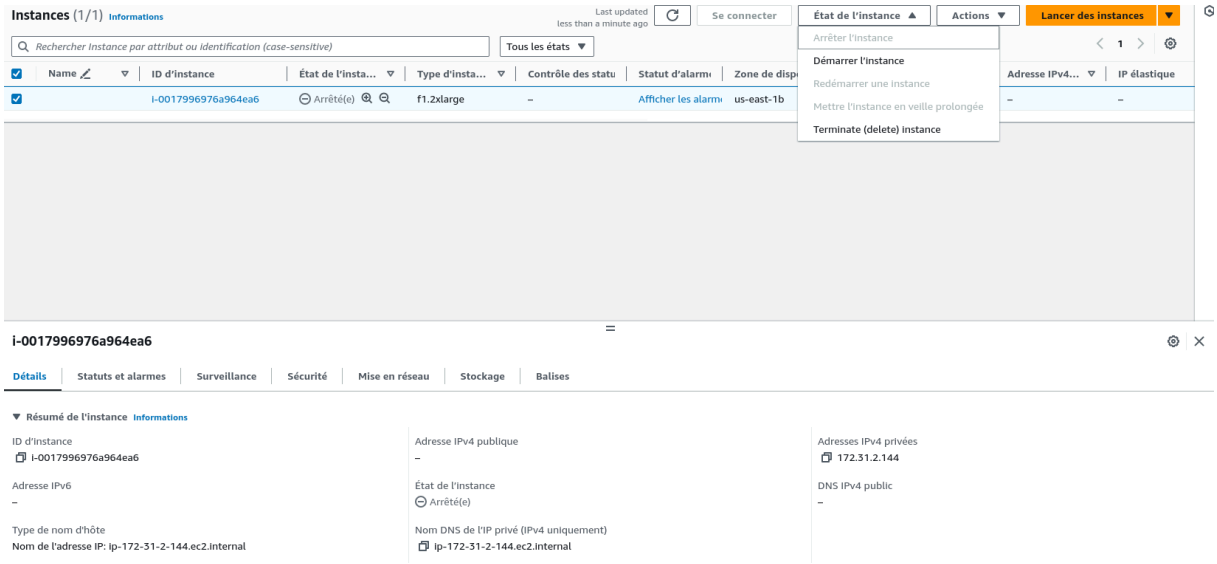
Guide d'utilisation RacEr sur AWS

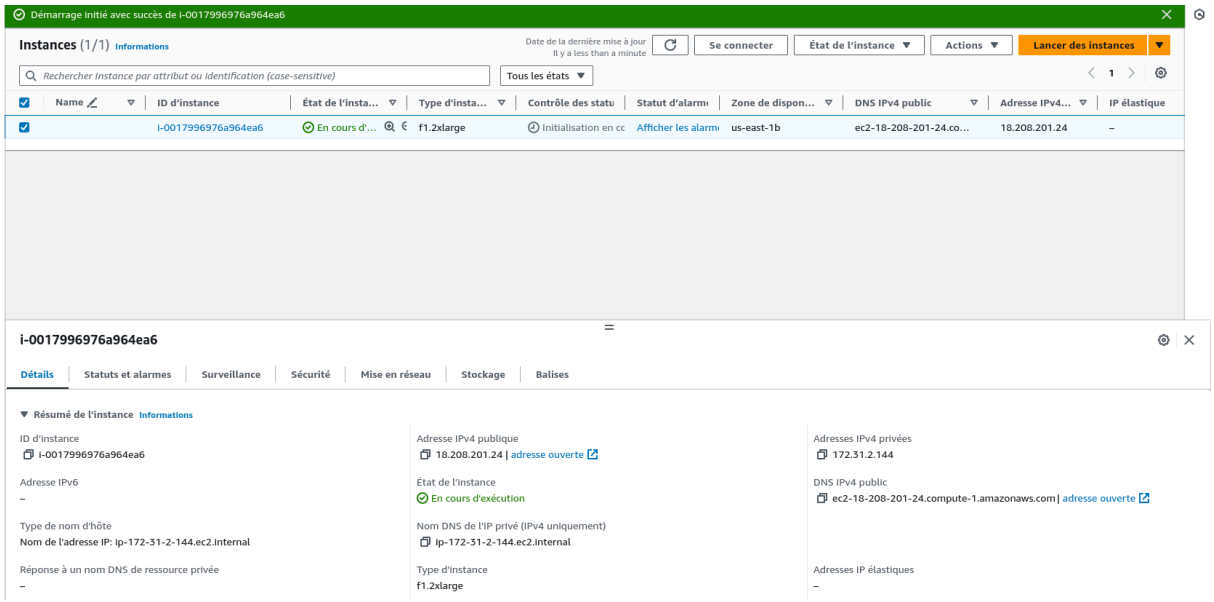
Il faut noter que l'initialisation de RacEr sur FPGA sur AWS a été faite par Vijay. Ainsi la suite de ce guide ne présente que les aspects permettants l'utilisation de RacEr ainsi que la création et l'exécution d'application.

Mise en route du serveur et connexion

- Aller sur le site AWS et sélectionner instances.
- Sélectionner l'instance à démarrer et faire "État de l'instance" puis "Démarrer l'instance"



- Copier l'adresse IP de l'instance visible en dessous (DNS IPv4 public)



- Se connecter à l'aide d'un terminal en utilisant la commande suivante :

`ssh centos@IPV4` et remplacer l'IPV4 par celle obtenue dans AWS, ici .

Compilation et exécution

Avant la compilation, il est nécessaire de charger le composant FPGA implémentant les coeurs Posits. Cela est possible avec la commande : `sudo fpga-load-local-image -S 0 -I agfi-0985378569afcdc65`

Pour faire fonctionner une application il faut faire 2 compilations. La première consiste à compiler la partie qui sera exécutée sur le device (FPGA) avec les coeurs Posit et la seconde est de compiler la partie hôte depuis laquelle le code sera lancé.

- Emplacement du code du device : `/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/spmd/RacEr_cuda_lite_runtime/float_vec_memcpy/`
- Commande de compilation du kernel : `make main.riscv`
- Sortie d'exemple attendue :

```
main.o
RacEr_set_tile_x_y.o
RacEr_printf.o
kernel_float_vector_memcpy.o
/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/riscv-tools/riscv-install/bin/../li
b/gcc/riscv32-unknown-elf-dramfs/9.2.0/../../../../riscv32-unknown-elf-dramfs/lib/libc.a
/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/riscv-tools/riscv-install/bin/../li
b/gcc/riscv32-unknown-elf-dramfs/9.2.0/../../../../riscv32-unknown-elf-dramfs/lib/libm.a
/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/riscv-tools/riscv-install/bin/../li
b/gcc/riscv32-unknown-elf-dramfs/9.2.0/../../../../riscv32-unknown-elf-dramfs/lib/libposit.a
/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/riscv-tools/riscv-install/bin/../li
b/gcc/riscv32-unknown-elf-dramfs/9.2.0/libgcc.a
/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/spmd/common/crt.o
[centos@ip-172-31-2-144 float_vec_memcpy]$
```

- On doit alors obtenir un répertoire avec un executable au format `main.riscv`

```
[centos@ip-172-31-2-144 float_vec_memcpy]$ ls
kernel_float_vector_memcpy.c         main.c         main.riscv             RacEr_printf.o
kernel_float_vector_memcpy.comp.log  main.comp.log  Makefile               RacEr_set_tile_x_y.comp.log
kernel_float_vector_memcpy.o         main.o         RacEr_printf.comp.log  RacEr_set_tile_x_y.o
[centos@ip-172-31-2-144 float_vec_memcpy]$
```

- Emplacement du code de l'hôte : `/home/centos/RacEr_modified/RacEr_bladerunner/RacEr_f1/regression/cuda`
- Le fichier permettant de gérer les exemples de régression à exécuter est `tests.mk`. Ici seulement le test `test_float_vector_memcpy` sera exécuté car il est le seul décommenté.

```
#INDEPENDENT_TESTS += test_float_vec_div
#INDEPENDENT_TESTS += test_float_vec_exp
#INDEPENDENT_TESTS += test_float_vec_sqrt
#INDEPENDENT_TESTS += test_float_vec_log
#INDEPENDENT_TESTS += test_float_vec_dotprod
#INDEPENDENT_TESTS += test_float_matrix_mul
#INDEPENDENT_TESTS += test_float_matrix_mul_shared_mem
#INDEPENDENT_TESTS += test_softmax
#INDEPENDENT_TESTS += test_log_softmax
#INDEPENDENT_TESTS += test_conv1d
#INDEPENDENT_TESTS += test_conv2d
#INDEPENDENT_TESTS += test_float_matrix_inverse
INDEPENDENT_TESTS += test_float_vector_memcpy
#INDEPENDENT_TESTS += test_float_matrix_memset
#INDEPENDENT_TESTS += test_float_matrix_mul2
#INDEPENDENT_TESTS += test_float_matrix_mul3

# REGRESSION_TESTS is a list of all regression tests to run.
```

- Commande de compilation et d'exécution du programme : `make regression`. Pour sélectionner quelle code sera exécuté lors de la régression, il faut commenter ou dé-commenter les applications choisies.

```
[centos@ip-172-31-2-144 cuda]$ make regression
sudo /home/centos/RacEr_modified/RacEr_bladerunner/RacEr_f1/regression/cuda/test_float_vector_memcpy /hom
e/centos/RacEr_modified/RacEr_bladerunner/RacEr_manycore/software/spmd/RacEr_cuda_lite_runtime//float_vec
tor_memcpy/main.riscv test_float_vector_memcpy | tee /home/centos/RacEr_modified/RacEr_bladerunner/RacEr_
f1/regression/cuda/test_float_vector_memcpy.log
RacEr INFO: test float vector memcpy 1
RacEr REGRESSION TEST PASSED

============================================================

Parsing cuda Regression Test results...

============================================================

PASS: Regression Test test_float_vector_memcpy passed!
============================================================

PASS! All 1 tests passed for cuda

============================================================
[centos@ip-172-31-2-144 cuda]$
```

Application custom

Pour faire un portage ou pour le développement d'une application, utiliser la documentation disponible dans `documentation/racer/documentation.pdf`

Pour la compilation d'une application custom, faire une copie des fichiers dans les emplacements comme pour la partie précédante et compiler de la même manière.

Il est conseillé d'utiliser le dossier custom, `custom_kernel` disponible dans le repertoire. Modifier les noms "custom_kernel" aux endroits suivants :

- nom de dossier
- fichier custom_kernel.c
  - nom de la fonction
  - nom du fichier
- Makefile
  - KERNEL_NAME
  - OBJECT_FILES

Pour la partie de l'hôte, faire la même chose en utilisant le dossier `custom_main` Modifier les noms "custom_main" aux endroits suivants :

- nom des fichiers
- nom de la fonction
- modifier le nom de la fonction kernel appelée dans la fonction : `RacEr_mc_kernel_enqueue`

Il faut ensuite mettre le kernel dans le dossier où sont les kernels, de même pour les fichiers de l'hôte et les compiler. voir la partie Compilation