Utilisation de la machine Blue Gene/Q de l'IDRIS

philippe.wautelet@aero.obs-mip.fr

CNRS — IDRIS

Version 1.5 — 21 juin 2016

Sommaire I

Sommaire II

Sommaire III

Problématique et approche Blue Gene/Q

Problématique et approche Blue Gene/Q

Problématique

- Puissance électrique dissipée = $frequence^3$
- Vitesse mémoire vive augmente beaucoup plus lentement que puissance crête processeur (*memory wall*)
- Puissance dissipée par cm^2 limitée par le refroidissement

Solution Blue Gene/Q

- Si f/2 => consommation d'1 cœur / 8
- Si 16 cœurs/puce => puissance crête x 8 et consommation x 2

=> efficacité énergétique x 4 (FLOP/s/Watt)

Avantages

Avantages d'une fréquence CPU basse :

- Diminution écart débits et latences mémoires
- Diminution écart débits et latences réseaux

=> machine plus équilibrée

- Consommation électrique par cœur faible
- Augmentation de la puissance crête à consommation électrique constante
- Augmentation de la densité (nb de cœurs par rack)
- Augmentation de la fiabilité (accrue aussi par technologie SoC et tests de validation)

=> puissance de calcul crête élevée pour une consommation électrique donnée, une surface au sol limitée et un nombre de pannes raisonnable

Problématique et approche Blue Gene/Q

Inconvénients

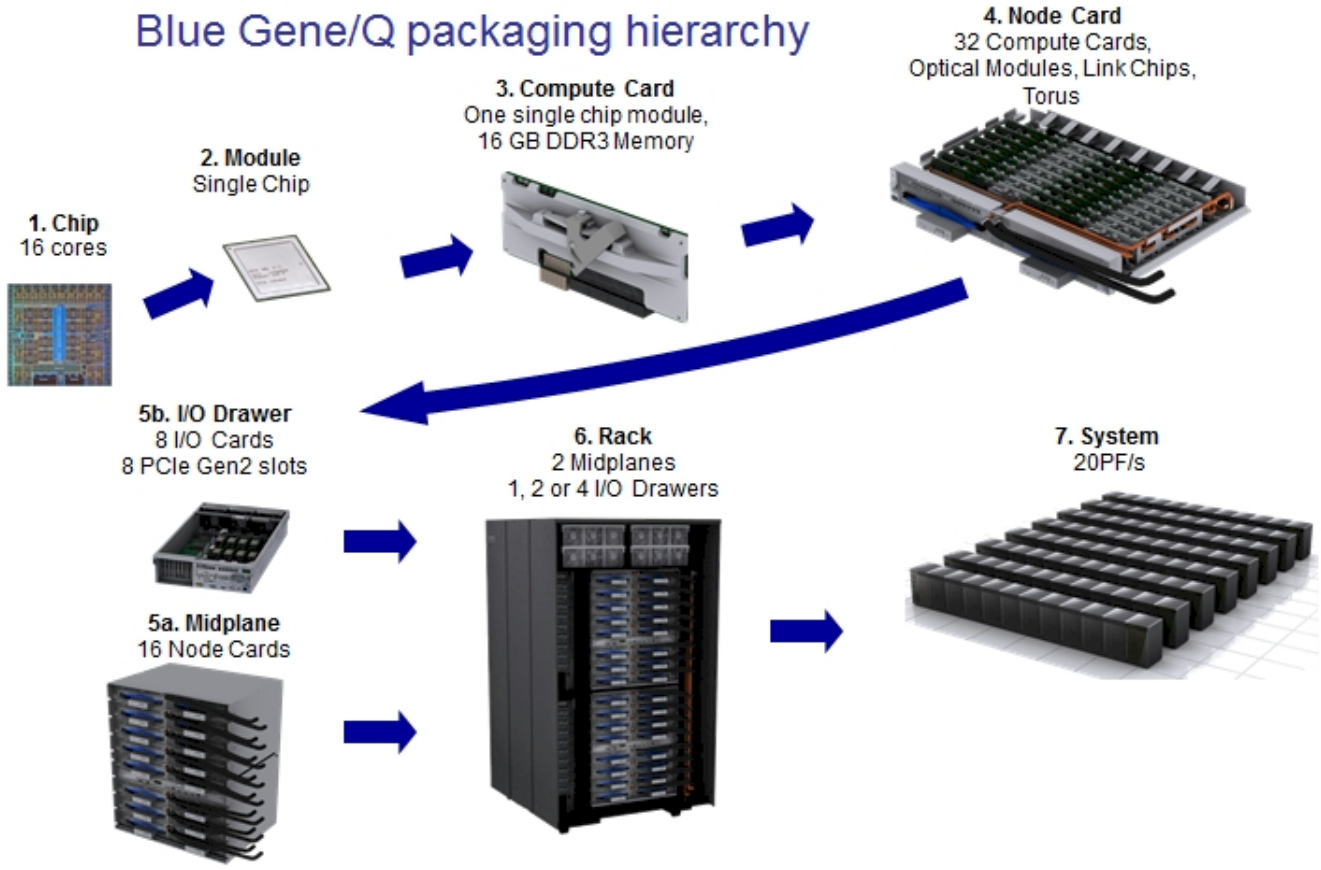
Inconvénients d'une fréquence CPU basse :

- Chaque cœur est lent
- Grand nombre de cœurs
- Puissance de calcul élevée seulement si grand nombre de cœurs utilisés
- Peu de mémoire par cœur

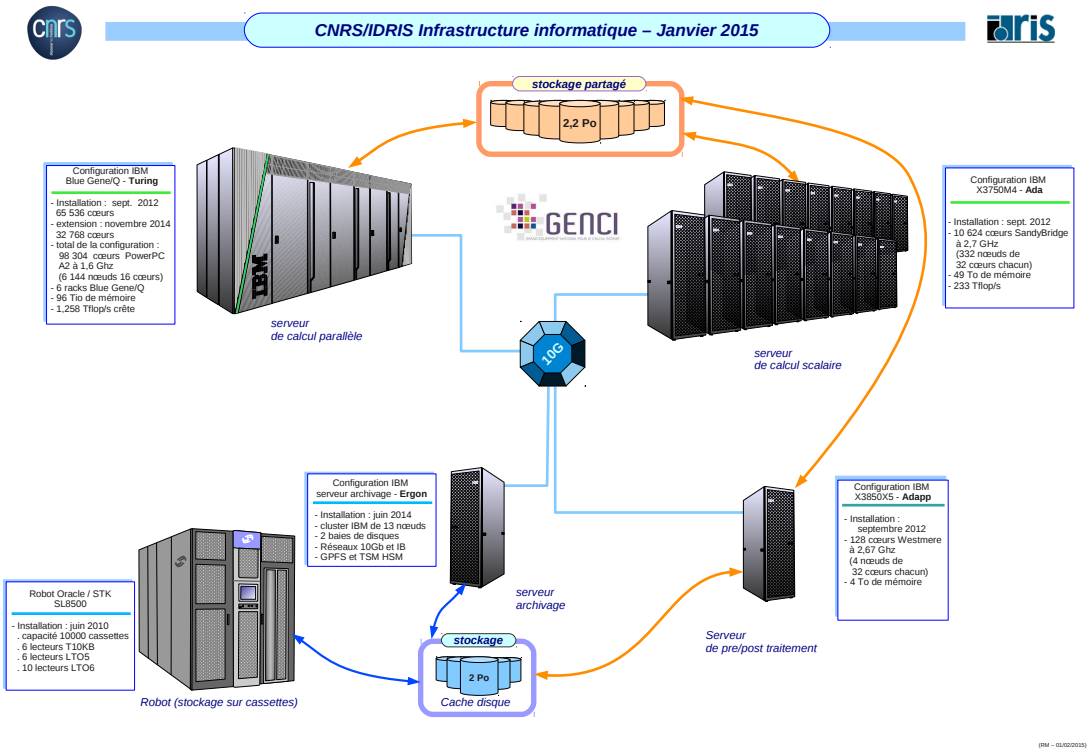
=> parallélisme massif, machine non généraliste, gamme d'applications limitée

Hardware

Architecture Blue Gene/Q



Configuration complète



Configuration complète

Chiffres importants

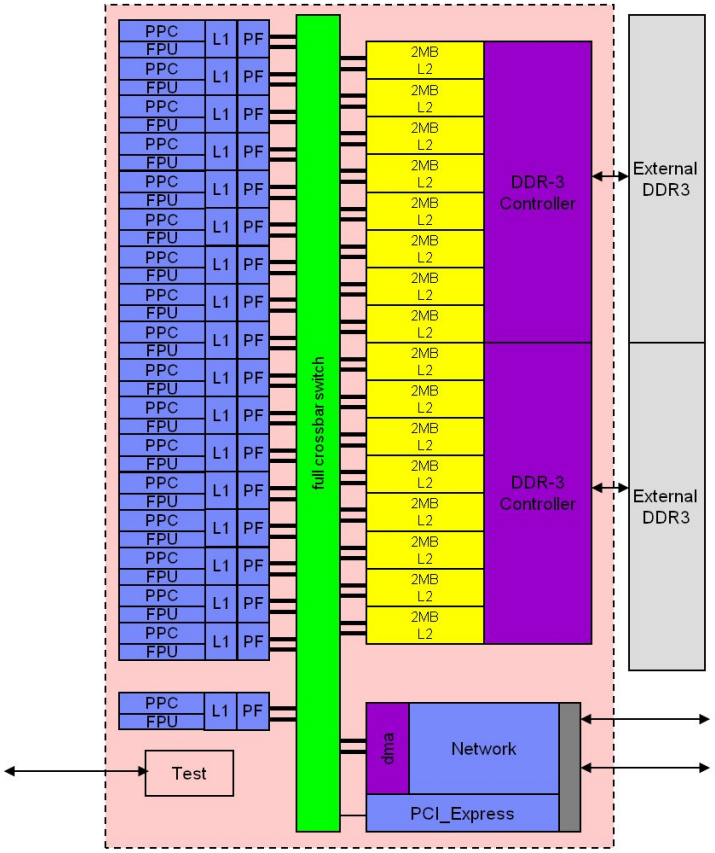
- **Turing : 6 *racks* Blue Gene/Q** :
  - 6.144 nœuds
  - 98.304 cœurs
  - 393.216 *threads*
  - 96 Tio
  - 1,258 Pflop/s
  - 636 kW (106 kW/*rack*)
- **Ada : 15 *racks* IBM x3750M4** :
  - 332 nœuds de calcul et 4 nœuds pour pré et post-traitement
  - 10.624 cœurs Intel SandyBridge à 2,7 GHz
  - 46 Tio
  - 230 Tflop/s
  - 366 kW
- 2,2 Po utiles et 50 Gio/s sur disques partagés entre BG/Q et Intel
- 1 MW pour la configuration complète (hors climatisation)

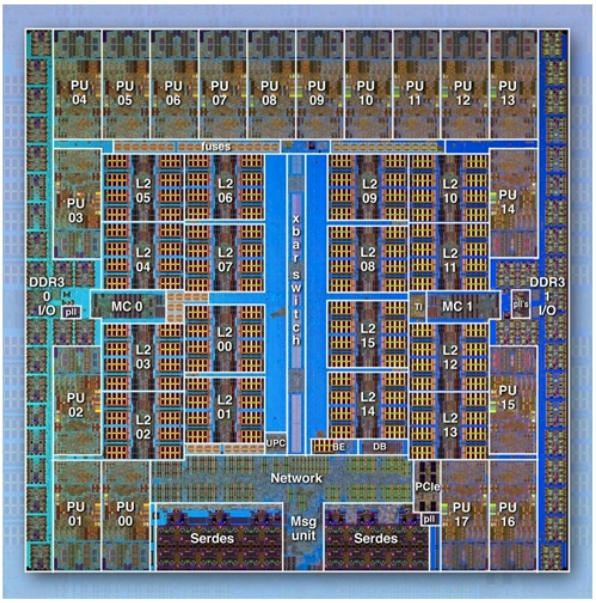
Cœurs/nœuds de calcul

Caractéristiques d'un nœud de calcul

Cœur	PowerPC A2 64-bit in-order
Gravure	45 nm
Cœurs par nœud	16+1+1, SMT 4
Fréquence horloge	1,6 GHz
Cache L1 privé par cœur	L1i : 16 kio + L1d : 16 kio
Cache L1P prefetching privé par cœur	32 x 128 octets
Cache L2 partagé	32 Mio eDRAM
Mémoire	16 Gio
Puissance crête par nœud	204,8 Gflop/s
Bande passante mémoire	40,7 Gio/s
Bande passante réseau	20 Go/s entrants et 20 Go/s sortants
Consommation électrique	ca 55 W

Cœurs/nœuds de calcul





Cœurs/nœuds de calcul

Opérations par cycle et surcharge cœurs

- Max 2 opérations par cycle : 1 en virgule flottante et 1 autre (entier, *load* ou *store* en mémoire ou contrôle)
- Opérations en virgule flottante : unité QPU (vecteur de 4 réels double précision et FMA) => max 8 FLOP/cycle
- Jusqu'à 4 *threads hardware* par cœur (1 *thread hardware* peut être 1 processus MPI)
- Max 1 opération par cycle pour 1 *thread hardware*
- => pour utiliser toutes les ressources de calcul, il faut plusieurs *threads* par cœur
- Gains fréquents au-delà d'un *thread* par cœur (mais moins de mémoire disponible par *thread*). 2 par cœur souvent l'optimum.

Cœurs/nœuds de calcul

Fonctionnalités particulières

- Support matériel des opérations atomiques (utile en OpenMP par exemple ; utilisation de façon explicite)
- Eveil rapide des *threads* endormis (plus de ressources disponibles pour les *threads* actifs si les autres sont en sommeil)
- Mémoire transactionnelle
  - élimine le besoin de verrous, le matériel détectant automatiquement les conflits entre *threads*
  - utile en OpenMP dans les régions *critical*
  - l'utilisation se fait de façon explicite par le développeur (pas automatique, ni portable) ou par des bibliothèques qui utilisent la mémoire transactionnelle
- Exécution spéculative des *threads* (proche de la mémoire transactionnelle)
- *Perfect Prefetcher* : possible d'enregistrer une séquence d'accès en lecture à la mémoire pour la rejouer plus tard et ainsi charger les données à l'avance
- Nombreux compteurs *hardware* (nombre d'opérations, cache misses, flux réseaux...)

Unité vectorielle QPU

Qu'est-ce que l'unité vectorielle QPU ?

- QPU (Quad Processing Unit) est une unité de calcul vectoriel en réels à virgule flottante double précision (64 bits).
- Cette unité peut réaliser des opérations identiques ou relativement proches en parallèle sur 4 flottants (par exemple : opérations sur des complexes, permutations, additions...).
- Approche de type SIMD.
- Exécute jusqu'à 4 opérations en virgule flottante par cycle. Si FMA (Fused Multiply-Add), 8 opérations/cycle.
- Peut gérer certaines exceptions IEEE (NaN et Inf).

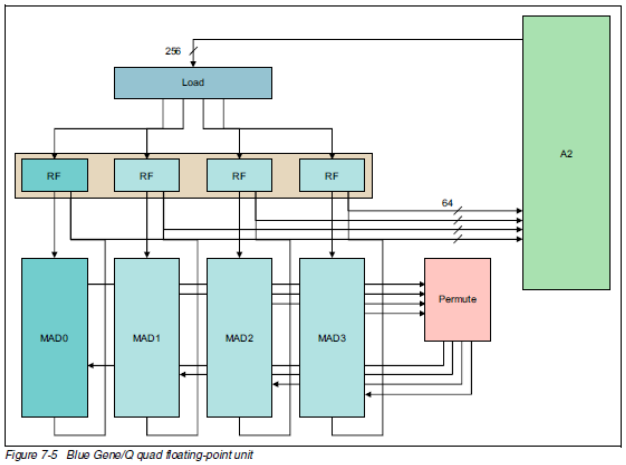


Figure 7-5 Blue Gene/Q quad floating-point unit

Caches et mémoire

Niveaux de caches et mémoire

Cache	Taille	Politique remplacement	Associativité
L1	16 kio-D + 16 kio-I	Pseudo Least Recently Used	8-way set-associative Lignes de 64 octets
L1P prefetch	32 x 128 octets	Depth stealing et round robin	Fully associative Lignes de 128 octets
L2	16 x 2 Mio	Least Recently Used	16-way set-associative Lignes de 128 octets
DRAM	16 Gio	-	Lignes de 128 octets

L'unité vectorielle QPU dispose également de 32 registres pouvant contenir chacun 4 flottants double précision.

Caches et mémoire

Cache L1 : caractéristiques

- 16 kio-D + 16 kio-I par cœur
- 8-way set-associative, lignes de 64 octets
- Remplacement par Pseudo Least Recently Used
- 1 bus (data) dans chaque sens entre L1 et cœur de 32 octets
- 1 bus (data) vers L1P de 32 octets (store) et 1 bus depuis L1P de 16 octets (load)
- Protection ECC contre les erreurs de parité

Cache L1 : performances

- Latence : 6 cycles (pour les entiers)
- Débit depuis et vers cœur : 32 octets/cycle à 1,6 GHz (48,8 Gio/s)
- Débit depuis L1P (lecture) : 16 octets/cycle à 1,6 GHz (24,4 Gio/s)
- Débit vers L1P (écriture) : 32 octets/cycle à 1,6 GHz (48,8 Gio/s)

Caches et mémoire

Cache L1P (unité de prefetching) : caractéristiques

- 32 x 128 octets par cœur
- Fully associative, lignes de 128 octets
- Remplacement par depth stealing et round robin
- 1 bus (data) depuis L1 de 32 octets (store) et 1 bus vers L1 de 16 octets (load)
- 1 bus vers switch L1P/L2 de 12 octets et 1 bus depuis switch de 32 octets
- Maximum 16 flux en prefetching (séquentiels avec stride max de 128 octets)
- List prefetching (possibilité d'enregistrer un pattern d'accès mémoire à rejouer, nécessité d'adapter le code)
- Protection ECC contre les erreurs de parité

Cache L1P (unité de prefetching) : performances

- Latence : 24 cycles entre L1 et L1P
- Débit vers L1 (load) : 16 octets/cycle à 1,6 GHz(24,4 Gio/s)
- Débit depuis L1 (store) : 32 octets/cycle à 1,6 GHz (48,8 Gio/s)
- Débit depuis L2 (load) : 32 octets/cycle à 800 MHz par cœur (24,4 Gio/s)
- Débit vers L2 (store) : 12 octets/cycle à 800 MHz par cœur (9,2 Gio/s)

Caches et mémoire

Cache L2 : caractéristiques

- 16 x 2 Mio par nœud de calcul
- 16-way set-associative, 1024 sets par bloc de 2 Mio, lignes de 128 octets
- Remplacement par Least Recently Used
- Write-back
- Unité de prefetching intégrée (pilotée par le L1P)
- 16 bus (1 par bloc de 2 Mio) vers switch L1P/L2 de 32 octets et 16 bus depuis switch de 12 octets (800 MHz)
- 2 bus vers DRAM de 16 octets (1 en load et 1 en store) (1,333 GHz)
- Protection contre les erreurs : ECC

Cache L2 : performances

- Latence : 82 cycles
- Débit vers L1P (load) : 32 octets/cycle à 800 MHz par cœur (24,4 Gio/s) (max 16 x 24,4 = 390,6 Gio/s)
- Débit depuis L1P (store) : 12 octets/cycle à 800 MHz par cœur (9,2 Gio/s) (max 16 x 9,2 = 146,5 Gio/s)
- Débit depuis et vers DRAM : 2 x 16 octets/cycle à 1,333 GHz (40,7 Gio/s)

Caches et mémoire

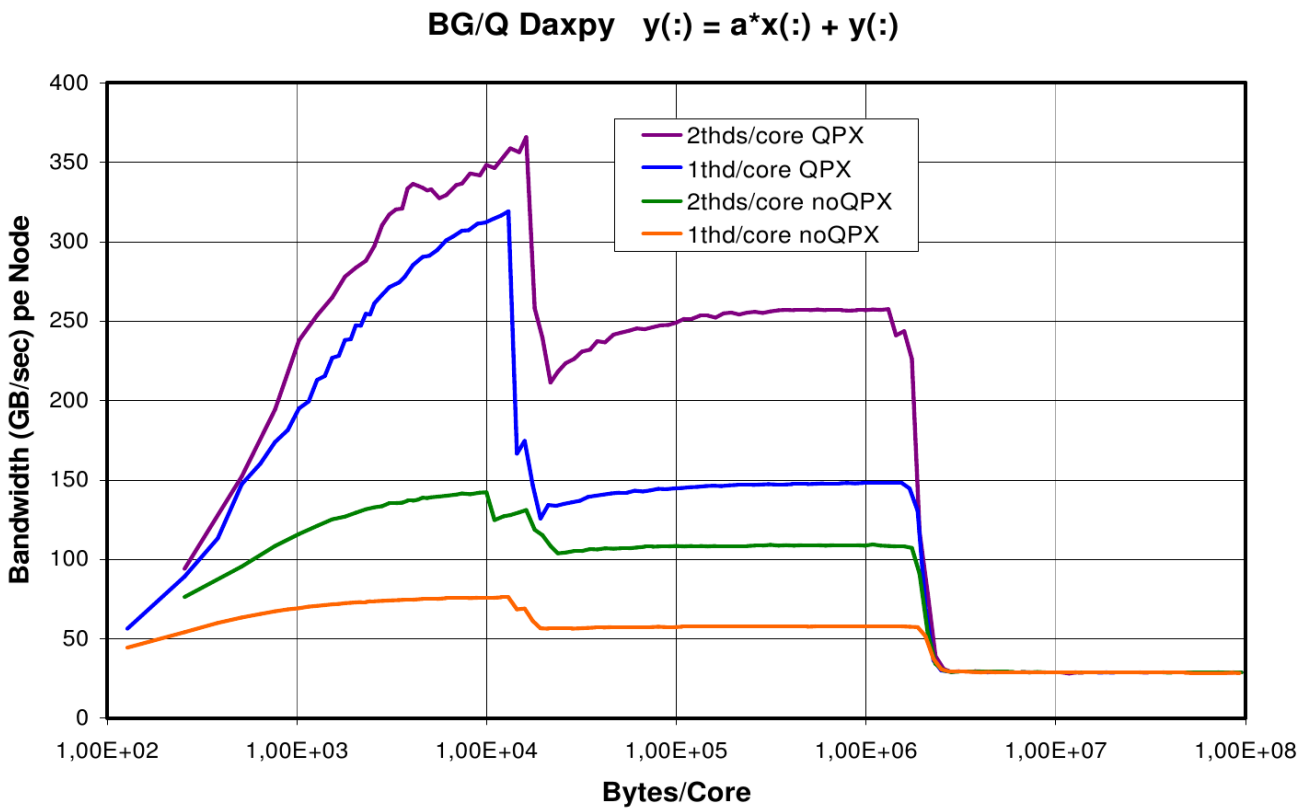
Mémoire DRAM : caractéristiques

- 16 Gio par nœud de calcul
- 2 bancs mémoire DDR-3 de 8 Gio
- 2 bus vers L2 de 16 octets (1 en load et 1 en store) (à 1,333 GHz)
- Protections contre les erreurs : ECC et Chipkill

Mémoire DRAM : performances

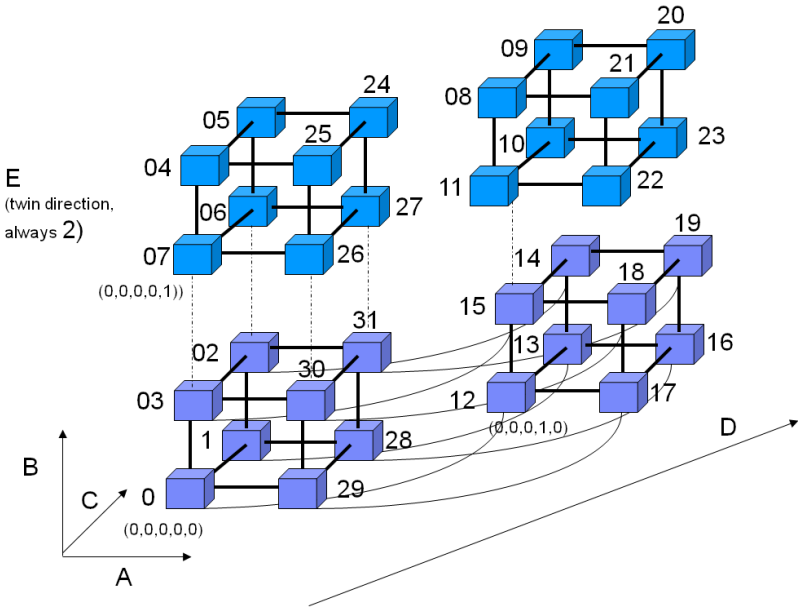
- Latence : 350 cycles
- Débit depuis et vers L2 : 2 x 16 octets/cycle (40,7 Gio/s)

Caches et mémoire



Réseaux

Tore 5D



- Connecte chaque nœud à ses 10 voisins
- 10 liens bi-directionnels (2 x 10 x 2 Go/s = 40 Go/s)
- 5ème dimension toujours de taille 2 et avec liens doublés
- Utilise un moteur DMA => recouvrement calculs/communications
- Selon le bloc d'exécution, chaque dimension peut être un vrai tore ou pas
- Calcul de certaines opérations de réduction sur les flottants par le réseau (somme, min, max, AND, OR et XOR)

Réseaux

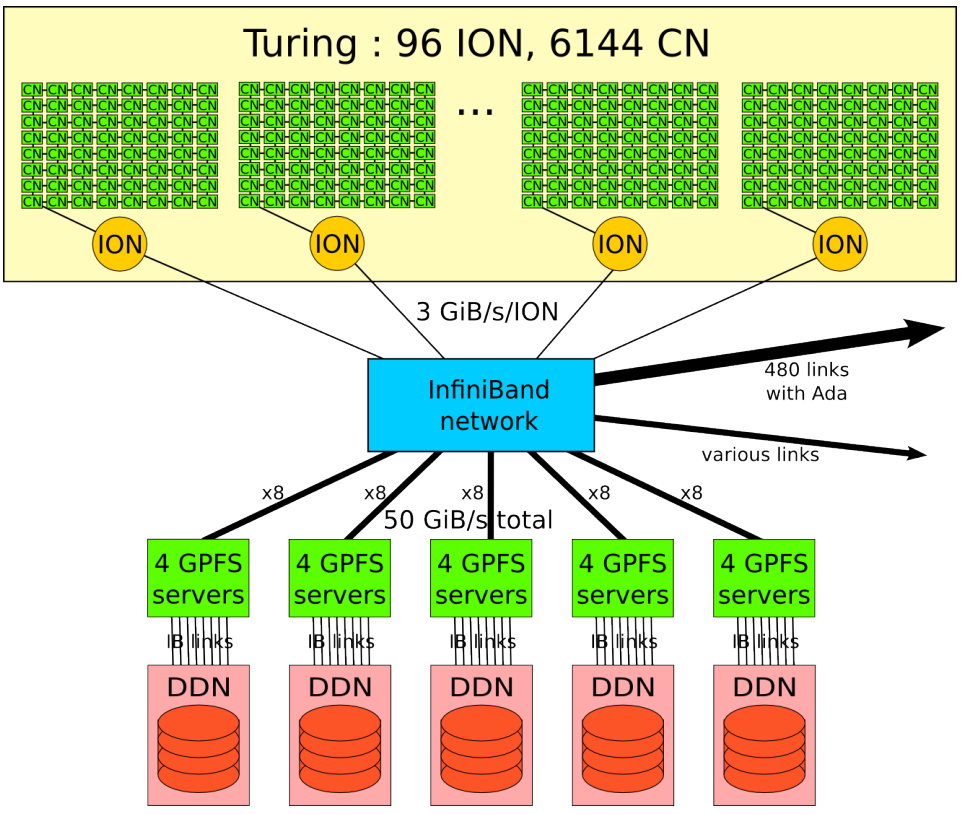
Configuration IDRIS

- 6 racks (2 midplanes par rack)
- 1 rangée de 6 colonnes
- Taille du tore 5D en CN : 12 x 4 x 8 x 8 x 2
- Taille du tore 5D en midplanes : 3 x 1 x 2 x 2

Autres réseaux

- 1 lien bi-directionnel, à 2 Go/s, qui relie un noeud de calcul sur 32 à un noeud d'I/O (2 CN par ION)
- JTAG : réseau de service
- Clock : pour l'horloge

Entrées/sorties



Entrées/sorties

Côté clients BG/Q (nœuds d'I/O)

- 1 I/O node tous les 64 compute nodes => 16/rack
- les ION gèrent tous les I/O (transparent côté CN)
- ION sont points de sortie des CN
- ION fournissent sockets aux CN
- ION : jusqu'à 3 Gio/s

Entrées/sorties

Côté serveurs I/O

- 20 serveurs GPFS
- 5 couplets DDN SFA10K (4 serveurs GPFS/baie) à 10 Gio/s/baie
- 2800 disques SATA 1 To (560/baie) en RAID6 8+2
- 2,2 Po utiles
- 1 Po pour le WORKDIR et 500 To pour le TMPDIR partagés entre Ada et Turing
- Taille de bloc (WORKDIR et TMPDIR) : 4 Mio
- Plus petite écriture : taille_bloc/32 (4 Mio / 32 = 128 kio)

Software

Système

Frontale

- Linux (RHEL 6.5)
- seul accès interactif pour utilisateurs
- compilateurs et débogueurs
- soumission des travaux

Noeuds de service

- gestionnaire de jobs
- base de données DB2 (comptabilité, problèmes détectés...)
- administration

Noeuds d'I/O

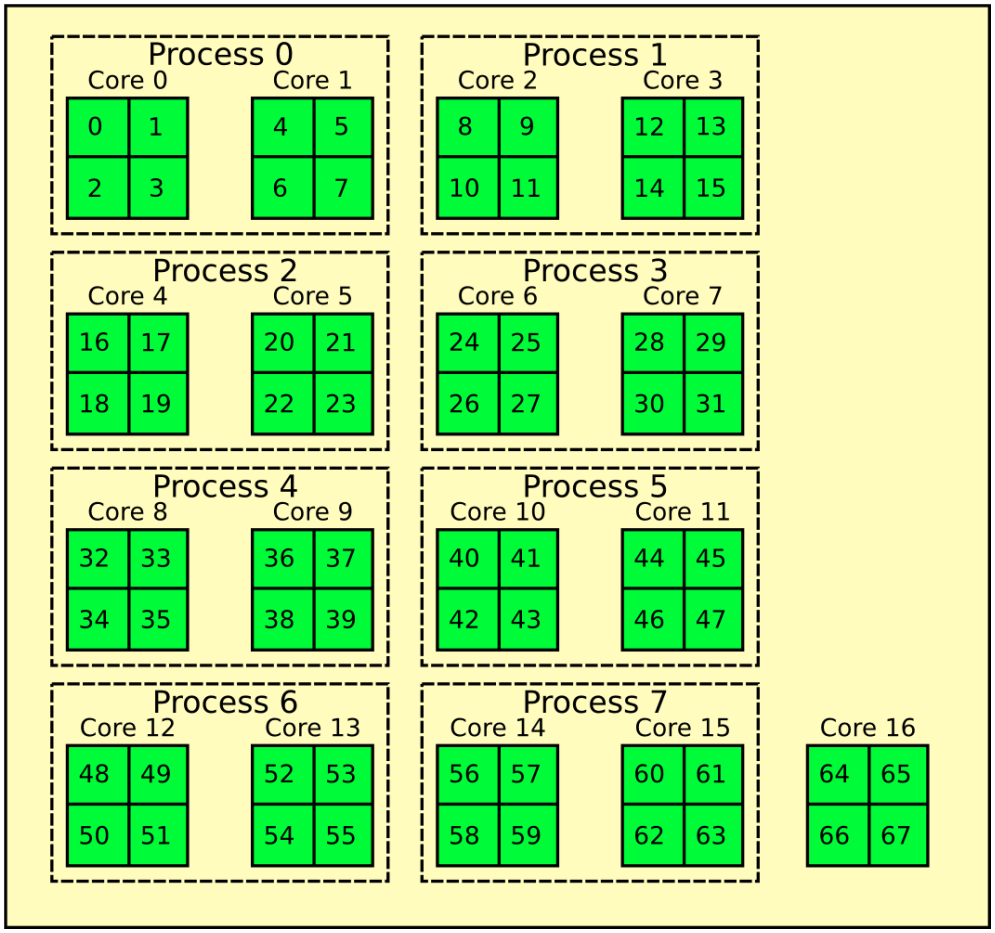
- Linux (RHEL 6 patché)
- CIOS : démon gérant les I/O
- au démarrage d'un job, reçoit une copie de l'exécutable et le lance sur les CN

Système

Nœuds de calcul

- CNK (Compute Node Kernel) : OS 64 bits allégé tournant sur le 17ème cœur
- binaire compatible avec linux (linux de l'ION)
- support des bibliothèques dynamiques
- paradigmes de programmation supportés (pas tous implémentés) : MPI, OpenMP, ARMCI, Global arrays, Charm++, UPC
- si 1 seul exécutable par CN => partage des zones text et read-only constant data
- gère 4 threads hardware par cœur (possible de mettre jusqu'à 5 threads utilisateur par thread hardware ; si plus d'un, switch à la main)
- mémoire virtuelle = mémoire physique
- pas de mémoire swap
- gestion basique de la mémoire => problèmes de fragmentation
- support complet NPTL (Native POSIX thread library ou pthreads)
- support de shared memory (POSIX)

Nombre de processus par nœud



Compilation

Compilateurs IBM

- IBM XL C/C++ Advanced Edition for Blue Gene/Q, V12.1
- IBM XL Fortran Advanced Edition for Blue Gene/Q, V14.1

	Frontale	**Cross-compilateur**
C	xlc, xlc_r	bgxlc_r, mpixlc_r
C++	xlc++, xlc++_r	bgxlc++_r, mpixlc++_r
Fortran	xlf, xlf90, xlf95, xlf2003, xlf2008 et versions réentrantes	bgxlf90_r, bgxlf95_r, bgxlf2003_r, bgxlf2008_r, mpixlf90_r, mpixlf95_r, mpixlf2003_r, mpixlf2008_r

Compilation

Compilateurs GNU

- Frontale : GNU Compiler Collection 4.4.7
- Cross-compilateur : GNU Compiler Collection 4.4.7

	Frontale	**Cross-compilateur**
C	gcc	powerpc64-bgq-linux-gcc
C++	g++	powerpc64-bgq-linux-g++
Fortran	gfortran	powerpc64-bgq-linux-gfortran

Ces commandes se trouvent dans le répertoire /bgsys/drivers/ppcfloor/gnu-linux/bin/.

D'autres versions plus récentes non supportées par IBM sont disponibles dans /bglocal/cn/pub/gcc/.

Le compilateur clang est aussi disponible dans /bglocal/fe/pub/Clang/.

Compilation

Options optimisation

- *-O0 -qnohot -qnosmp -qstrict* : inhibe toute optimisation
- *-O2 -qnohot -qnosmp -qstrict* : optimisation faible (sans risques)
- *-O3* : optimisation moyenne (conseillée, valeur par défaut IDRIS). Risques de modification de la sémantique du programme
  Exemple : (2.*3.1)*4.2 peut être interprété comme 2.*(3.1*4.2).
- *-O4* : optimisation aggressive + IPA légère (à tester et résultats à vérifier)
- *-O5* : *-O4* + IPA
- *-qnostrict/-qstrict* : autorise (ou pas) les modifications sémantiques
- *-qhot* : (par défaut en *-O4* et *-O5*) : transformations d'ordre élevé

Compilation

Options débogage génériques

- *-g* (valeur par défaut IDRIS) génère une table des symboles (numéro de ligne, variables ...). Génère aussi les numéros d'instructions dans les tracebacks. Niveau d'optimisation conservé. Option conseillée et sans surcoûts.
- *-qfullpath* inscrit les chemins d'accès aux fichiers source et d'include dans les fichiers objets (doit être combinée avec *-g*).
- *-O0 -qnohot -qnosmp -qstrict* permet d'inhiber toute optimisation.
- *-qinitauto=motif* initialise au préalable la mémoire correspondant aux variables temporaires à l'aide du motif précisé (ou à zéro si *-qinitauto* seul) :
  - *motif*`=FF` pour mettre les flottants simple et double précision à `NaNQ`

Compilation

Options débogage génériques

- *-qflttrap=overflow:underflow:nanq:zerodivide:invalid:enable* permet de détecter les exceptions (overflow, underflow, NaNq, division par zéro et NaNs). Il faut obligatoirement spécifier *-qsigtrap* pour intercepter les exceptions.
- *-qsigtrap* intercepte le signal SIGTRAP émis suite à une exception ou à un débordement de tableau et interrompt l'exécution avec un fichier core.
- *-qkeepparm* oblige le compilateur à conserver dans le stack les valeurs des arguments d'appel (certaines optimisations peuvent changer cela). Celles-ci seront ainsi récupérables dans n'importe quel débogueur.

Compilation

Options débogage Fortran

- *-C* génère un signal SIGTRAP (à intercepter avec *-qsigtrap*) lorsqu'un tableau est adressé en dehors de ses bornes (bound checking).
- *-qsave* force le mode d'exécution statique. Sur Turing, par défaut, nous sommes en mode stack.
- *-qlanglvl* vérifie la conformité du source à une norme Fortran. Par exemple *-qlanglvl=95std* pour la norme Fortran 95.

Compilation

Options débogage C/C++

- *-qcheck=bounds* vérifie si l'on n'adresse pas un tableau en dehors de ses bornes (bounds checking).
- *-qcheck=nullptr* vérifie la validité des adresses contenues dans les variables de type pointeur (valorise par défaut tous les pointeurs à *NULL*).
- *-qcheck=divzero* permet de détecter les divisions entières par zéro.
- *-qcheck=all* regroupe les 3 options ci-dessus.
- *-qdbxextra* inclut toutes les définitions de type enum, struct, union dans les fichiers objet pour qu'elles puissent être exploitées par un débogueur (doit être combinée avec *-g*).
- *-qlanglvl* vérifie la conformité du source à une norme C donnée. Par exemple *-qlanglvl=stdc99* pour la norme standard ISO C99.

Compilation

Fichiers de listage

Générations de fichiers de listage (en .lst) :

- *-qsource* : source du programme compilé.
- *-qattr* : références de tous les identificateurs avec leurs attributs respectifs.
- *-qxref* : références croisées de tous les identificateurs.
- *-qlistopt* : liste des options actives lors de la compilation.
- *-qreport* : liste des transformations liées à l'optimisation (boucles, appels de fonctions ESSL, etc) ou à la parallélisation (requiert au moins une des options *-qhot* ou *-qsmp*) :
  - *-qreport=smplist* : pour savoir comment le programme est parallélisé,
  - *-qreport=hotlist* : pour savoir comment les boucles sont transformées.

Compilation

Options diverses

- *-qsmp=omp* : activation des directives OpenMP
- *-qsmp=auto* : auto-parallélisation (multithreads uniquement)
- *-qintsize=2/4/8* (Fortran uniquement) : force la réservation en mémoire sur des mots de 2/4/8 octets (4 par défaut) pour les entiers et logiques déclarés avec le type INTEGER ou LOGICAL (sans spécification du paramètre KIND=n)
- *-qrealsize=4/8* (Fortran uniquement) : force la réservation en mémoire sur des mots de 4/8 octets (4 par défaut) pour les entités déclarées avec les types REAL, DOUBLE PRECISION, COMPLEX et DOUBLE COMPLEX (sans spécification du paramètre KIND=n)
- *-qautodbl=dbl4/dbl8/dbl/...* (Fortran uniquement) : peuvent aussi être utilisées pour promouvoir les réels de façon plus sélective

Compilation

Conseils

- Toujours compiler en réentrant (commandes suffixées par *_r*).
- Toujours compiler en mode debug (*-g*) (pas de surcoûts).
- Commencer par un niveau d'optimisation faible (*-O2*) et monter progressivement en vérifiant les résultats et les gains de performance.

Soumission

Modes de soumission

Tout travail ou job peut être exécuté sur les nœuds de calcul en :

- batch avec runjob et autres commandes dans un script LoadLeveler

Il n'est actuellement pas possible de lancer un travail en interactif.

Soumission

Travaux parallèles simples

```
# @ job_name = job_simple
# @ job_type = BLUEGENE
# Fichier sortie standard du travail
# @ output = $(job_name).$(jobid)
# Fichier erreur standard du travail
# @ error = $(output)
# Temps elapsed maximum demande
# @ wall_clock_limit = 1:00:00
# Taille bloc d'execution
# @ bg_size = 64
# @ queue

runjob --ranks-per-node 32 --np 2048 : ./my_code my_args
```

A soumettre via : *llsubmit job.ll*

Soumission

Travaux multi-steps

- Travail découpé en plusieurs sous-travaux ou étapes (steps)
- Obligatoire en cas de phase séquentielle importante (transfert de fichiers...)
- Evite le gaspillage de ressources
- $TMPDIR conservé entre les steps
- Phases séquentielles exécutées sur la frontale

Soumission

Exemple travail multi-steps

```
#========== Global directives ========
#@ job_name = test_multi-steps
#@ output = $(job_name).$(step_name).$(jobid)
#@ error    = $(output)

#========== Step 1 directives ========
#===== Sequential preprocessing =====
#@ step_name = seq_preprocessing
#@ job_type  = serial
#@ class     = archive
#@ queue

#========== Step 2 directives ========
#============ Parallel step ==========
#@ step_name  = parallel_step
#@dependency = (seq_preprocessing==0)
# (executed only if previous step
#  completed without error)
#@ job_type   = bluegene
#@ bg_size    = 64
#@ wall_clock_limit = 1:00:00
#@ queue

#========== Step 3 directives ========
#===== Sequential postprocessing ====
#@ step_name  = seq_postprocessing
#@ dependency = (parallel_step >= 0)
# (executed even if previous step
#  completed with an error)
#@ job_type   = serial
#@ class     = archive
#@ queue
```

```
case $LOADL_STEP_NAME in
  #========== Step 1 commands =========
  #===== Sequential preprocessing =====
  seq_preprocessing )
    set -ex
    cd $tmpdir
    mfget input_par/parameters.nml
    mfget inputs/big_data.in
    ;;

  #========== Step 2 commands =========
  #============ Parallel step ==========
  parallel_step )
    set -x
    cd $tmpdir
    runjob --ranks-per-node 32 --np 2048 \
    --mapping ABCDET : \
    $LOADL_STEP_INITDIR/my_exec parameters.nml
    ;;

  #========== Step 3 commands =========
  #===== Sequential postprocessing ====
  seq_postprocessing )
    set -x
    ls $tmpdir
    cd $tmpdir
    tar cvf result.tar *.dat
    mfput result.tar test/result.tar
    ;;
esac
```

Soumission

Directives LoadLeveler obligatoires

- *# @ job_type = BLUEGENE* : le type du job. Seul *BLUEGENE* est accepté pour tourner sur les nœuds de calcul. *SERIAL* peut être utilisé pour les phases de pre/post-processing sur la frontale ou de transferts de fichiers ;
- *# @ bg_size* : taille du bloc d'exécution réservé. Cette variable ne peut prendre que certaines valeurs : 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 ou 4096. Pour un nombre de nœuds réservés inférieur à 64, les ressources réseaux (tore 5D) peuvent être partagées avec d'autres travaux. Il peut donc y avoir un impact sur les performances des communications MPI et sur les entrées/sorties (un seul nœud d'I/O pour 64 nœuds de calcul) ;
- *# @ wall_clock_limit* : durée maximale d'exécution du code. Elle est donnée soit au format HH:MM:SS, soit directement en secondes ;
- *# @ queue* : dernière directive LoadLeveler. Celles qui suivent ne seront pas prises en compte.

Soumission

Directives LoadLeveler optionnelles

- *# @ bg_connectivity* : Torus, Mesh, Either ou choix pour 4 premières dimensions tore 5D (par exemple : *Torus Mesh Mesh Torus*). Vrai tore 5D possible seulement pour multiples d'un midplane. Si <512CN, au mieux 2 dont la 5ème pour 64 nœuds, 3 pour 128 et 4 pour 256. Par défaut : Mesh ;
- *# @ job_name* : nom du travail (choix libre) ;
- *# @ output* : nom du fichier de sortie standard ;
- *# @ error* : nom du fichier d'erreur standard. Si vous voulez mettre l'erreur standard dans le même fichier que la sortie standard, il suffit de mettre la directive égale à *$(output)* ;
- *# @ notification* : si mis à la valeur *complete*, vous recevrez automatiquement un email récapitulatif en fin d'exécution.

Cette liste n'est pas exhaustive. La documentation IBM vous les fournira toutes (certaines ne sont pas adaptées à la machine ou sont interdites).

Soumission

Options indispensables de runjob

- *--np NN*, avec NN le nombre de processus MPI demandés.
- *--ranks-per-node PP*, avec PP le nombre de processus MPI par nœud de calcul demandé. Les valeurs possibles sont 1, 2, 4, 8, 16, 32 et 64.

Options utiles et usuelles de runjob

- *--mapping MAPPING*, avec MAPPING le type de positionnement des processus. Le défaut est ABCDET et est **la valeur conseillée**.
- *--envs "variables_environnement"*, pour passer des variables d'environnement à votre application.
- *--exp-env*, pour exporter une variable de l'environnement actuel vers le job.
- *--env-all*, pour exporter toutes les variables définies lors du lancement de l'exécutable sur Blue Gene.
- *--label*, pour placer devant chaque écriture de la sortie standard et de l'erreur standard une étiquette avec le rang du processus qui écrit.
- *--cwd*, pour changer de répertoire de travail (par défaut : répertoire de soumission)

Soumission

Commandes de contrôle

- *llsubmit* pour soumettre un travail en batch
- *llq [-u]* affiche des informations sur l'évolution et la consommation de tous les travaux batch sur la machine. L'option *-u mon_login* restreint cet affichage à vos propres travaux

```
rlab432@turing> llq -u rlab432
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
turing1.23129.0          rcib040     2/25 23:08 R  100 MRt2         turing1
turing1.23134.0          rcib040     2/25 23:09 R  100 MRt2         turing1
turing1.23184.0          rcib040     2/26 12:37 I  100 MRt2

3 job step(s) in queue, 1 waiting, 0 pending, 2 running, 0 held, 0 preempted
```

  La colonne ST (Status) indique si votre job est R (Running) ou en attente I (Idle). Les autres états courants sont Not Queued en dehors des files d'attente, Hold, Changing state quand un job ou une étape sont terminés mais en train de sortir des files d'attente.
- *llcancel* pour supprimer un travail.

Soumission

Comptabilité

idrjar affiche des informations concernant la consommation de vos travaux batch sur la machine Turing. Elle permet notamment l'affichage :

- de la durée d'exécution (temps elapsed) ;
- des dates de soumission, de démarrage et de fin d'exécution ;
- des ressources réservées.

Les informations pour un travail ne sont récupérables qu'à partir du lendemain de son exécution.

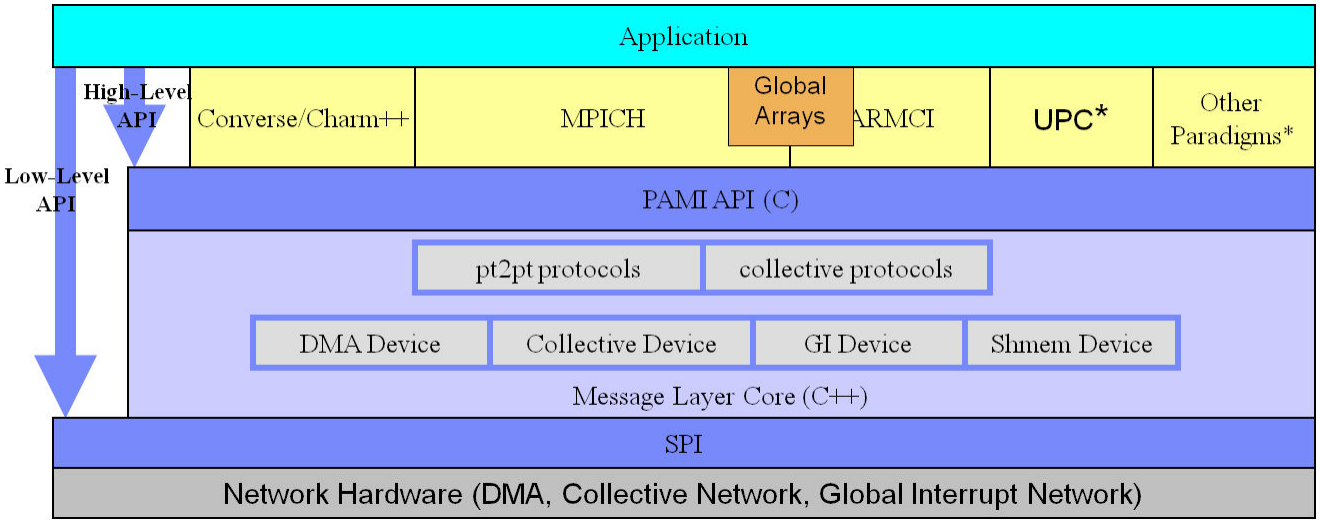
```
rlab432@turing1:~> idrjar
|-----------------------------------------------|
|--- IDRIS/CNRS. Version du  5 février 2014 ---|
|-----------------------------------------------|

Sorties concernant l'identifiant rlab432 pour la période du
        ==> 01 février 2013 au 06 février 2014

 Owner      Job Name          JobId      Queue tEse  #T  S
------- ---------------- --------------- ----- ---- ---- -
rlab432 hydro_bg64_64x1  turing1.19732.0 MRt1   540 1024 R
rlab432 hydro_bg64_64x32 turing1.19733.0 MRt1   125 1024 C
rlab432 hydro_bg64_64x1  turing1.19751.0 MRt1   541 1024 R
rlab432 hydro_bg64_64x2  turing1.19757.0 MRt1   541 1024 R
rlab432 test_multi-steps turing1.22978.1 MRt1     3 1024 C
-------------------------------------------------------------

  CONSOMMATION TOTALE DES TRAVAUX CI-DESSUS ==> 1792000s, soit 497.78h
...
```


Infrastructure de communications



MPI

Implémentation

- Norme MPI-2.2 (support partiel de MPI-3.0)
- Sauf gestion dynamique de processus
- Implémentation MPICH2-1.5 (Open Source)
- Optimisée pour utiliser le tore 5D
- Fonctions cartésiennes optimisées (MPI_Dims_create, MPI_Cart_create)
- Extensions BG/Q
- stdin seulement pour le rang 0

MPI

Communications point-à-point

Passent entièrement par le tore 5D.

Routage

- Statique/déterministe pour les petits messages : route connue à l'avance
  - Avantages : ordre des paquets respecté sans logique additionnelle et faible latence
  - Inconvénient : apparition de hot-spots dans le réseau si plusieurs communications simultanées passant par les mêmes liens
- Dynamique/adaptatif : route variant en fonction de la charge sur les liens
  - Avantage : charge réseau plus équilibrée
  - Inconvénient : latences plus importantes

Protocoles pour les messages point-à-point

- MPI immediate protocol (<=112 octets) : un seul paquet, routage statique
- MPI short protocol (<=496 octets) : un seul paquet, routage statique
- MPI eager protocol (<=2048 octets par défaut) : pas de négociation avec le destinataire, routage statique
- MPI rendezvous protocol (>2048 octets par défaut) : transfert négocié, routage dynamique, récepteur utilisant le moteur RDMA pour obtenir les données

Limite modifiable par variable environnement PAMID_EAGER (2049 octets par défaut)

Conseils

- Excellent recouvrement calculs-communications grâce au moteur DMA du tore 5D. **Attention :** pour l'utiliser pleinement, mettre *PAMID_THREAD_MULTIPLE=1* (*--envs "PAMID_THREAD_MULTIPLE=1"* pour *runjob*) ; peu efficace si 64 processus ou threads utilisateurs par nœud.
- Eviter d'utiliser les envois bufferisés et synchrones => utiliser le mode standard.
  (MPI_Bsend : implique des copies mémoires supplémentaires => long et occupe de la mémoire)
  (MPI_Ssend : opération non-locale => risque d'augmenter la latence)
- Poster les réceptions le plus tôt possible (avant l'envoi).
- Eviter les types dérivés non-contigus en mémoire.
  (car opérations de packing et donc copies mémoires et utilisation de mémoire pour le stockage + problèmes d'alignements mémoires non-optimaux)
- Faire attention à ne pas avoir trop de messages en cours de transfert simultanément à un endroit donné (via des MPI_Isend par exemple). Problème accentué pour protocole eager.

Conseils

- Ne pas saturer de messages certains processus (envoi d'un message par tous les processus à un seul autre). Risque de saturation mémoire.
- En envoi non-bloquant (MPI_Isend), ne jamais écrire dans l'espace mémoire envoyé avant de vérifier que l'envoi est terminé.
- Limiter le nombre de messages : il vaut mieux envoyer un gros message que de multiples petits.
- Utiliser le plus possible les communications collectives par rapport aux point-à-point (sauf si toutes les communications ont lieu avec des voisins directs).

Mapping/placement des processus

Définitions

- Chaque processus MPI a un rang dans le communicateur MPI_COMM_WORLD (allant de 0 à Nprocessus-1).
- Chaque processus MPI est placé de façon statique sur un cœur de la machine (pas de déplacements en cours d'exécution).
- Le placement ou mapping correspond à la relation entre le rang du processus et sa position sur la BlueGene/Q.

Importance

Plus le nombre de processus est élevé, plus il y a de communications et plus la distance moyenne (le nombre de liens à traverser) entre chaque processus augmente.

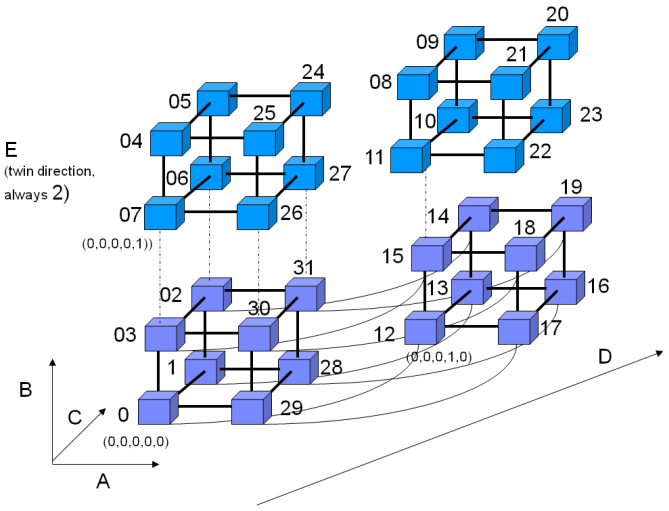
- Or, la latence augmente avec la distance ;
- la contention du réseau augmente si des messages traversent plusieurs liens.

L'impact d'un mauvais placement peut être très élévé.

Mapping/placement des processus

Topologie de la BlueGene/Q

- La topologie réseau pour les communications MPI est un tore 5D.
- Chaque nœud de calcul est connecté à ses 10 voisins par des liens réseaux bidirectionnels.
- L'idéal est de ne communiquer qu'avec ses voisins directs.
- L'impact sur les performances ne sera généralement visible qu'au-delà d'un midplane car tore 5D très compact.



Attention : la topologie ne peut être un vrai tore 5D que si vous utilisez un multiple de midplanes (512 CN).

Mapping/placement des processus

Topologies cartésiennes

Les topologies cartésiennes jusqu'à 6D peuvent être placées de façon optimale sur le tore 5D (6ème dimension au niveau des cœurs).

Adaptation logiciel

- Utiliser MPI_Dims_create et MPI_Cart_create et laisser le plus de liberté possible à MPI (ne pas imposer de dimensions).
- Connaître son application et ses modes/patterns de communications.
- Communiquer avec ses voisins les plus proches.
- Découper ses maillages pour coller au mieux à la machine.

Mapping/placement des processus

Utilisation fichier de mapping

- 6 coordonnées : A, B, C, D, E, T
- Placements prédéfinis : ABCDET + permutations (la dernière coordonnée varie en premier)
- Par défaut : ABCDET
- **Conseillé : ABCDET**
- La direction E est toujours de taille 2 (et est toujours un tore)
- Fichier de placement : 1 ligne par processus, 6 coordonnées par ligne (A B C D E T)

Placement effectué grâce à l'option *--mapping* de runjob.

Multithreading et OpenMP

Implémentation

- OpenMP (support complet du standard OpenMP 3.1)
- POSIX Threads (basé sur NPTL "Native POSIX Thread Library")
- Par défaut, maximum 4 threads par cœur

Approche hybride MPI/multithreads

- Un processus ne peut être décomposé en threads (ou processus légers) que dans un espace mémoire partagé.
- L'utilisation de plusieurs threads n'a un sens qu'en approche hybride sur la BlueGene/Q (multithreads à l'intérieur d'un nœud de calcul).

Avantages :

- Moins de communications
- Economies de mémoire
- Gains potentiels d'extensibilité

Multithreading et OpenMP

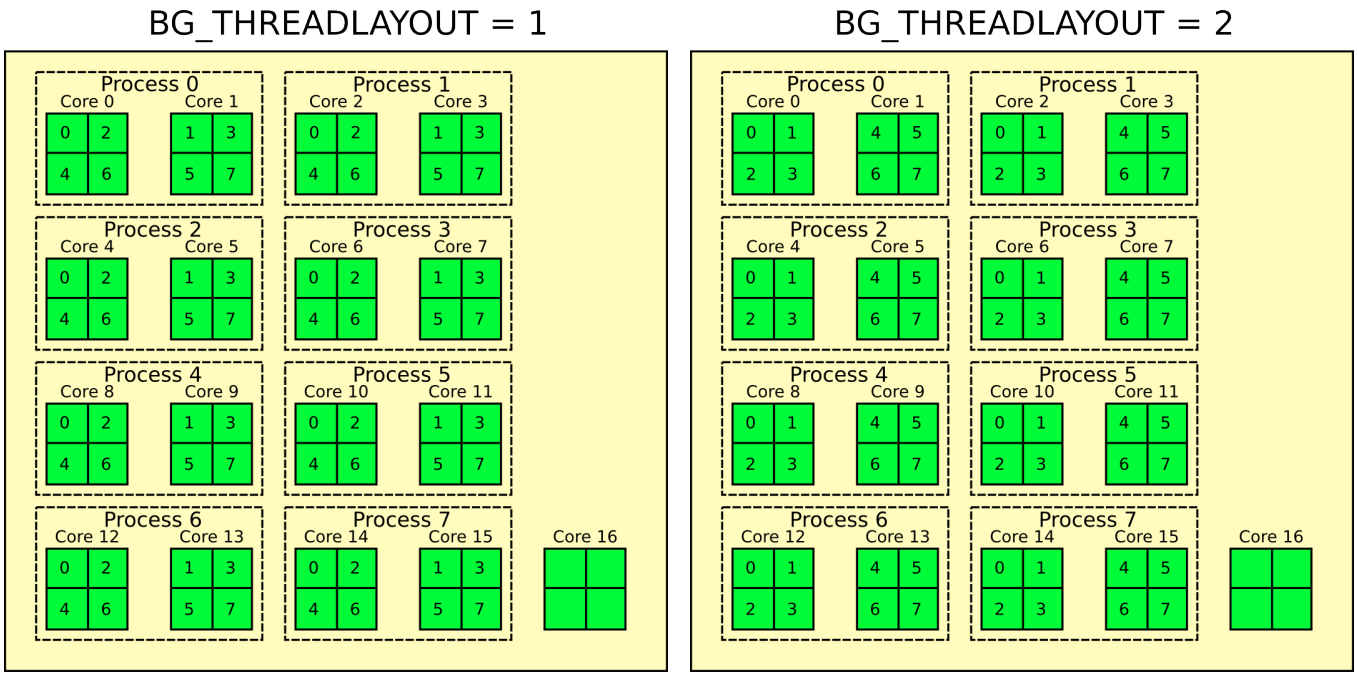
Compilation

- Toujours utiliser une version réentrante du compilateur (mpixlf_r, bgxlc_r...)
- Code OpenMP : utiliser l'option *-qsmp=omp*
- Auto-parallélisation (seulement multithreads) : utiliser l'option *-qsmp=auto*. Résultats non garantis (utile en phase de parallélisation OpenMP)
- Remplacer *MPI_Init* par *MPI_Init_thread*

Affinité

- Les *threads* sont toujours placés sur un cœur donné et ne peuvent se déplacer en cours d'exécution
- L'affinité est choisie à l'aide de la variable *BG_THREADLAYOUT*
  - Si *BG_THREADLAYOUT=1* (par défaut), *breadth-first assignment* : on place d'abord un *thread* sur un cœur, puis on passe au cœur suivant et on ne met un nouveau *thread* sur un cœur en ayant déjà un qui si tous les cœurs assignés au processus en ont déja un ;
  - Si *BG_THREADLAYOUT=2*, *depth-first assignment* : on remplit un cœur de *threads* avant de passer au cœur suivant.

Multithreading et OpenMP



Unité vectorielle QPU

Qu'est-ce que l'unité vectorielle QPU ?

- QPU (Quad Processing Unit) est une unité de calcul vectoriel en réels à virgule flottante double précision (64 bits).
- Cette unité peut réaliser des opérations identiques ou relativement proches en parallèle sur 4 flottants (par exemple : opérations sur des complexes, permutations, additions...).
- Approche de type SIMD.
- Exécute jusqu'à 4 opérations en virgule flottante par cycle. Si FMA (Fused Multiply-Add), 8 opérations/cycle.
- Peut gérer certaines exceptions IEEE (NaN et Inf).

Utilisation

- Le plus simple : utiliser les bibliothèques déjà optimisées.
- Passer l'option *-qsimd=auto* au compilateur (l'est par défaut).
- Utiliser les fonctions intrinsèques fournies par IBM (attention : non portable). Ces fonctions travaillent sur des vecteurs de 4 flottants double précision.

Unité vectorielle QPU

Alignement mémoire

Pour utiliser l'unité QPU, le compilateur doit pouvoir déterminer l'alignement des données. Un alignement mémoire naturel (càd sur 32 octets pour les double précision) est idéal.
Le compilateur peut déterminer l'alignement mémoire si :

- variable globale
- variable locale (hors pointeurs)
- Pour les pointeurs, on peut tester à la main l'alignement (en C/C++ uniquement : *if((int)x & 0x1F==0)*), utiliser *alignx* (si c'est aligné).
- Si IPA, le compilateur peut dans certains cas déterminer l'alignement (pas trop compter dessus). Le compilateur peut également faire du versioning.
- L'option *-qattr* permet d'obtenir certaines informations sur l'alignement mémoire.

Unité vectorielle QPU

Aider le compilateur

- Utiliser les pragmas spécifiant que les données sont alignées :
  - en C : *__alignx(alignement,adresse)*
  - en Fortran : *ALIGNX(alignement,nom_variable)*
- Préciser l'absence d'aliasing en C : *#pragma disjoint (a,b)* ou *restrict* en C99
- En Fortran, dire au compilateur que les tableaux à taille implicite sont contigus en mémoire (*-qassert=contig*)
- Eviter l'aliasing (qui est interdit en Fortran) et éventuellement passer par des variables locales
- Boucles avec nombre d'itérations connu dès la compilation ou aider le compilateur (*!IBM* assert(itercnt(100)* en Fortran et *#pragma ibm iterations(100)* en C/C++)
- Pour les boucles très courtes : désactiver la QPU
  - en C : *#pragma nosimd*
  - en Fortran : *!IBM* NOSIMD*
- Utiliser des pas de 1 pour les accès mémoire.

Unité vectorielle QPU

Aider le compilateur

- Ne pas utiliser de tableaux à taille implicite en Fortran (*dimension(*)*)
- Eviter les appels de fonctions dans les boucles (ou inliner)
- Organiser les données en groupes de 4 valeurs adjacentes (déclaration et utilisation)
- Eviter les dépendances entre itérations dans une boucle (code vectorisable !)
- Si pas de dépendances dans une boucle, le préciser dans le fichier source (*!IBM* INDEPENDENT* en Fortran et *#pragma ibm independent_loop* en C/C++)

Unité vectorielle QPU

Options du compilateur

- *-qsimd=auto* doit être positionné (valeur par défaut). Mettre à *-qsimd=noauto* pour la désactiver.
- Plus le niveau d'optimisation est agressif, plus il y a des chances qu'il arrive à utiliser la QPU.
- *-qflttrap=qpxstore:enable* permet la génération d'exceptions flottantes sur la QPU (« disable » par défaut). A utiliser avec *-qsigtrap* pour intercepter les exceptions.
- *-qreport -qattr -qsource -qlist* : créent un fichier .lst contenant de nombreuses informations sur la vectorisation

Bibliothèques

Bibliothèques scientifiques

Les bibliothèques sont installées dans les répertoires /bglocal/cn/pub et /bglocal/cn/prod et sont accessibles avec la commande *module*.

- ARPACK/PARPACK
- BLACS
- Boost
- ESSL
- FFTW
- GSL
- Hypre
- LAPACK
- MASS
- P3DFFT
- ParMETIS
- PETSc
- ScaLAPACK
- SCOTCH
- SLEPc

Bibliothèques

Bibliothèques d'entrées/sorties

Les bibliothèques sont installées dans les répertoires /bglocal/cn/pub et /bglocal/cn/prod et sont accessibles avec la commande *module*.

- HDF5
- netCDF
- Parallel netCDF
- SIONlib

Applicatifs

Liste applicatifs

Les applicatifs sont installés dans les répertoires /bglocal/pub et /bglocal/prod et sont accessibles avec la commande *module*.

- Abinit
- BigDFT
- CP2K
- CPMD
- Crystal
- GROMACS
- LAMMPS
- NAMD
- NWChem
- Quantum Espresso
- VASP

Outils

Outils

Les outils sont accessibles sur le chemin ou avec la commande *module*.

- addr2line
- CMake
- CVS
- FPMPI2
- Git
- GNU gprof
- GDB
- idrmemmpi
- libhpcidris
- PAPI
- SCALASCA
- Subversion (SVN)
- TAU
- TotalView

Outils : addr2line

Description

addr2line permet de retrouver la ligne dans les fichiers sources à partir d'une adresse dans la pile d'appels (*stack*).

Utilisation

- Chemin : /bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-addr2line
- Usage : addr2line -e nom_executable adresse_stack
- Compiler avec *-g* (par défaut à l'IDRIS)
- BG_COREDUMPONEXIT=1 pour forcer des cores à la fin de l'exécution

Outils : addr2line

Exemple

```
program addr2line
  implicit none

  integer :: i
  integer,parameter::n=5000,maxidx=10000

  integer,dimension(n) :: tab

  do i = 1,n
    tab(i)=2*i-1
  end do

  print *,'tab(maxidx)=',tab(maxidx)
end program
```

```
> bgxlf90_r -g test_addr2line.f90 -o test_addr2line
```


Outils : addr2line

Exemple

```
> runjob --ranks-per-node 16 --np 1 : ./test_addr2line

2013-03-07 12:56:27.784 (WARN ) [0xfff55408b30] :15869:ibm.runjob.client.Job: terminated by signal 11
2013-03-07 12:56:27.784 (WARN ) [0xfff55408b30] :15869:ibm.runjob.client.Job: abnormal termination by signal 11 from rank 0

> cat core.0
+++PARALLEL TOOLS CONSORTIUM LIGHTWEIGHT COREFILE FORMAT version 1.0
+++LCB 1.0
Program   : ./test_addr2line
Job ID    : 15869
Personality:
   ABCDET coordinates : 0,0,0,0,0,0
   Rank               : 0
   Ranks per node     : 16
   DDR Size (MB)      : 16384
+++ID Rank: 0, TGID: 1, Core: 0, HWTID:0 TID: 1 State: RUN
***FAULT Encountered unhandled signal 0x0000000b (11) (SIGSEGV)
...
...
...
+++STACK
Frame Address     Saved Link Reg
0000001fbfff6b40  0000000001012ce4
0000001fbfff6be0  0000000001000468
0000001fbfffbaa0  000000000105d328
0000001fbfffbd80  000000000105d624
0000001fbfffbe40  0000000000000000
---STACK

> powerpc64-bgq-linux-addr2line -e ./test_addr2line 0x01000468
/gpfs5r/workgpfs/idris/sos/ssos176/turing/formation_bg/trunk/demos/addr2line/test_addr2line.f90:13
```

Outils : GNU gprof

Description

GNU gprof est un outil de profilage d'applications. Ces fonctions sont :

- Identification des fonctions ou sous-programmes les plus consommateurs par une technique d'échantillonnage,
- Classement en fonction de leurs importances respectives,
- Arbre des appels.

Utilisation

- Compiler avec *-pg* (compilation et édition de liens)
- Introduit un peu d'overhead
- Un fichier gmon.out.x par processus
- Disponible via /bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gprof
- BG_GMON_RANK_SUBSET pour choisir les processus à suivre (par défaut rangs 0 à 31) (au format *N* ou *N* : *M* ou *N* : *M* : *S*)
- BG_GMON_START_THREAD_TIMERS pour les applications multithreadées (pas de suivi des *threads* par défaut, "all" pour les suivre tous, "nocomm" pour tous sauf les *threads* MPI)

Outils : GNU gprof

Exemple

```
> powerpc64-bgq-linux-gprof nom_executable gmon.out.0
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 52.31     45.36    45.36   400000     0.00     0.00  .__hydro_utils_NMOD_riemann
 13.27     56.87    11.51      640     0.02     0.13  .__hydro_principal_NMOD_godunov
  9.51     65.12     8.25   400000     0.00     0.00  .__hydro_utils_NMOD_trace
  6.50     70.76     5.64   400000     0.00     0.00  .__hydro_utils_NMOD_slope
  6.01     75.97     5.22   400000     0.00     0.00  .__hydro_utils_NMOD_constoprim
  4.99     80.30     4.33   500000     0.00     0.00  .__hydro_utils_NMOD_eos
  2.24     82.24     1.94   400000     0.00     0.00  .__hydro_utils_NMOD_cmpflx
  1.42     83.47     1.23      160     0.01     0.01  .__hydro_principal_NMOD_cmpdt
  0.74     84.11     0.64                             .__mcount_internal
  0.42     84.47     0.36                             .PAMI_Context_trylock_advancev
  0.28     84.71     0.24                             .SpinWaitTaskSwitchBGQ
  0.27     84.94     0.23                             .PAMI::Interface::Context<PAMI::Context>::advance(...
...
...
...
```

Outils : GNU gprof

Exemple

```
granularity: each sample hit covers 2 byte(s) for 0.01% of 86.71 seconds

index % time    self  children    called     name
                0.00   10.45       1/8           ._xlsmpParRegionSetup_TPO [6]
                0.00   73.13       7/8           .ThdCode [3]
[1]     96.4    0.00   83.58       8         .hydro_main$$OL$$1 [1]
               11.51   69.92     640/640         .__hydro_principal_NMOD_godunov [2]
                1.23    0.87     160/160         .__hydro_principal_NMOD_cmpdt [10]
                0.05    0.00       8/8           .__hydro_principal_NMOD_init_hydro [30]
                0.00    0.00       8/8           .__omp_domain_decomposition_NMOD_init_omp [56]
-----------------------------------------------
               11.51   69.92     640/640         .hydro_main$$OL$$1 [1]
[2]     93.9   11.51   69.92     640         .__hydro_principal_NMOD_godunov [2]
               45.36    0.00  400000/400000      .__hydro_utils_NMOD_riemann [4]
                8.25    5.64  400000/400000      .__hydro_utils_NMOD_trace [5]
                5.22    3.46  400000/400000      .__hydro_utils_NMOD_constoprim [7]
                1.94    0.00  400000/400000      .__hydro_utils_NMOD_cmpflx [11]
                0.05    0.00  200000/200000      .__omp_domain_decomposition_NMOD_sync_x [29]
                0.01    0.00     640/640         .__hydro_utils_NMOD_make_boundary [37]
                0.00    0.00  200000/200000      .__omp_domain_decomposition_NMOD_sync_y [47]
                0.00    0.00     640/640         .allocate_work_space [48]
                0.00    0.00     640/640         .deallocate_work_space [49]
-----------------------------------------------
                                                <spontaneous>
[3]     84.3    0.00   73.13                 .ThdCode [3]
                0.00   73.13       7/8           .hydro_main$$OL$$1 [1]
-----------------------------------------------
               45.36    0.00  400000/400000      .__hydro_principal_NMOD_godunov [2]
[4]     52.3   45.36    0.00  400000         .__hydro_utils_NMOD_riemann [4]
-----------------------------------------------
                8.25    5.64  400000/400000      .__hydro_principal_NMOD_godunov [2]
[5]     16.0    8.25    5.64  400000         .__hydro_utils_NMOD_trace [5]
                5.64    0.00  400000/400000      .__hydro_utils_NMOD_slope [8]
-----------------------------------------------
...
...
...
```


Outils : SCALASCA

Description

SCALASCA est un outil graphique d'analyse de performances pour applications parallèles. Principales caractéristiques (sur Turing) :

- Support de MPI et des applications multithreadées
- Mode profil ou trace
- Identification/analyse automatique des problèmes courants de performance (en mode trace)
- Nombre de processus illimité
- Support préliminaire des compteurs hardware

Utilisation

- Compiler votre application avec *skin mpixlf95_r* (ou autre compilateur IBM)
- Exécuter en batch avec *scan runjob*. Option -t pour le mode trace.
- Visualiser les résultats avec *square*

Outils : SCALASCA

Exemple

Soumission du job suivant (exécutable compilé en remplacant *mpixlf95_r* par *skin mpixlf95_r*) :

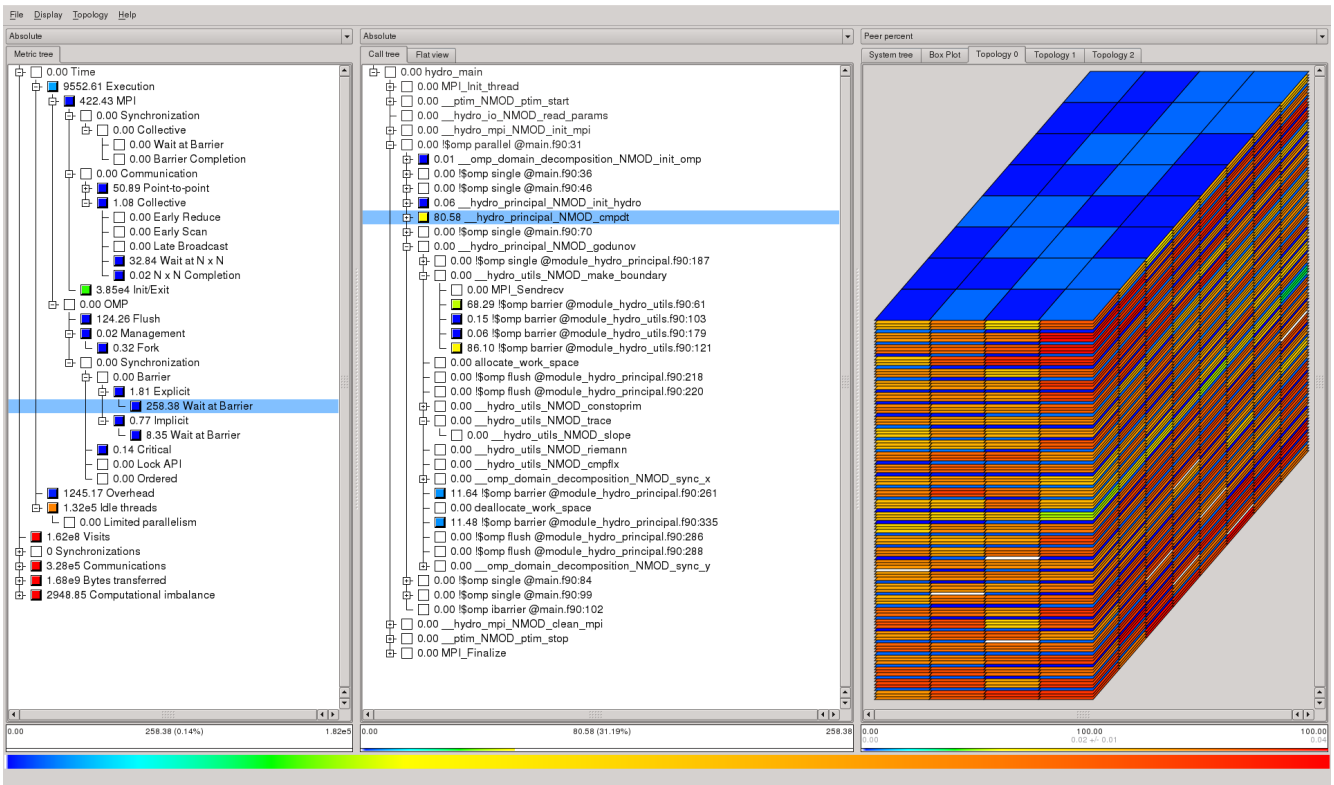
```
# @ job_name = scalasca_run
# @ job_type = BLUEGENE
# @ output = $(job_name).$(jobid)
# @ error = $(output)
# @ wall_clock_limit = 1:00:00
# @ bg_size = 64
# @ queue

module load scalasca
scan runjob --ranks-per-node 8 --envs "OMP_NUM_THREADS=8" \
            --np 512 : ./my_appli my_args
```

Visualisation en tapant la commande :

```
square repertoire_sortie_scalasca
```


Outils : SCALASCA



Outils : TotalView

Description

TotalView est un outil de débogage graphique d'applications parallèles. Principales caractéristiques (sur Turing) :

- Support de MPI et des applications multithreadées
- Support des langages C/C++ et Fortran95
- Débogueur mémoire intégré
- Jusqu'à plus de 4096 processus (limité par licence)

Utilisation

- Compiler votre application avec *-g* (par défaut à l'IDRIS, *-qnooptimize* pour aider TotalView mais pas obligatoire, *-qfullpath* peut s'avérer utile)
- Exporter votre DISPLAY vers votre ordinateur

Outils : TotalView

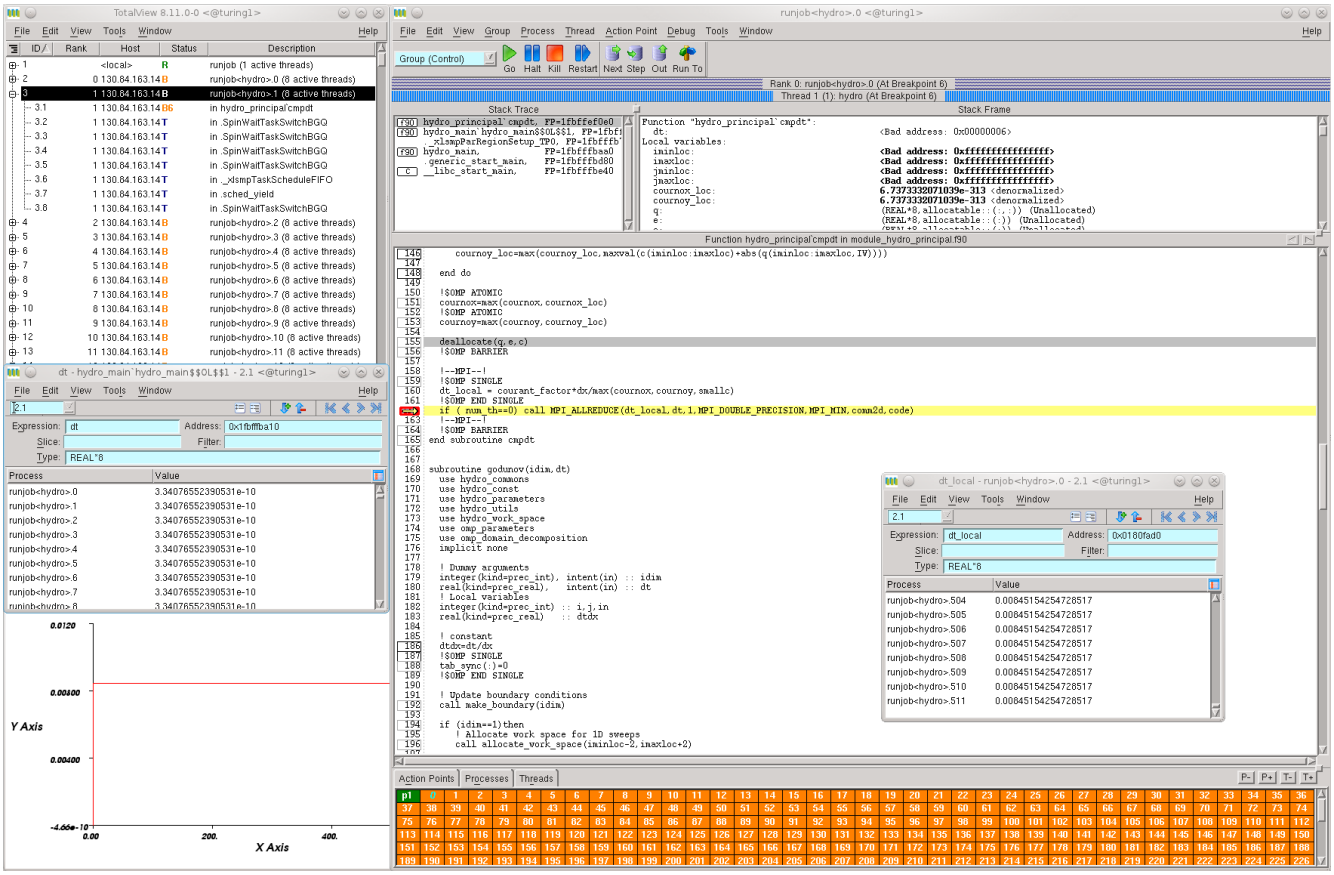
Exemple

```
# @ job_name = test_totalview
# @ job_type = BLUEGENE
# Fichier sortie standard du travail
# @ output = $(job_name).$(jobid)
# Fichier erreur standard du travail
# @ error = $(output)
# Temps elapsed maximum demande
# @ wall_clock_limit = 1:00:00
# Taille bloc d'execution
# @ bg_size = 64
# Environnement X
# @ environment = $DISPLAY
# @ queue

module load totalview

xterm -sb -e 'totalview runjob -a --ranks-per-node 8 --envs "OMP_NUM_THREADS=8" \
          --np 512 : ./mon_code mes_parametres'
```

Outils : TotalView



Outils : libhpcidris

Description

libhpcidris est une bibliothèque développée par l'IDRIS. Ces fonctions sont :

- Mesure de la consommation mémoire
- Mesure de temps
- Mesure des performances (Mflop/s, débits mémoires, débits réseaux...)

Utilisation

- Instrumenter le code
- Charger le module (commande : module load libhpcidris)
- Compiler
- Exécuter

Outils : libhpcidris

Exemple (consommation mémoire)

```
program test
 use hpcidris_mem
 use mpi
 implicit none

 integer::ierr,rank
 integer,dimension(1000) :: a
 real,dimension(:),allocatable :: b
 type(HPCIDRIS_F03_memusage)::mu

 call MPI_INIT(ierr)
 call MPI_COMM_RANK(MPI_COMM_WORLD,rank,ierr)

 if(rank==0) print *,'Programme commence'
 if(rank==0) print *,'Alloue tableau travail'
 allocate(b(1000000*(rank+1)))

 if(rank==0) then
  ! Le processus obtient son occupation
  ! memoire et affiche 2 de ses champs
  print *,'Mesure memoire et valeurs choisies'
  call HPCIDRIS_F03_memusage_get(mu)
  print *,'Processes per node :',mu%ppn
  print *,'Heapmax :',mu%heapmax

  ! Le processus 0 ecrira son occupation
  ! memoire a la fin de l'execution
  print *,'Force affichage occupation a la fin'
  call HPCIDRIS_F03_memusage_print_at_exit()
 end if

 a(1)=1;a(1000)=3
 b(1)=a(1000);b(1000000*(rank+1))=a(1)

 ! Affiche l'occupation memoire de tous les
 ! processus avec un niveau detaille
 if(rank==0) print *,'Affichage detaille'
 call HPCIDRIS_F03_MPI_memusage_print &
        (MPI_COMM_WORLD,2)

 if(rank==-1) then
  print *,a(1),a(1000)
  print *,b(1),b(1000000*(rank+1))
 end if

 deallocate(b)
 if(rank==0) then
  print *,'Desallocation tableau de travail'
 end if

 ! Affiche l'occupation memoire de tous les
 ! processus avec un niveau peu detaille
 if(rank==0) print *,'Demande affichage court'
 call HPCIDRIS_F03_MPI_memusage_print &
        (MPI_COMM_WORLD,0)
 call MPI_FINALIZE(ierr)
 if(rank==0) print *,'Programme termine'
end program test
```


Outils : libhpcidris

Exemple (consommation mémoire)

```
 Programme commence
 Alloue tableau travail
 Mesure memoire et valeurs choisies
 Processes per node : 8
 Heapmax : 11800576
 Force affichage occupation a la fin
 Affichage detaille
-------------------------------------------------------------------------------
                      MEMORY USAGE (libhpcidris version 4.0)
                    (C)2009-2013 Philippe WAUTELET, IDRIS-CNRS
-------------------------------------------------------------------------------
                      |  Max (position)   |  Min (position)   |  Average  |    Sum
-------------------------------------------------------------------------------
Used by application   |    1.9GiB(  511) |   19.6MiB(    0) |  998.1MiB |  499.0GiB
Available             |    2.0GiB(    2) |   14.5MiB(  504) |    1.0GiB |  521.4GiB
-------------------------------------------------------------------------------
Text                  |    6.6MiB(    0) |    6.6MiB(    0) |    6.6MiB |  424.0MiB
Data                  |    1.1MiB(    0) |    1.1MiB(    0) |    1.1MiB |  559.4MiB
BSS                   |  903.2KiB(    0) |  903.2KiB(    0) |  903.2KiB |  451.6MiB
Stack                 |   23.7KiB(    0) |   23.7KiB(    0) |   23.7KiB |   11.8MiB
Heap                  |    1.9GiB(  511) |   11.0MiB(    0) |  989.5MiB |  494.7GiB
Reserved by kernel    |   16.0MiB(    0) |   16.0MiB(    0) |   16.0MiB |    1.0GiB
Shared                |   64.0MiB(    0) |   64.0MiB(    0) |   64.0MiB |    4.0GiB
Max heap              |    1.9GiB(  511) |   11.3MiB(    0) |  989.7MiB |  494.9GiB
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
 Rank      Used Available      Text      Data       BSS     Stack      Heap    Kernel    Shared     Guard   Persist  Max heap
    0   19.6MiB    1.9GiB    6.6MiB    1.1MiB  903.2KiB  23.7KiB   11.0MiB   16.0MiB   64.0MiB       0B        0B     11.3MiB
    1   27.3MiB    2.0GiB    6.6MiB    1.1MiB  903.2KiB  23.7KiB   18.6MiB   16.0MiB   64.0MiB       0B        0B     18.9MiB
    2   31.1MiB    2.0GiB    6.6MiB    1.1MiB  903.2KiB  23.7KiB   22.4MiB   16.0MiB   64.0MiB       0B        0B     22.7MiB
...
```

Outils : libhpcidris

Exemple (consommation mémoire)

```
 Desallocation tableau de travail
 Demande affichage court
-------------------------------------------------------------------------------
                        MEMORY USAGE (libhpcidris version 4.0)
                      (C)2009-2013 Philippe WAUTELET, IDRIS-CNRS
-------------------------------------------------------------------------------
                      |  Max (position)  |  Min (position)  |  Average  |    Sum
-------------------------------------------------------------------------------
Used by application   |   19.7MiB(    0) |   19.6MiB(    1) |   19.6MiB |    9.8GiB
Available             |    2.0GiB(    2) |    1.9GiB(    0) |    2.0GiB | 1010.6GiB
-------------------------------------------------------------------------------
 Programme termine

--------------------------------
 Program finished
 Memory usage at exit:
--------------------------------
          Memory usage
 libhpcidris version 4.0
 (C)2009-2013 Ph. WAUTELET
              IDRIS-CNRS
--------------------------------
Used by application:     19.7MiB
Available:                1.9GiB
--------------------------------
Text:                     6.6MiB
Data:                     1.1MiB
BSS:                    903.2KiB
Stack:                   23.2KiB
Heap:                    11.1MiB
Reserved by kernel:      16.0MiB
Shared:                  64.0MiB
Max heap:                11.3MiB
--------------------------------
```


Outils : libhpcidris

Exemple (compteurs hardware)

Exemple de sorties des compteurs hardware (test avec le code RAMSES sur 8192 cœurs)

```
-----------------------------------------------------------------------
                PERFORMANCE (libhpcidris version 5.0)
              (C)2009-2014 Philippe WAUTELET, IDRIS-CNRS
-----------------------------------------------------------------------
 Elapsed time: 86.877202s (on process 0)
 Processes: 8192 (16 per node), threads per process: 1
 Reserved nodes (bg_size): 512 (IO nodes: 8)
 Torus dimensions: 4x4x4x4x2, is a torus: 1 1 1 1 1
 WARNING: values per thread are extrapolated (only 1 thread per core is instrumented)
-------------------------------------------------------------------------------------------
                |        Sum        |          Average          |      Volume     |   %peak
-------------------------------------------------------------------------------------------
FLOP/s total    |    2.622TFLOP/s   |  320.069MFLOP/s  /thread  |  227.780TFLOP   |    2.50%
FLOP/s FPU      |    1.562TFLOP/s   |  190.708MFLOP/s  /thread  |  135.719TFLOP   |    5.96%
 NoFMA FPU      |  804.623GFLOP/s   |   98.221MFLOP/s  /thread  |   69.899TFLOP   |    6.14%
 FMA FPU        |  757.658GFLOP/s   |   92.488MFLOP/s  /thread  |   65.819TFLOP   |    2.89%
FLOP/s QPU      |    1.060TFLOP/s   |  129.361MFLOP/s  /thread  |   92.061TFLOP   |    1.01%
 NoFMA QPU      |  531.613GFLOP/s   |   64.894MFLOP/s  /thread  |   46.182TFLOP   |    1.01%
 FMA QPU        |  528.111GFLOP/s   |   64.467MFLOP/s  /thread  |   45.878TFLOP   |    0.50%
FLoat inst/s    |    1.465TFLINS/s  |  178.856MFLINS/s /thread  |  127.284TFLINS  |   11.18%
Integer op/s    |  897.173GINTOP/s  |  109.518MINTOP/s /thread  |   77.940TINTOP  |    6.84%
Instructions/s  |    4.957TINST/s   |  605.120MINST/s  /thread  |  430.638TINST   |   18.91%
-------------------------------------------------------------------------------------------
L2 read         |    5.371TiB/s     |  687.446MiB/s    /thread   |   466.562TiB    |    2.82%
DDR read        |    3.004TiB/s     |    6.008GiB/s    /node     |   260.983TiB    |   15.12%
DDR write       |    2.727TiB/s     |    5.454GiB/s    /node     |   236.901TiB    |   13.73%
Network sent    |   20.062GiB/s     |   40.124MiB/s    /node      |     1.702TiB    |    0.21%
IO reads        |   89.249MiB/s     |   11.156MiB/s    /ION       |     7.571GiB    |    0.29%
IO writes       |    1.524GiB/s     |  195.052MiB/s    /ION       |   132.379GiB    |    5.11%
-------------------------------------------------------------------------------------------
```

Support avancé

Support avancé

Principes

L'IDRIS fournit un service de support avancé à ses utilisateurs. Sur la Blue Gene/Q, il fournit de l'aide pour :

- le portage des applications,
- leur optimisation,
- aider au passage à l'échelle (extensibilité, parallélisme massif).

Modalités

- Remplir le formulaire de demande,
- Evaluation technique par l'IDRIS,
- Support sur une période de temps déterminée (3 mois renouvelables) et sur une mission très précisément définie,
- Evaluation finale à la fin du projet.

Pour plus d'informations, voir le site web de l'IDRIS : www.idris.fr

Documentation

Documentation

Pour en savoir plus

- Le site web de l'IDRIS : http://www.idris.fr (section IBM Blue Gene/Q)
- *Best Practice Guide - Blue Gene/Q* : http://www.prace-ri.eu/Best-Practice-Guide-Blue-Gene-Q-HTML
- *Blue Gene/Q Application Development* : http://www.redbooks.ibm.com/abstracts/sg247948.html
- *Blue Gene/Q Code Development and Tools Interface* : http://www.redbooks.ibm.com/abstracts/redp4659.html
- Documentation des compilateurs IBM pour la Blue Gene/Q : http://pic.dhe.ibm.com/infocenter/compbg/v121v141/index.jsp
- *The Blue Gene/Q Compute Chip* : ftp://public.dhe.ibm.com/common/ssi/ecm/en/dcw03006usen/DCW03006USEN.PDF
- *The IBM Blue Gene/Q Interconnection Network and Message Unit* : http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=DCW03005USEN

Retours d'expérience

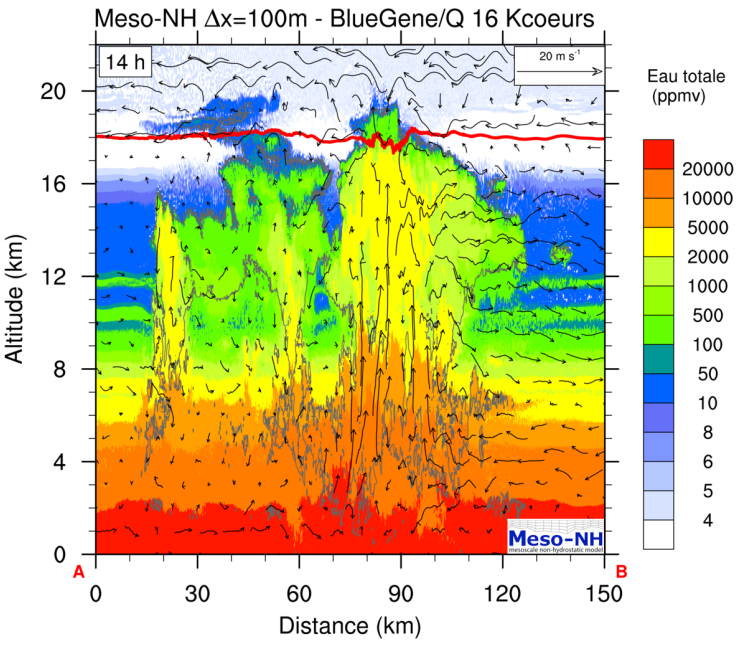
Méso-NH

Description projet

Modèle de recherche météorologique mésoéchelle communautaire français

- Impact du transport convectif sur la stratosphère tropicale
- Etude d'un orage (Hector se formant sur les îles Tiwi au nord de l'Australie)
- Maille la plus fine : cube de 100 m de côté (première mondiale pour ce type d'étude)
- Grille de 2560 x 2048 x 256 (1,34 milliards)
- Utilisation de 16.384 cœurs, 8 millions d'heures de calcul
- 20 Tio de données générées

Laboratoire d'Aérologie, université de Toulouse et CNRS



APAFA

Description projet

Etude de la propagation de la flamme inter-injecteurs sur le brûleur expérimental KIAI

- Réduction de la consommation de carburant et augmentation de l'efficacité de la combustion tout en garantissant la sécurité du rallumage de la chambre de combustion
- Méthode LES à l'aide du code AVBP
- Maillage de 38 millions de tétraèdes (résolution de 0,25 mm dans les zones critiques)
- Exécutions jusqu'à 16.384 cœurs

CERFACS, Toulouse

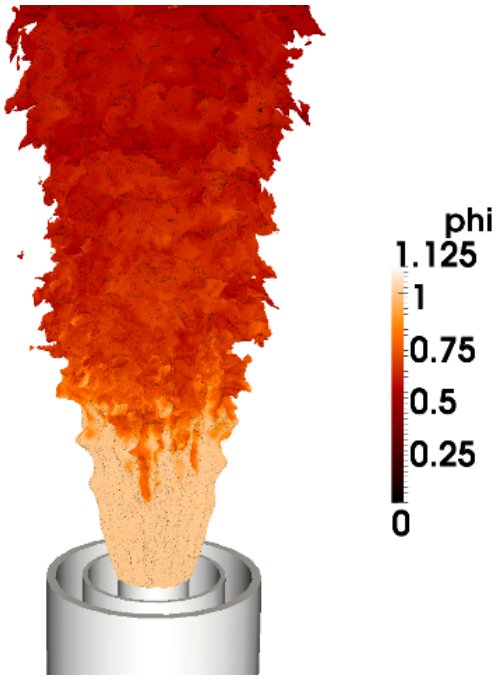
ECOPREMS

Description projet

Etude de la COmbustion PREmélangée en Milieu Stratifié

- Flammes prémélangées avec des proportions différentes carburant/comburant dans la chambre de combustion
- Simulations de 2 flammes de laboratoire pour comparer les résultats numériques et expérimentaux
- Code YALES2
- Exécutions jusqu'à 16.384 cœurs, 9 millions d'heures de calcul

CORIA, INSA et Université de Rouen, Saint-Etienne-du-Rouvray



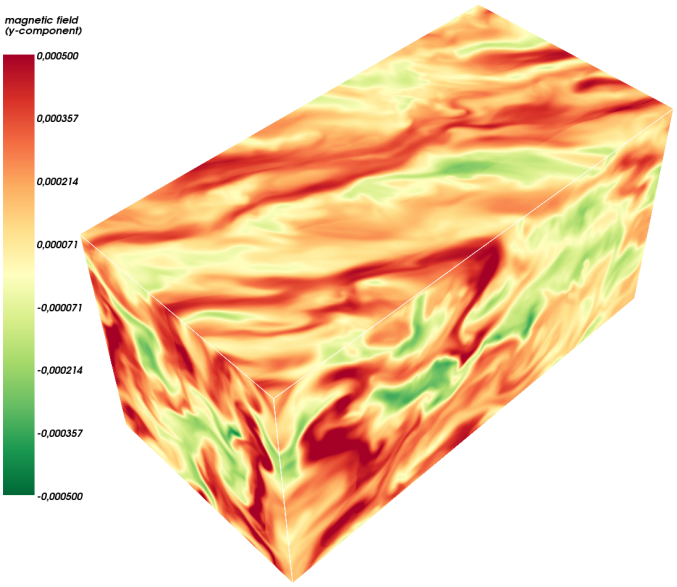
MHDTURB

Description projet

Turbulence magnétohydrodynamique dans les disques d'accrétion

- Détermination du taux de transport de moment cinétique résultant de l'instabilité magnétorotationelle
- Code RAMSES
- Maillage de 800 x 1600 x 832 (1 milliards de cellules), 800.000 pas de temps
- Exécutions sur 32.768 cœurs, 11 millions d'heures de calcul
- 200 Gio de données par sortie

Maison de la simulation, CEA, Université de Grenoble



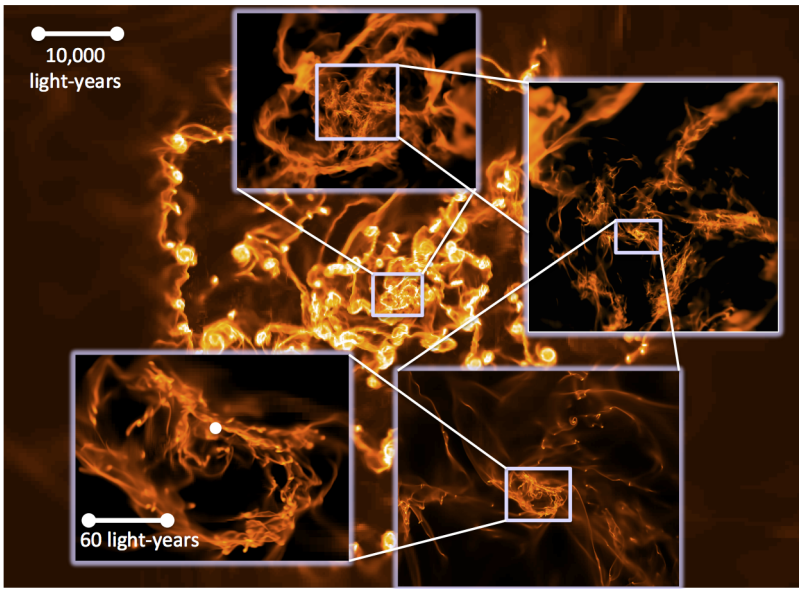
ZoomBHA

Description projet

Zoomed simulations of Black Hole Accretion

- Etude de la croissance des trous noirs
- Simulation d'une galaxie en forme de disque riche en gaz pour quelques centaines de millions d'années et zooms sur zones intéressantes
- Utilisation de 8.192 cœurs

CEA, DSM/IRFU/SAp



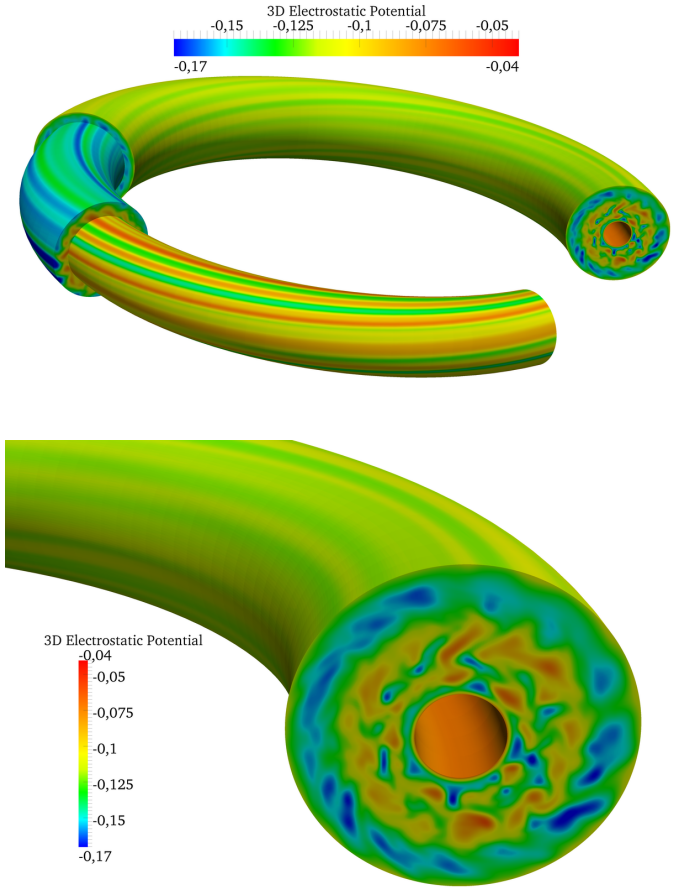
GYSELA

Description projet

Simulations multi-espèces turbulentes dans les plasmas de Tokamak

- Prédiction du ratio de la puissance générée par rapport à celle injectée
- Maillage de plus de 34 milliards de points (256 x 256 x 128 en espace et 128 x 32 en vitesse)
- 43.000 pas de temps pour 6,6 millions d'heures de calcul
- Exécutions jusqu'à 65.536 cœurs

CEA/IRFM, Maison de la simulation



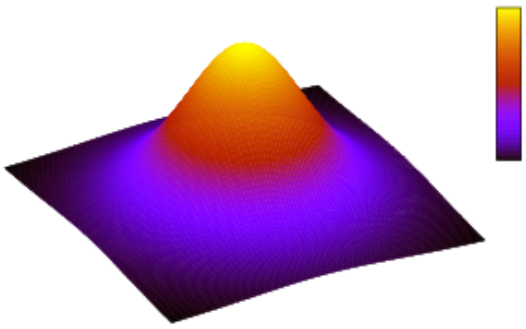
PrecLQCD

Description projet

Calculs de précision en QCD sur réseau

- Détermination de la constante de couplage forte α_s
- Exécutions jusqu'à 32.768 cœurs
- 17 millions d'heures de calcul

Laboratoire de physique subatomique et de cosmologie de Grenoble, Laboratoire de physique théorique d'Orsay, Laboratoire de physique corpusculaire de Clermont-Ferrand, Laboratoire de l'accélérateur linéaire d'Orsay et Fac. ciencias experimentales d'Huelva (Espagne)



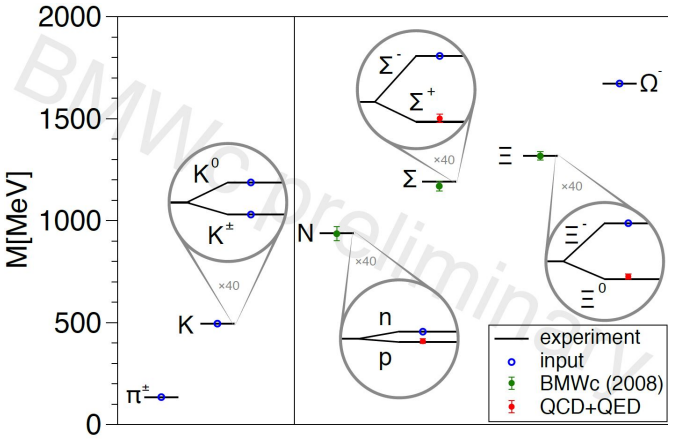
STABMAT

Description projet

Stabilité de la matière ordinaire

- Etude de l'electromagnétisme et de la différence de masse des quarks *u* et *d*
- Exécutions jusqu'à 65.536 cœurs
- 33 millions d'heures de calcul

Physique des particules élémentaires CNRS et Aix-Marseille U./Centre de Physique Théorique



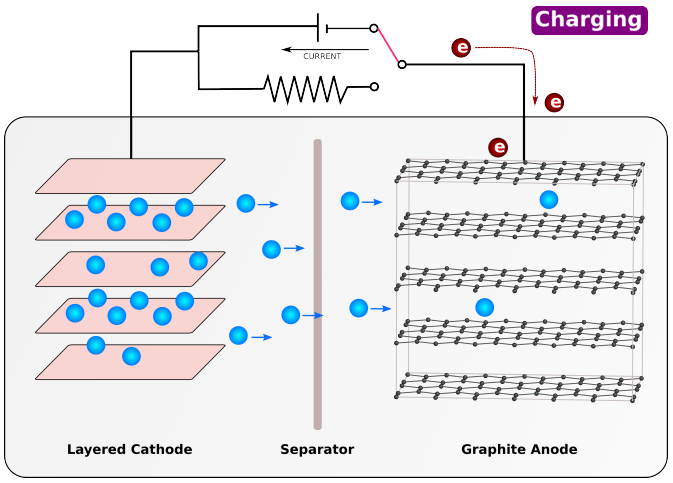
BigDFT

Description projet

Etude *ab initio* de la structure et de la cinétique des matériaux d'anode pour les batteries ion-lithium

- Analyse des propriétes de l'anode et des changements structuraux lors de l'utilisation de la batterie
- Systèmes de 208 à 625 atomes
- Exécutions jusqu'à 16.384 cœurs
- 15 millions d'heures de calcul

Laboratoire de Simulation atomistique, SP2M, INAC, CEA-UJF, Grenoble



Travaux pratiques

TP1 - Compilation et soumission

Objectif

Apprendre à compiler des applications sur Turing et à les exécuter sur les nœuds de calcul.

Enoncé

1. Compléter le Makefile afin de compiler l'application *poisson* ;
2. Ecrire un fichier de soumission pour exécuter *poisson* sur 1024 processus dans une configuration appropriée ;
3. *Optionnel :* Tester différentes options du compilateur et comparer les résultats ;
4. *Optionnel :* Tester différents nombre de processus par nœud, placements des processus, nombre de processus (ne pas utiliser plus de 256 nœuds de calcul)...

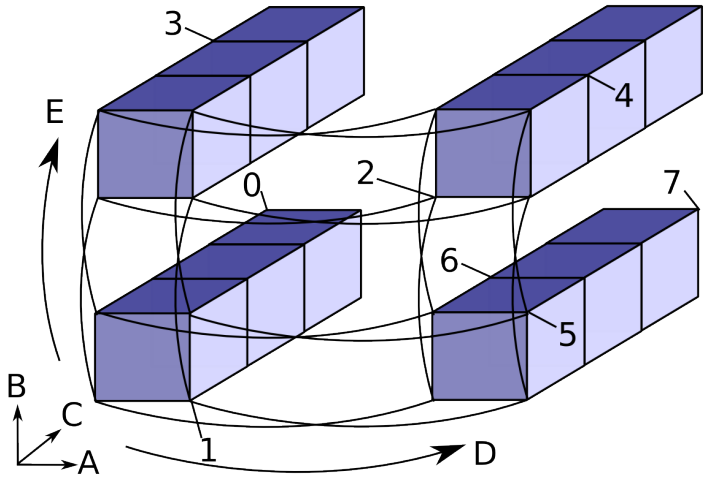
TP2 - Mapping

Objectif

Apprendre à placer les processus MPI sur les cœurs de Turing et à observer l'importance de ce placement sur les performances des communications.

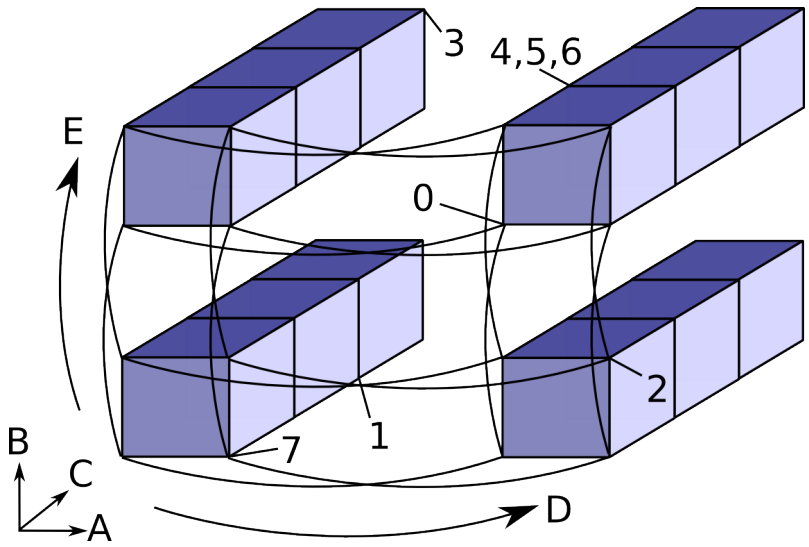
Enoncé

1. Pour les configurations suivantes (voir figures), écrire le fichier de placement (*mapfile*). Le programme à exécuter est celui compilé à partir de *map1.f90*. Le bloc utilisé contiendra 64 nœuds de calcul avec une topologie 5D 2x2x4x2x2 ;



TP2 - Mapping

Enoncé



2. Déterminer le placement optimum pour les deux applications *map1.f90* et *map2.f90* afin de minimiser les temps de communications, écrire les fichiers de placement correspondants et exécuter les applications ;
3. Comparer les résultats avec d'autres placements non-optimaux.

Note : Le nombre de processus est limité pour une meilleure compréhension.

TP3 - Unité vectorielle QPU

Objectif

Observer que l'utilisation de l'unité vectorielle QPU peut augmenter fortement les performances d'un noyau de calcul mais qu'il est souvent nécessaire d'aider le compilateur pour obtenir ces dernières.

Enoncé

1. Lire les sources du fichier *qpu.f90* et en particulier les 2 noyaux de calcul ;
2. Compiler et exécuter l'application *qpu* en activant ou non l'unité vectorielle ;
3. Essayer différentes options d'optimisations ;
4. Comparer et interpréter les résultats ;
5. *Optionnel :* Améliorer les performances des noyaux de calcul.

TP4 - Mesures de performance

Objectif

Apprendre à utiliser les outils de mesures de performance et de profilage et à diagnostiquer d'éventuels problèmes de performances.

Enoncé

1. Compiler et exécuter l'application *poisson*. Celle-ci doit être instrumentée de façon à utiliser la bibliothèque SCALASCA ;
2. Evaluer les performances de l'application, établir un diagnostic et identifier les parties critiques du code ;
3. Améliorer les parties critiques du code ;
4. Mesurer les améliorations éventuelles ;
5. *Optionnel :* Utiliser d'autres outils (GNU gprof...).

TP5 - Recouvrement calculs/communications

Objectif

Comprendre et implémenter le recouvrement des calculs par les communications afin d'optimiser le temps d'exécution d'un code et améliorer son extensibilité.

Enoncé

1. Compiler et exécuter le programme,
2. Observer la structure des échanges de messages et son importance sur le temps d'exécution,
3. Modifier le programme afin de permettre un recouvrement des calculs par les communications. Chiffrer le gain de performance.

TP6 - Approche hybride MPI/OpenMP

Objectif

Modifier une application MPI pour ajouter un niveau de parallélisation OpenMP à l'intérieur de chaque processus. Observer les gains d'extensibilité que rend possible cette approche hybride.

Enoncé

1. Ajouter des directives OpenMP dans le code *poisson* (uniquement dans la phase itérative) ;
2. Comparer les résultats de cette nouvelle version hybride avec la version purement MPI sur 64 nœuds de calcul.