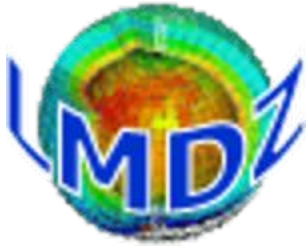
Hackathon GPU IDRIS/NVIDIA 2021



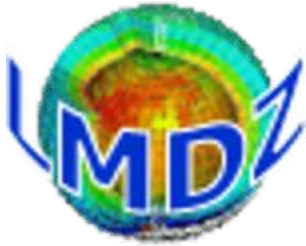
GPU:

> Graphical Process Unit: *processeur dédié au traitement des données graphiques. Ils sont utilisés pour accélérer certains types de calcul. Les machines HPC à venir (Exascale) seront très probablement basées sur cette architecture. Les GPUs sont très performants mais extrêmement spécialisés car dédiés au calcul seul. Les utiliser avec efficacité demande une gestion fine et contraignante des transferts de données entre CPUs et GPUs au niveau du code. Nos codes ont donc besoin d'un portage et d'une optimisation spécifiques vers ces architectures pour bénéficier de leur rapidité et parallélisme massif tout en tenant compte des contraintes sur la mémoire restreintes de ces processeurs.*

Principe du Hackathon:

> déf. dictionnaire : *Évènement au cours duquel des spécialistes se réunissent durant plusieurs jours autour d'un projet collaboratif de programmation informatique ou de création numérique.*

Hackathon GPU IDRIS/NVIDIA 2021

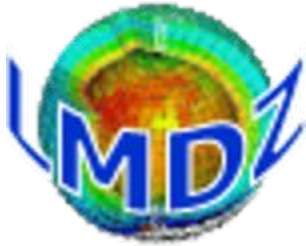


L'IDRIS et NVIDIA proposent un Hackathon GPU en mai 2021 afin de
permettre à des équipes de développeurs ou chercheurs d'améliorer les performances de leurs codes HPC ou IA sous la houlette d'un ou plusieurs mentors, experts en programmation GPU et issus des universités, de laboratoires nationaux, de centres de calcul, d'institutions gouvernementales et de constructeurs

L'équipe "GPU" LMDZ décide de candidater au hackathon avec pour objectifs:

- avoir une première approche du portage d'un code sur GPU
- se rassurer face à l'avenir (exascale) en démystifiant le GPU
- débuter avec un code simple mais en lien avec nos thématiques : la physique simplifiée de LMDZ
- développer un ensemble de règles simples pour porter la physique LMDZ sur GPU

Hackathon GPU 2021: méthode utilisée



Portage de la physique simplifiée LMDZ (une dizaine de routines) appelée soit

- par un driver autonome (une boucle sur le temps et un nombre de colonnes 1d avec longitudes et latitudes aléatoires, pour valider rapidement les modifications de code
- par le coeur DYNAMICO (pour les tests ‘grandeur nature’)

Programmation à l’aide du standard OpenACC. Le code reste compilable et exécutable en séquentiel/parallèle MPI/OMP, une option spécifique du compilateur permet de transférer des noyaux de calcul sur GPU.

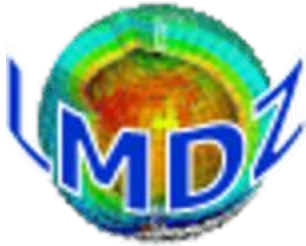
OpenACC est peu intrusif et fonctionne à base de directives simples intégrées dans le code.

(Une option radicale étant la ré-écriture du code dans un langage spécifique GPU.)

Fortran

```
!$acc parallel
  !$acc loop
  do i = 1, N
    a(i) = 0
  end do
!$acc end parallel
```

Hackathon GPU 2021: méthode utilisée (suite)



Pour la validation :

- Développement de fichiers de sorties spécifiques qui permettent de valider les modifications apportés au code et le passage sur GPU
- comparaison systématique CPU vs GPU (fortement facilitée par des options adaptées du compilateur!)

Une intégration continue sur gitlab (via gitlab-ci) qui permet à chacun :

- de travailler sur sa routine préférée
- de partager les modifications avec l'ensemble du groupe
- de garder un historique
- d'établir une chaîne de contrôle continu du code (DYNAMICO + physique) avec déclenchement automatique de tests lors d'un commit

Hackathon GPU 2021: le déroulé

Les membres de l'équipe : LMD (Jussieu, Polytechnique) / LSCE / IPSL
+ 2 mentors : IDRIS et NVIDIA





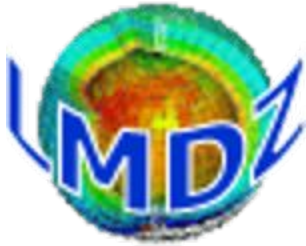




2 semaines de travail :

- lundi 17 mai: présentation de toutes les équipes et travail sur le code avec les mentors
- semaine du 17: travail sans mentors → début du portage des routines de la physique
- 25 au 27 mai: travail intensif avec les mentors: codage, data management, refactoring, optimisation (avec présentation des résultats du jour pour toutes les équipes)

Hackathon GPU 2021: OpenACC les recettes retenues



Utilisation des directives *descriptives* `!$acc kernels` :

- le compilateur analyse le code
- le compilateur fait la parallélisation
- gère si nécessaire le transfert de données

Gestion des données : Important pour le coût de calcul

- Copies de données sur GPU le plus en amont possible dans le code :

  `!$acc copyin / copyout / copy` (similaires aux `INTENT`)
- gestion des tableaux temporaires liés aux modules et présents uniquement sur le GPU :

  `!$acc declare create`

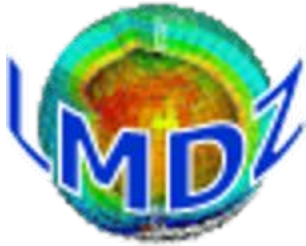
```
MODULE soil_mod

 REAL, ALLOCATABLE :: dz1(:),dz2(:)
  !$OMP THREADPRIVATE(dz1,dz2)
  !$acc declare create(dz1,dz2)
```

```
!$acc kernels default(none) present(dz1, dz2)
!$acc loop private(rk1, rk2)
  DO jk=1,nsoil
     rk1=jk
     rk2=jk-1
     dz2(jk)=fz(rk1)-fz(rk2)
  ENDDO
```

Hackathon GPU 2021: OpenACC

Par où attaquer le problème ?



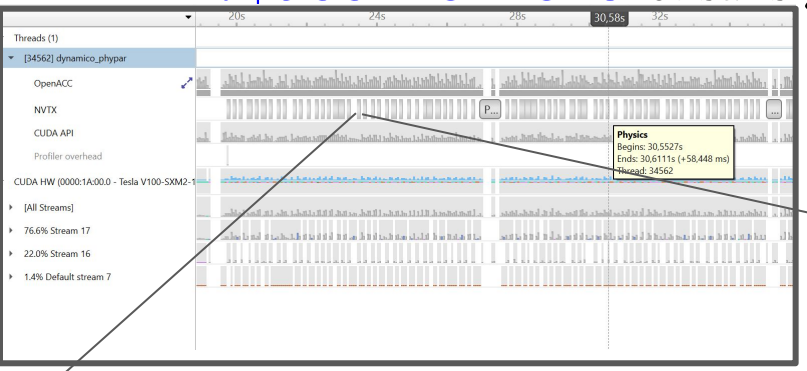
Hackathon (code léger) :
→ Utiliser un profiling pour déterminer la routine la plus coûteuse et attaquer le portage par elle.

In real life (LMDZ dans son intégralité) :
→ Commencer par le bas de l'arbre d'appels pour étendre petit à petit la zone de portage sur GPU
→ Cas des parties complexes (= non parallélisable en l'état) : faire un choix entre réécriture et laisser sur CPU

Hackathon GPU 2021: Profiling: nsight-sys

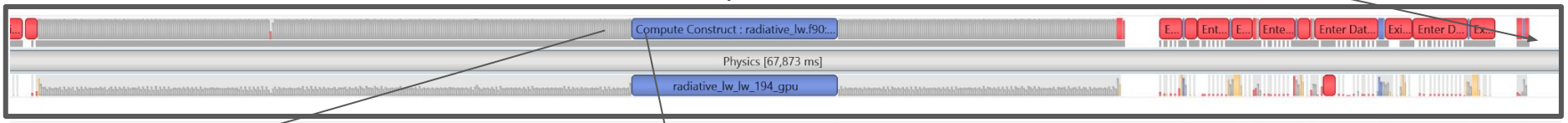
dégrossir le problème après instrumentation simple à base de :

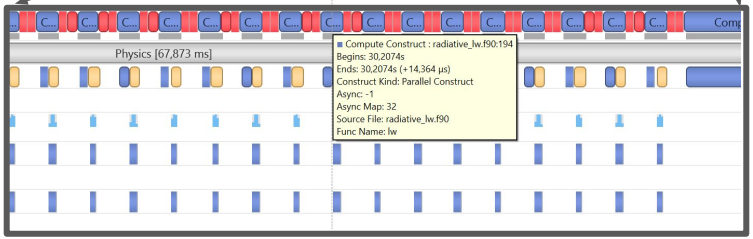
`!$acc kernels` et sans gestion de la mémoire



AVANT les modifications pour gestion de la mémoire

rayonnement

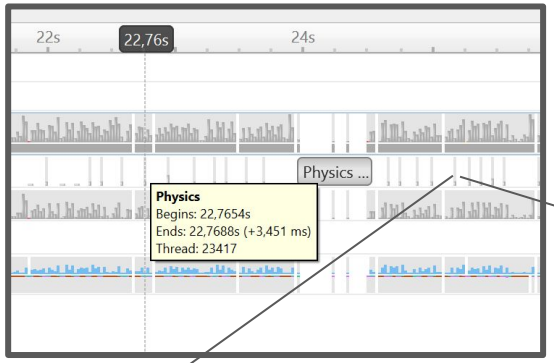




trop de transferts mémoire (rouge) coûteux entre CPU et GPU entre les noyaux de calcul (bleu)

Hackathon GPU 2021: Profiling: nsight-sys

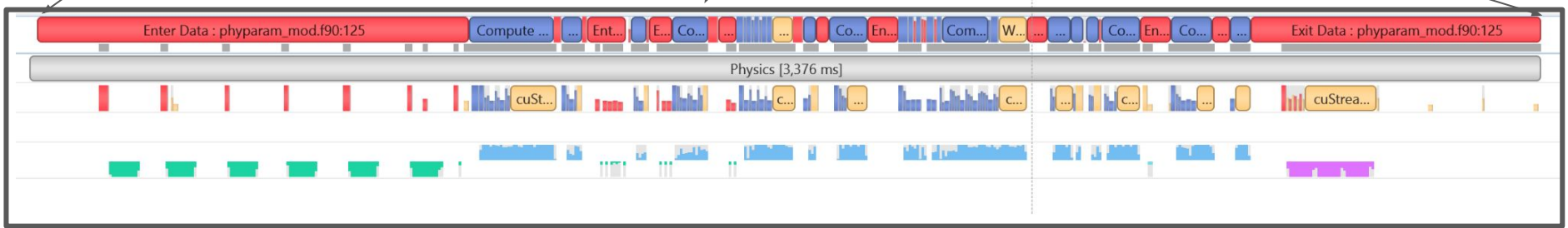
dégrossir le problème …. : gestion de la mémoire



APRÈS

Optimisation des transferts mémoire

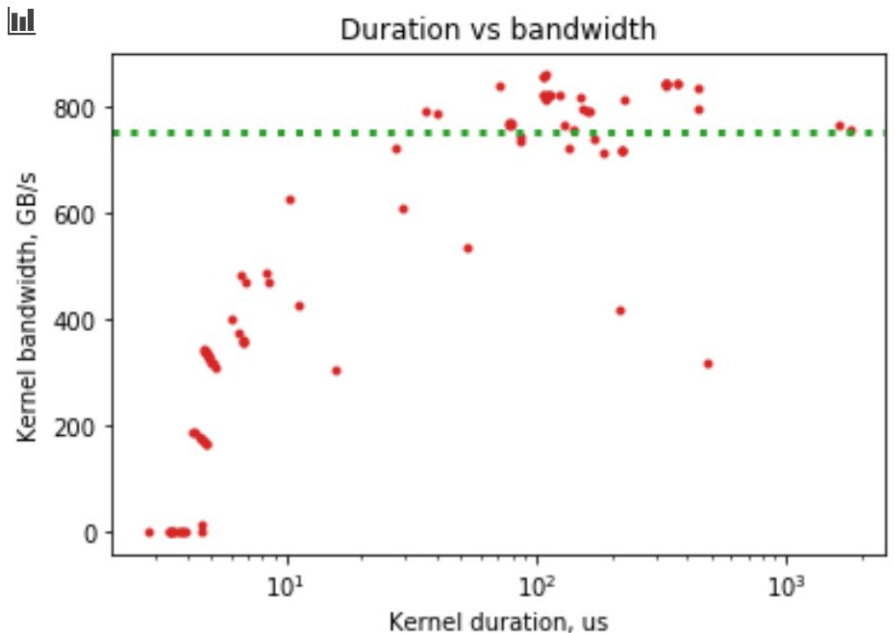
rayonnement



Hackathon GPU 2021: Profiling: nsight-compute

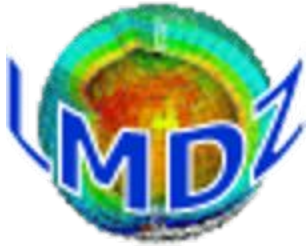
optimisation en détail ….

	Name	Repetitions	Bandwidth, GB/s	Duration, us	Improvement, us	Total improvement, us
4	convection_convadj_193_gpu	24	319.502379	479.706667	301.420896	7234.101507
35	radiative_lw_lw_197_gpu	24	758.853915	1792.396000	210.964026	5063.136633
31	radiative_lw_lw_145_gpu	24	767.258940	1612.544000	173.916484	4173.995609
77	turbulence_vdif_334_gpu	24	718.276346	219.882667	36.243693	869.848633
83	turbulence_vdif_372_gpu	24	718.853498	219.585333	36.	
73	turbulence_vdif_305_gpu	24	719.267925	219.518667	35.	
90	turbulence_vdif_k_111_gpu	24	796.529417	443.681333	32.	
66	turbulence_vdif_208_gpu	24	714.590359	185.441333	31.	
15	phyparam_mod_phyparam_178_gpu	24	740.266664	169.221333	23.	
43	radiative_sw_sw_142_gpu	24	535.422176	52.406667	19.	
45	radiative_sw_sw_162_gpu	24	755.940963	139.496000	16.	
48	radiative_sw_sw_193_gpu	24	764.241482	128.260000	14.	
7	geopot_221_gpu	24	789.992010	161.942667	13.	
87	turbulence_vdif_422_gpu	24	834.742208	445.357333	13.	
34	radiative_lw_lw_187_gpu	24	733.095637	86.557333	12.	
49	radiative_sw_sw_207_gpu	24	793.043420	158.793333	12.	
1	convection_convadj_167_gpu	24	815.048702	225.869333	11.	



Hackathon GPU 2021: résultats

(à consommer avec modération)



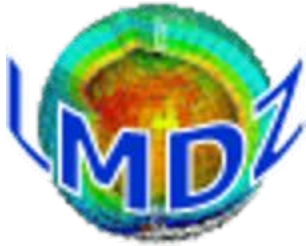
Après la traversée du désert les performances sont arrivées petit à petit … et au final :

(comparaisons idéalisées noeud à noeud)

							phys				
llm	ngrid (per GPU)	nbp	dt (dyn)	itau_phys	run_length	steps (phys)	CPU total	GPU total	CPU ms/step	GPU ms/step	speedup
79	4000	40	480	5	864000	360	5,633	0,971	15,65	2,70	5,80
79	16000	80	240	5	432000	360	24,841	2,841	69,00	7,89	8,74
79	64000	160	120	5	60000	100	31,483	2,652	314,83	26,52	11,87
79	256000	320	60	5	30000	100	170,93	10,054	1 709,30	100,54	17,00

driver with "fake" MPI parallelism							
llm	ngrid (total)	ngrid (per GPU)	nday	step_per_day	CPU Time	GPU Time	speedup
79	4000	1000	730	24	51,11515056	21,13881734	2,418070497
79	16000	4000	730	24	218,8372104	33,97676242	6,440790552
79	64000	16000	730	24	1146,126221	92,4700283	12,39456981
79	256000	64000	100	24	691,7642689	49,03794125	14,10671515
79	1024000	256000	25	24	891,6164964	50,79147975	17,55445009

Hackathon GPU 2021: Conclusions

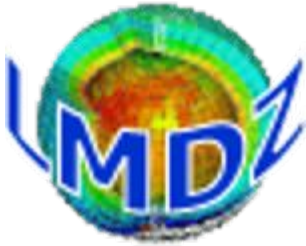


- L'exercice fut particulièrement instructif. Il montre qu'il y a du travail à faire pour porter un code sur GPU, et a fortiori pour l'optimiser pour en tirer un maximum de profit, mais que c'est accessible sur un code (simplifié) comparable à LMDZ

- Nous sommes rassurés sur le fait qu'en suivant un nombre limité de "recettes" simples, pour les règles de codage et pour la validation des développements, les non-spécialistes puissent continuer de coder dans LMDZ sans (trop) casser le portage

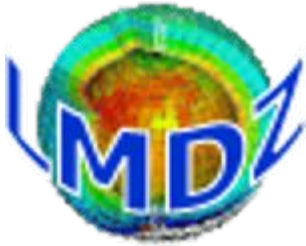
- Il semble évident que dans certains cas la complexité d'une paramétrisation donnée empêche un portage efficace (à moins peut-être d'une ré-écriture en profondeur). Reste à voir à quel point cela sera pénalisant sur un cas réel comme LMDZ.

Et maintenant ? LMDZ !...



- Que peut-on espérer d’un portage de LMDZ sur GPU ?
- Par quoi commencer ? Quand et qui ?
- Statut vis-à-vis du trunk LMDZ

Que vise-t-on ?



- Code unique CPU + GPU
- Ticket d'entrée sur les machines exascale
- Performance : *Nécessite le portage de la quasi-totalité du code critique*
- Certaines parties de LMDZ sont "importées" : essentiellement ECRad

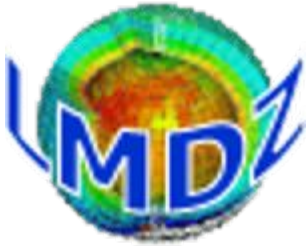
=> portage de toute la physique "maison"

- Toutes les variantes ? (rétro-compatibilité)
- Seulement CMIP6 voire CMIP7 ?

Pour la physique "importée" :

- se rapprocher des auteurs
- émuler (ML) ?

Par quoi commencer ? Quand, qui ?



- Instrumentation du code ?
- p.ex. “dump” des valeurs en entrée et sortie d’une routine pour vérification du portage

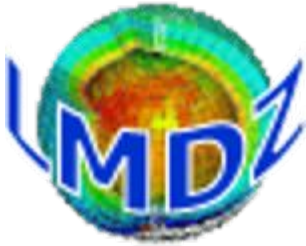
- Driver multi-colonne ?

- Dans quel ordre porter les différents modules ?
- Pas trop difficile tout de suite, mais aborder rapidement les difficultés
  - Turbulence, bucket
  - Convection (thermiques), ondes de gravité
  - Convection profonde, poches froides, précipitations
  - Isotopes ?

- Qui : besoin de tous, notamment experts du contenu physique
- On peut avoir besoin de reformuler

- Quand : automne ?

Portage GPU et trunk LMDZ



- Code unique CPU + GPU
- Le portage GPU prendra du temps, LMDZ a besoin de continuer d' évoluer

=> les dvp GPU doivent être régulièrement intégrés au trunk
s'attendre à des changements sur les parties portées
suivi à prévoir

=> pas de branche / répertoire spécifique ? Branches (svn/git) éphémères ?