Performance measurement with py-veo

This notebook shows an example of performance measurement of VE kernels with py-veo and py_veosinfo.

In [1]:

```
import veosinfo as vi
from veo import *
import psutil, os, signal, pprint
pp = pprint.PrettyPrinter(indent=4)
```

Now get Aurora node information:

In [2]:

```
print vi.node_info()
```

```
{'status': [0, 0], 'cores': [8, 8], 'nodeid': [0, 1], 'total_node_c
ount': 2}
```

Define build options and VEO kernel source

In [3]:

```
bld = VeBuild()
bld.set_build_dir("_ve_build")
bld.set_c_src("_average", r"""
double average(double *a, int n)
{
    int i;
    double sum = 0;

    for (i = 0; i < n; i++)
        sum += a[i];

    return sum / (double)n;
}
""", flags="-O2 -fpic -pthread -report-all -fdiag-vector=2")
```

In [4]:

```
#veorun_name = bld.build_veorun(flags="-O2 -fpic -pthread", verbose=True)
ve_so_name = bld.build_so(flags="-O2 -fpic -shared", verbose=True)
```

```
/opt/nec/ve/bin/ncc -O2 -fpic -pthread -report-all -fdiag-vector=2
-c _average.c -o _average.o
ncc: vec( 101): _average.c, line 7: Vectorized loop.
ncc: vec( 126): _average.c, line 8: Idiom detected.: Sum

---------
NEC C/C++ Compiler (2.1.27) for Vector Engine     Fri May  3 11:23:
02 2019
FILE NAME: _average.c

FUNCTION NAME: average
DIAGNOSTIC LIST

 LINE              DIAGNOSTIC MESSAGE

     7: vec( 101): Vectorized loop.
     8: vec( 126): Idiom detected.: Sum


NEC C/C++ Compiler (2.1.27) for Vector Engine     Fri May  3 11:23:
02 2019
FILE NAME: _average.c

FUNCTION NAME: average
FORMAT LIST

 LINE   LOOP      STATEMENT

     2:           double average(double *a, int n)
     3:           {
     4:               int i;
     5:               double sum = 0;
     6:
     7: V------>      for (i = 0; i < n; i++)
     8: V------           sum += a[i];
     9:
    10:               return sum / (double)n;
    11:           }


---------
compile _average -> ok
/opt/nec/ve/bin/ncc -O2 -fpic -shared -o _ve_build/_average.so _ve_
build/_average.o
```

Create VEO process, context, load VEO kernel as library

In [5]:

```
otids = set([_thr.id for _thr in psutil.Process().threads()])
```

In [6]:

```
nodeid = 0   # VE node ID
#print vi.cpu_info(0)
vi.metrics.ve_cpu_info_cache[nodeid] = vi.cpu_info(nodeid)

#proc = VeoProc(nodeid, veorun_bin=os.getcwd() + "/" + veorun_name)
proc = VeoProc(nodeid)
```

In [7]:

```
ctxt = proc.open_context()
```

In [8]:

```
lib = proc.load_library(os.getcwd() + "/" + ve_so_name)
#lib = proc.static_library()
lib.average.args_type("double *", "int")
lib.average.ret_type("double")
```

Create a numpy array filled with random numbers

In [9]:

```
n = 128 * 1024      # length of random vector: 128k elements
a = np.random.rand(n)
print("VH numpy average = %r" % np.average(a))
```

```
VH numpy average = 0.5001446878124531
```

submit async VE function request

In [10]:

```
perf_before = vi.ve_pid_perf(nodeid, ctxt.tid)
```

In [11]:

```
req = lib.average(ctxt, OnStack(a), n)

avg = req.wait_result()
print("VE kernel average = %r" % avg)
```

```
VE kernel average = 0.5001446878124531
```

In [12]:

```
perf_after = vi.ve_pid_perf(nodeid, ctxt.tid)
metrics = vi.calc_metrics(nodeid, perf_before, perf_after)
pp.pprint(metrics)
```

```
{   'AVGVL': 256.0,
    'CPUPORTCONF': 0.0,
    'EFFTIME': 1.01785510080272e-07,
    'ELAPSED': 45.48079299926758,
    'L1CACHEMISS': 7.298256441907113,
    'MFLOPS': 28369.171424162938,
    'MOPS': 57524.20922697114,
    'USRSEC': 0.00042564214285714284,
    'USRTIME': 4.629285714285714e-06,
    'VLDLLCHIT': 100.0,
    'VOPRAT': 98.63310000901252,
    'VTIMERATIO': 82.99645116494368}
```

In [ ]:

```
del req
del lib
del proc
```

NOTE Cleanup does not work well. No idea, yet, why. Some processes are left over. You should do: `Kernel -> Restart & Clean up` after the next step (which cleans up the bld files).

Cleanup VEO kernel build files

In [ ]:

```
bld.clear()
bld.realclean()
```

In [ ]:

```
```