch Streams, ..)
  ⇒ Elles sont mises par défaut sur tous les nœuds
- Qu'est-ce que cela change pour les utilisateurs ?
  ⇒ Rien ou presque .... sauf à l'édition des liens !
- Il faut aussi légèrement modifier les scripts de soumission
- Les *Large Pages* sont employées pour les données globales (heap)
- Les *Small Pages* sont employées pour les variables locales (stack)

1.6. Espaces utilisateurs

Chaque utilisateur dispose de 3 espaces permanents au sein de son projet qui sont physiquement localisés dans la baie FastT500.

- un espace utilisateur `/home/projet/login` (5 Go),
- un espace de travail `/work/projet/login` (25 Go),
- un espace de stockage `/save/projet/login` (300 Go).



⇒ Les exécutions des programmes en mode batch se font dans des espaces temporaires situés dans la baie de disques FastT500

2. Description logicielle

2.1. Environnement de travail

Système d'exploitation	AIX 5.3 , UNIX d'IBM
Compilateur Fortran	XLF, v. 9.1.0.4
Compilateur C	XLC, v. 7.0.0.3
Bibliothèques scientifiques	ESSL (inclus BLAS, FFT), v 4.2.0.0 ; LAPack, v 3.0 ; P-ESSL, v 3.2.0.0
Calcul parallèle	MPI, POE v 3.2.0.20 ; OpenMP
Gestionnaire de batch	LoadLeveler, v. 4.2.2.1
Outils d'analyse	prof, gprof, xprofiler, PE Benchmarker (pct, pvt, ute), jumpshot
Débogueurs symboliques	dbx, pdbx
Editeurs de texte	vi ; emacs, v 21.4.1
Graphisme	gnuplot, v 3.7.2 ; xmgrace, v 5.1.11 ; pgplot ; ncview
Bibliothèque	netCDF (32 et 64 bits), v 3.6.0-p1 ; NCO v 3.0.0

2.2. Logiciels scientifiques de chimie disponibles

Codes commerciaux accessibles à tous les utilisateurs

Gaussian	G03, G98
Schroedinger	Jaguar 6.0, 5.0, 4.1
Accelrys	Catalyst 4.9, CNX 2002

Codes ”libres” sous licence (usage non commercial)

IBM	CPMD v 3.9.1
Yale Univ.	CNS v 1.1 (version gratuite de CNX)
Iowa State Univ.	Gamess 16/02/2002

⇒ On peut installer vos logiciels commerciaux sous licence :
→ fournissez-nous vos licences ...
→ on restreint les droits d’accès à votre projet

2.3. Connexion et modes d'exécution

- Les ressources du cluster sont séparées en deux parties :

  (i) les frontales de connexion, bi-processeurs, p510, pour la connexion et la petite mise au point ;

  (ii) les nœuds de calculs, dédié au batch : 22 nœuds octo-processeurs p575 et 2 nœuds p690.

- Les connexions, les éditions, les compilations, les exécutions en interactif prennent leurs ressources dans la partie **(i)**
- L'utilisateur exprime ses besoins en temps, processeurs, mémoire (data + stack), dans un script

- Chaque nœud p575 dispose de 16 Go de mémoire physique, soit environ 12 Go de mémoire disponible pour les applications
  ⇒ Sa mémoire est configurée en 10 Go de *Large Pages* (LP) et 2 Go de *Small Pages* (SP)
- Chaque nœud p690 dispose de 32 Go de mémoire physique, soit environ 30 Go de mémoire disponible pour les applications
  ⇒ Sa mémoire est configurée en 20 Go de *Large Pages* (LP) et 10 Go de *Small Pages* (SP)

3. Soumission des calculs au travers de LoadLeveler

- Loadleveler : logiciel de soumission de travaux sur calculateur IBM
- La soumission fonctionne à l'aide de scripts
- Les paramètres sont transmis à l'aide de directives
- les directives ont la forme `# @ directive`
- Il n'y a pas de queue batch sur le cluster
- L'expression des besoins "pondère" les soumissions et décide des priorités

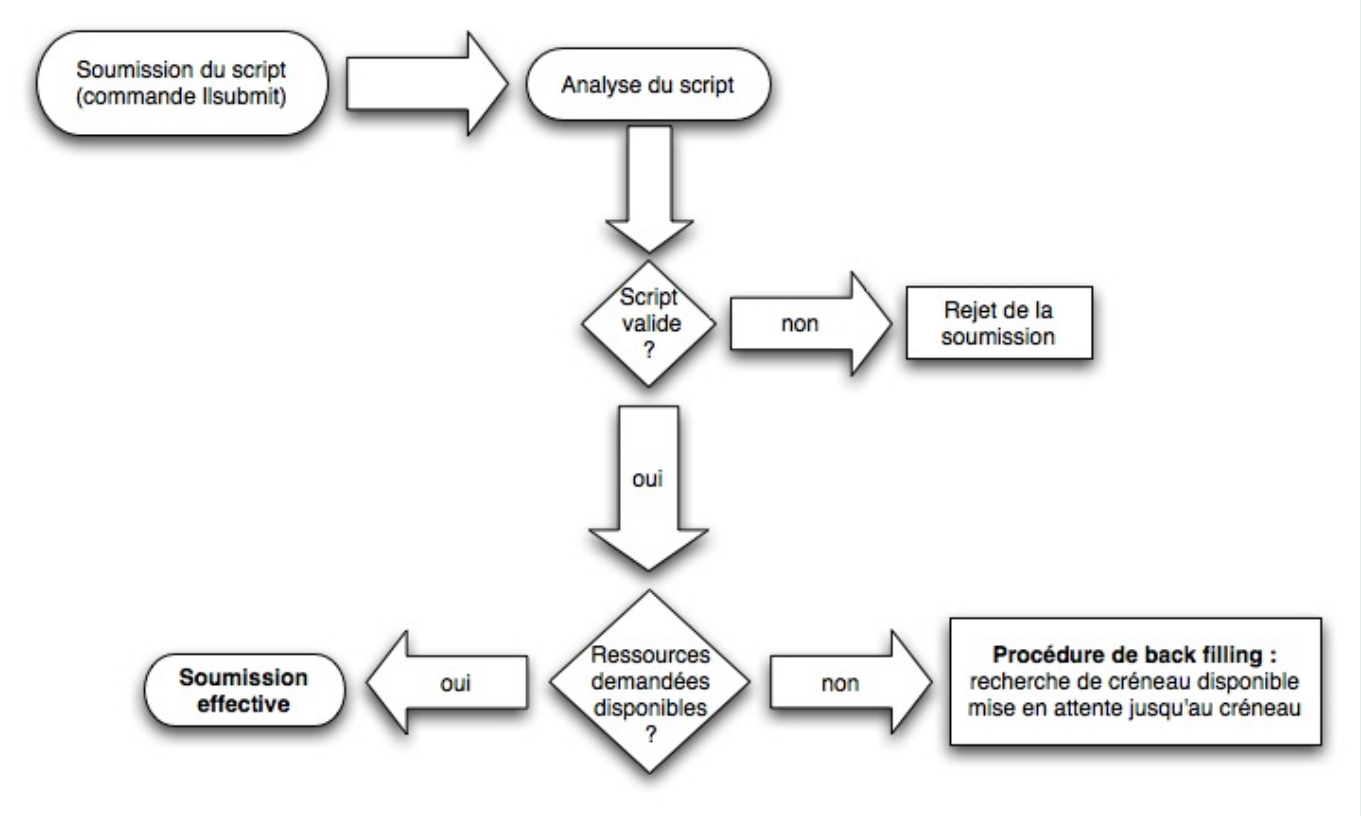
3.1. Spécificités architecturales

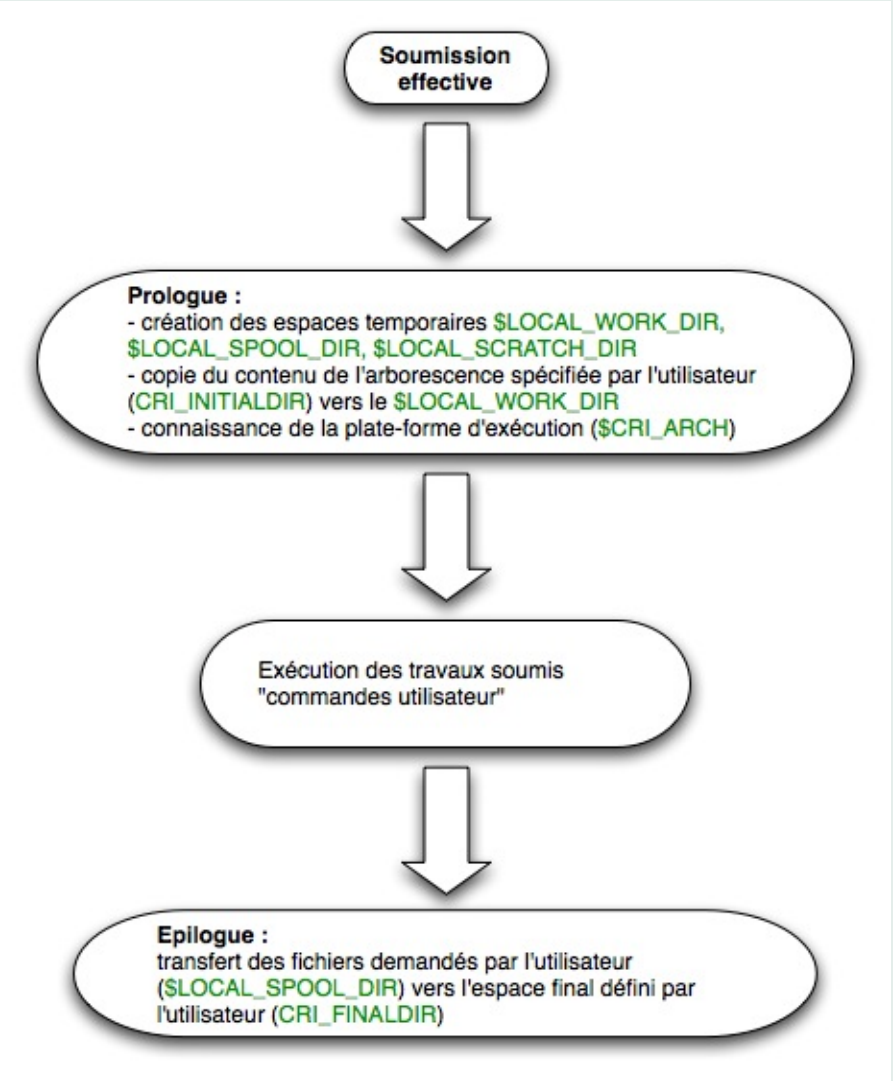
- Les comptes utilisateurs sont situés dans la baie externe FastT500
- Les exécutions ont lieu dans les disques de la baie externe (/dlocal)

  ⇒ les jobs MPI sont à cheval sur les nœuds par défaut

  ⇒ Le système gère les procédures de transfert de fichiers
- Il faut fixer les quantités de mémoire allouées pour les *Large Pages* et les *Small Pages*

3.2. Principe des soumissions




Soumission effective
Prologue :
- création des espaces temporaires $LOCAL_WORK_DIR, $LOCAL_SPOOL_DIR, $LOCAL_SCRATCH_DIR
- copie du contenu de l'arborescence spécifiée par l'utilisateur (CRI_INITIALDIR) vers le $LOCAL_WORK_DIR
- connaissance de la plate-forme d'exécution ($CRI_ARCH)
Exécution des travaux soumis "commandes utilisateur"
Epilogue :
transfert des fichiers demandés par l'utilisateur ($LOCAL_SPOOL_DIR) vers l'espace final défini par l'utilisateur (CRI_FINALDIR)

- Le **$LOCAL_WORK_DIR** représente le répertoire temporaire crée par le système dans les disques de la baie FastT500, visible par tous les nœuds d'exécution ;
  → /dlocal/run/ll-william.xxxx.0 ou /dlocal/run/ll-averell.xxxx.0

- Le **CRI_INITIALDIR** représente le répertoire spécifié par l'utilisateur dans son espace permanent sur la baie FastT500 dont le contenu est dupliqué dans le $LOCAL_WORK_DIR ;
  → /work/projet/login/REPERTOIRE_ENTREE

- Le **CRI_FINALDIR** représente le répertoire spécifié par l'utilisateur dans son espace permanent sur la baie FastT500 dans lequel les fichiers spécifiés par l'utilisateur sont rapatriés par le système ;
  → /work/projet/login/REPERTOIRE_SORTIE

- Le **$LOCAL_SPOOL_DIR** représente le répertoire temporaire crée par le système dans les disques de la baie FastT500 qui sert de transit avant le rapatriement vers l'espace permanent de l'utilisateur situé dans la baie, visible par tous les nœuds d'exécution.
  → /dlocal/spool/ll-william.xxxx.0 ou /dlocal/spool/ll-averell.xxxx.0

- Le **$LOCAL_SCRATCH_DIR** représente le répertoire temporaire crée dans les disques internes de **chaque nœud** et pouvant servir de mémoire tampon si nécessaire
  ⇒ Très utile pour les codes utilisant des gros volumes disques comme zone de scratch
  ⇒ Espace d'environ 80 Go par nœud, partagé entre les jobs du nœud, **sans contrôle** (quota de 70 Go par utilisateur)

- Le **$LOCAL_SCRATCH_DIR** n'est visible qu'au sein de son nœud
  ⇒ Son usage est donc restreint aux applications s'exécutant en intra-nœud et aux fichiers gérés par un ensemble de processus dans un même nœud

- Les **CRI_INITIALDIR** et **CRI_FINALDIR** doivent obligatoirement respecter les règles suivantes :
  - être des chemins absolu dans le $HOME_DIR ou le $WORK_DIR
  - ne contenir que des fichiers réguliers
  - avoir des droits (lecture/écriture) compatibles avec les actions
  - le **CRI_INITIALDIR** ne doit pas être vide
  - le **CRI_FINALDIR** est crée s'il n'existe pas

3.3. Sur quels nœud(s) tournent mes jobs ?

- Plate-forme hétérogène
- Les jobs non spécifiques (1) sont susceptibles de s'exécuter sur tous les nœuds
- Loadleveler attribue les ressources parmi ses disponibilités
- L'optimisation d'un code à une plate-forme améliore ses performances sur cette plate-forme mais ....
  qu'en est-il sur une autre ???

  - La compatibilité est ascendante :
    { un code optimisé pour Power 4 s'exécute normalement sur Power 5
    { un code optimisé pour Power 5 **peut ne pas s'exécuter** normalement sur Power 4 !!
  - Un code optimisé pour Power 4 est moins efficace sur Power 5 qu'un code optimisé pour Power 5 (dégradation variable, dépend de l'application)

  ⇒ Alors que faire ???

(1) jobs spécifiques : une seule architecture peut satisfaire les besoins en ressources, par exemple
demander plus de 10 Go par processus envoie sur p690,
demander plus de 32 processeurs en MPI envoie sur p575, ...

Sur quels nœud(s) tournent mes jobs ?

- Avoir un binaire, si possible, pour chaque plate-forme
  ⇒ Avoir deux fichiers de compilation Makefile
  ⇒ Avoir deux répertoires pour les fichiers objet
- S'adapter à la plate-forme d'exécution :
  `-qarch=pwr4 -qtune=pwr4` pour les code destinés aux p690
  `-qarch=pwr5 -qtune=pwr5` pour les code destinés aux p575
- Ajouter le choix du binaire, selon la plate-forme d'exécution, dans le script de soumission :

Script de soumission en Korn Shell	**Script de soumission en C Shell**
if [$CRI_ARCH = "pwr4"]	if ($CRI_ARCH == pwr4)
then	then
./a.out_ pwr4	./a.out_ pwr4
else	else
./a.out_ pwr5	./a.out_ pwr5
fi	endif

- Cela ne concerne pas les applications précompilées installées par le CRIHAN

3.4. Mots-clés

Mot-clé	Séquentiel	MPI	OpenMP	Gaussian (03,98)
`cri_job_type`	`serial`	`mpi`	`openmp`	`gaussian`
`cri_total_tasks`	sans objet	nombre de processus MPI	nombre de threads OpenMP	nombre de threads Gaussian

- `wall_clock_limit` : temps de **présence** du job sur la machine
- `data_limit` : mémoire **par processus**
  ⇒ Mémoire prise dans les *Large Pages*
  ⇒ Attention à l'option `-bmaxdata`
- `stack_limit` : mémoire **par processus**
  ⇒ Mémoire prise dans les *Small Pages*
  ⇒ Mot-clef optionnel (256 Mo par défaut)
- `core_limit` : taille maximale **par processus**
  ⇒ Mot clef optionnel (0 Mo par défaut)
- Liste non exhaustive ... (il existe aussi des scripts pour Gamess, Jaguar, ...)
- à compléter selon les besoins des utilisateurs

3.5. Commandes

- soumission : `llsubmit` script_job → *ref_job* (ll-william.xxxx.0 ou ll-averell.xxxx.0)

  lsubmit ll_mpi

  llsubmit: Processed command file through Submit Filter: ”/soft/loadl/LLFILTER/llfilter”.

  llsubmit: The job ”william-a1.crihan.fr.2045” has been submitted.

- suivi : `llq`

Id	Owner	Submitted	ST	PRI	Class	Running On
william-a1.1973.0	login	2/1 13:10	R	50	hps_indus	p5-a1-n23
william-a1.1848.0	login	1/30 09:21	R	50	hps_large	p5-a1-n7
william-a1.1899.0	login	1/30 20:51	R	50	medium	p5-a1-n24
william-a1.1978.0	login	2/1 14:28	R	50	xsmall	p5-a1-n2
william-a1.1965.0	login	1/31 22:51	R	50	medium	p5-a1-n11
william-a1.1960.0	login	1/31 17:13	I	50	hps_xlarge	

- destruction : `llcancel` *ref_job*

  llcancel william-a1.crihan.fr.2045

  llcancel: Cancel command has been sent to the central manager.

3.6. Soumission job séquentiel

```
# !/bin/csh
# Script de soumission Loadleveler, job sequentiel
# Nom du job
# @ job_name = job_sequentiel
# Nom des fichiers de sortie et d'erreur standard
# @ output = $(job_name).o$(jobid)
# @ error = $(job_name).e$(jobid)

# Type du job
# @ cri_job_type = serial
# temps de restitution (heures[:minutes[:secondes]])
# @ wall_clock_limit = 1:00:00

# Mémoire maximale par processus (mb, gb, mw, gw, ..)
# @ data_limit = 600mb

# Stack maximale par processus (mb, gb, mw, gw, ..)
# @ stack_limit = 100mb
```

```
# Répertoire du compte utilisateur dont le contenu est copié
# @ cri_initialdir = /work/crihan/gm/JOB_SEQ/ENTREE

# Répertoire du compte utilisateur pour y placer les résultats
# @ cri_finaldir = /work/crihan/gm/JOB_SEQ/SORTIE

# Politique d'envoi des mels
# @ notification = complete
# Adresse d'envoi des mels
# @ notify_user = gm@crihan.fr

# Obligatoire
# @ queue
```

```
###
### Commandes utilisateur
###
# Déplacement dans le répertoire temporaire
cd $LOCAL_WORK_DIR

# Exécution du programme séquentiel
if ($CRI_ARCH == pwr4) then
  ./a.out_pwr4 > $LOCAL_SPOOL_DIR/OUT
else
  ./a.out_pwr5 > $LOCAL_SPOOL_DIR/OUT
endif

# Déplacement des fichiers à récupérer
mv resultat.dat $LOCAL_SPOOL_DIR
```

3.7. Soumission job parallèle MPI

```
# !/bin/csh
# Script de soumission Loadleveler, job MPI
# Nom du job
# @ job_name = job_mpi

# Nom des fichiers de sortie et d'erreur standard
# @ output = $(job_name).o$(jobid)
# @ error = $(job_name).e$(jobid)

# Type du job
# @ cri_job_type =  mpi
# Nombre de processus MPI
# @ blocking = 4 pour rester dans un seul nœud
# @ cri_total_tasks = 4

# temps de restitution (heures[:minutes[:secondes]])
# @ wall_clock_limit = 1:00:00
```

```
# Mémoire maximale par processus (mb, gb, mw, gw, ..)
# @ data_limit = 1000mb

# Stack maximale par processus (mb, gb, mw, gw, ..)
# @ stack_limit = 120mb

# Répertoire du compte utilisateur dont le contenu est copié
# @ cri_initialdir = /work/crihan/gm/JOB_MPI/ENTREE
# Répertoire du compte utilisateur pour y placer les résultats
# @ cri_finaldir = /work/crihan/gm/JOB_MPI/SORTIE

# Politique d'envoi des mels
# @ notification = complete
# Adresse d'envoi des mels
# @ notify_user = gm@crihan.fr

# Obligatoire
# @ queue
```

```
###
### Commandes utilisateur
###
# Déplacement dans le répertoire temporaire
cd $LOCAL_WORK_DIR

# Exécution du programme MPI
if ($CRI_ARCH == pwr4) then
  ./a.out_pwr4 > $LOCAL_SPOOL_DIR/OUT
else
  ./a.out_pwr5 > $LOCAL_SPOOL_DIR/OUT
endif

# Déplacement des fichiers à récupérer
mv resultat.dat $LOCAL_SPOOL_DIR
```

3.8. Soumission job parallèle OpenMP

```
# !/bin/csh
# Script de soumission Loadleveler, job OpenMP
# Nom du job
# @ job_name = job_openmp

# Nom des fichiers de sortie et d'erreur standard
# @ output = $(job_name).o$(jobid)
# @ error = $(job_name).e$(jobid)

# Type du job
# @ cri_job_type =  openmp
# Nombre maximal de threads OpenMP
# @ cri_total_tasks = 4

# temps de restitution (heures[:minutes[:secondes]])
# @ wall_clock_limit = 1:00:00
```

```
# Mémoire maximale pour l'application (mb, gb, mw, gw, ..)
# @ data_limit = 1600mb

# Stack maximale pour l'application (mb, gb, mw, gw, ..)
# @ stack_limit = 400mb

# Répertoire du compte utilisateur dont le contenu est copié
# @ cri_initialdir = /work/crihan/gm/JOB_OMP/ENTREE
# Répertoire du compte utilisateur pour y placer les résultats
# @ cri_finaldir = /work/crihan/gm/JOB_OMP/SORTIE

# Politique d'envoi des mels
# @ notification = complete
# Adresse d'envoi des mels
# @ notify_user = gm@crihan.fr

# Obligatoire
# @ queue
```

```
###
### Commandes utilisateur
###
# Déplacement dans le répertoire temporaire
cd $LOCAL_WORK_DIR

# Exécution du programme OpenMP
if ($CRI_ARCH == pwr4) then
  ./a.out_pwr4 > $LOCAL_SPOOL_DIR/OUT
else
  ./a.out_pwr5 > $LOCAL_SPOOL_DIR/OUT
endif
#

# Déplacement des fichiers à récupérer
mv resultat.dat $LOCAL_SPOOL_DIR
```

3.9. Soumission job parallèle Gaussian

```
# !/bin/csh
# Script de soumission Loadleveler, job Gaussian
# Nom du job
# @ job_name = job_g03

# Nom des fichiers de sortie et d'erreur standard
# @ output = $(job_name).o$(jobid)
# @ error = $(job_name).e$(jobid)

# Type du job
# @ cri_job_type =  gaussian
# Nombre maximal de threads Gaussian
# @ cri_total_tasks = 4

# temps de restitution (heures[:minutes[:secondes]])
# @ wall_clock_limit = 1:00:00
```

```
# Mémoire maximale pour l'application (mb, gb, mw, gw, ..)
# @ data_limit = 2000mb

# Stack maximale pour l'application (mb, gb, mw, gw, ..)
# @ stack_limit = 400mb

# Répertoire du compte utilisateur dont le contenu est copié
# @ cri_initialdir = /work/crihan/gm/JOB_G03/ENTREE
# Répertoire du compte utilisateur pour y placer les résultats
# @ cri_finaldir = /work/crihan/gm/JOB_G03/SORTIE

# Politique d'envoi des mels
# @ notification = complete
# Adresse d'envoi des mels
# @ notify_user = gm@crihan.fr

# Obligatoire
# @ queue
```

```
###
### Commandes utilisateur
###
# Déplacement dans le répertoire temporaire
cd $LOCAL_WORK_DIR

# Variables d'environnements nécessaires à Gaussian
# N.B. :$LOCAL_WORK_DIR est dans /dlocal (partagé)
# N.B. :$LOCAL_SCRATCH_DIR est dans /local (propre au noeud)

setenv PATH "${PATH}:.:/soft/g03c02/g03"
setenv g03root /soft/g03c02
setenv GAUSS_SCRDIR $LOCAL_SCRATCH_DIR
setenv GAUSS_EXEDIR /soft/g03c02/g03
setenv LD_LIBRARY64_PATH /soft/g03c02/g03

# Exécution du programme Gaussian
g03 < gaussian.in > $LOCAL_SPOOL_DIR/gaussian.log
# Déplacement des fichiers à récupérer
mv gaussian.chk $LOCAL_SPOOL_DIR
```

4. Environnement de compilation XLF

4.1. Les commandes de compilation

Il existe une famille de compilateurs XLF selon le type d'application et le respect de la norme POSIX pour les threads.
⇒ Il est conseillé de toujours utiliser des compilateurs *threadsafes*, i.e. qui permettent l'exécution simultanée correcte d'une même portion de code (parallélisable !) par plusieurs threads :
⇒ suffixe _r

Type d'application	Fortran 77	Fortran 90	Fortran 95
Code séquentiel	`xlf_r`	`xlf90_r`	`xlf95_r`
Code parallèle MPI	`mpxlf_r`	`mpxlf90_r`	`mpxlf95_r`
Code parallèle OpenMP, PVM, P-threads, ...	`xlf_r`	`xlf90_r`	`xlf95_r`

L'édition des liens se fait avec la même commande que la compilation.

4.2. Options de compilation : entrées / sorties du compilateur

- **-qfixed=132** : fichier source au format fixe (-qfixed=72 défaut pour Fortran 77)
- **-qfree=f90** : fichier source au format libre (défaut pour Fortran 9x)
- **-I***dir* : chemin pour les fichiers à inclure
- **-qsuffix=f=f90** (ou **f**) : extension des fichiers source
- **-qmoddir=***dir* : précise le répertoire de création des fichiers modules

4.3. Options de compilation : portage

- **-qdpc=e** : les constantes réelles numériques sont converties en double précision
- **-qautodbl=dbl4** : conversion automatique des REAL(4) en REAL(8) et des COMPLEX(4) en COMPLEX(8)
- **-qnosave** : variables locales dynamiques et non pas statiques
- **-qundef** : rejet des déclarations implicites pour les variables
- **-qextname** : ajout d'un caractère _ à la fin de tous les noms d'objets venant d'autres systèmes où ils sont absents (exemple : flush)

4.4. Options de compilation : déboguage

- **-qnooptimize** : pas d'optimisation du code
- **-qcheck** : vérification des accès aux éléments de tableaux explicitement dimensionnés
- **-qdbg** : informations pour le débogueur symbolique
- **-qextchk** : vérification des informations sur les types de données dans les blocs commons, dans les définitions des procédures
- **-qflttrap=ov:und:zero:inv:en** : type d'exceptions flottantes tracées
- **-qfullpath** : inclusion du chemin absolu UNIX vers les fichiers sources
- **-qinitauto=FF** : initialisation à NaN de toutes les variables automatiques
- **-qfloat=nans** : détection des opérations flottantes utilisant des NaN

- Proposition d'options :
  -qnooptimize -qcheck -qdbg -qflttrap=ov:und:zero:inv:en -qfullpath
  -qinitauto=FF -qfloat=nans [-qextchk]

4.5. Options de compilation : profilage / optimisation

- **-O2**, **-O3** : niveau d'optimisation
- **-qstrict** : respect de la sémantique des programmes (si nécessaire)
- **-qhot** : optimisation plus aggressives (**-O3** nécessaire)
- **-qunroll=[auto|yes]** : active ou renforce l'analyse des boucles
- **-qarch=pwr[5|4]** et **-qtune=pwr[5|4]** : optimisations spécifiques à l'architecture
- **-qipa** : analyse interprocédurale
- **-Q<x>** : inlining (**-Q**) ou non (**-Q!**), sélectif (**-Q+***name1[:name2]* ou **-Q-***name1[:name2]*)
  → nécessite **-qipa** et au moins **-O2**
- **-qsmp=omp** : parallélisation avec directives OpenMP
- **-pg** : prépare le programme au profilage par `gprof`
- **-qreport=[hotlist|smplist]** fichiers listing des transformations du code

- Proposition d'options :
  -qarch=pwr[5|4] -qtune=pwr[5|4] [-qhot] -O3 [-qstrict] [-qsmp=omp] [-qipa]

4.6. Edition des liens

- **-bmaxdata:0x40000000** : 4 segments de 256 Mo pour les données du programme (*heap*)
- **-bmaxstack:0x10000000** : 256 Mo pour les variables locales (*stack*)

⇒ Options INutiles et pénalisantes en mode 64 bits

- **-brename:.name, .name_** : ajout d'un caractère souligné '_' à la fin d'un nom

⇒ Option utile pour le portage de `flush`, ...

- **-blpdata** : demande de larges pages (16 Mo) pour les applications consommatrices de mémoire
  Cette option est obligatoire et mise par défaut dans la configuration du compilateur MAIS pour les binaires importés d'autres plate-formes AIX, il faut appliquer la commande `ldedit` :

  ldedit -blpdata <executable>

- **-qipa** : si elle est présente lors de la compilation

Bilan compilation et soumission des travaux

- J'ai les sources de mon code :
  1. adapter le Makefile
  2. `make clean ; make`
- Je ne dispose que des binaires :
  ⇒ forcer la prise en compte des Large Pages avec

  `ldedit -blpdata <executable>`
- J'enrichi le script de soumission pour couvrir les deux architectures

Bons calculs à tous ....