What is rabbit?

rabbit is a high-performance computing system at Oregon State University. It includes:

- **Xeon system**
  - rabbit.engr.oregonstate.edu
  - 2 E5-2630 Xeon Processors
  - 16 Cores total
  - 64 GB of memory
  - 2 TB of disk

- **Xeon Phi support**
  - icc, icpc, libraries, drivers

- **PCIe Bus**

- **31S1P Xeon Phi system**
  - 57 Cores
  - 22 nm
  - 8 GB of memory
  - No disk

- **Application support**

What is rabbit?

```
rabbit 151% lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                32
On-line CPU(s) list:   0-31
Thread(s) per core:    2
Core(s) per socket:    8
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Stepping:              2
CPU MHz:               2399.982
BogoMIPS:              4799.30
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              20480K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
```

What is rabbit?

rabbit lives in a rack in our server room in the Kelley Engineering Center:

- **NVIDIA Titan Black**
  - PCIe Bus
  - 15 SMs
  - 2880 CUDA cores
  - 8 GB of memory

- **Intel Xeon Phi**
  - PCIe bus and riser

What is rabbit?

rabbit lives in a rack in our server room in the Kelley Engineering Center:

- **PCIe bus and riser**

- **mgmt port**

- **network**
Getting to Xeon Phi and setting up your account

To verify that the Xeon Phi card is there:

source /nfs/guille/a2/rh80apps/intel/studio.2013-sp1/bin/iccvars.csh intel64
set path=( $path $ICCPATH )

To run some operational tests on the Xeon Phi:

To see the Xeon Phi card characteristics:

Put this in your cshrc:

ssh rabbit.engr.oregonstate.edu –l yourengrusername

Running micinfo

Running miccheck

Running micsmc, I

Running micsmc, II
To compile on rabbit for rabbit:
```bash
icpc -o try try.cpp -lm -openmp -align -qopt-report=3 -qopt-report-phase=vec
```

To cross-compile on rabbit for the Xeon Phi:
```bash
icpc -mmic -o try try.cpp -lm -openmp -align -qopt-report=3 -qopt-report-phase=vec
```

Note: The summary of vectorization success or failure is in a `.optvec` file

To execute on the Xeon Phi, type this on rabbit:
```bash
micnativeloadex try
```

Cross-compiling and running from rabbit

To cross-compile on rabbit for the Xeon Phi, deliberately disabling vectorization:
```bash
icpc -mmic -o try try.cpp -lm -openmp -no-ve -align -qopt-report=3 -qopt-report-phase=vec
```

Gaining Access to the Cores, I

```c
#pragma omp parallel for
for (int i = 0; i < N; i++)
C[i] = A[i] * B[i];
```

```c
float sum = 0.;
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < N; i++)
sum += A[i] * B[i];
```

Gaining Access to the Cores, II

```c
#pragma omp parallel sections
#pragma omp section
... 
#pragma omp section
... 
#pragma omp task
... 
```

Gaining Access to the Vector Units

```c
C[0:N] = A[0:N] * B[0:N];
```

```c
#pragma omp simd
for (int i = 0; i < N; i++)
C[i] = A[i] * B[i];
```

```c
#pragma omp parallel for simd
for (int i = 0; i < N; i++)
C[i] = A[i] * B[i];
```

Turning Off All Vectorization

```bash
icpc -mmic -o try try.cpp -O3 -lm -openmp -align -qopt-report=3 -qopt-report-phase=vec
```

Compiling for OpenCL

```bash
printinfo: printinfo.cpp
icpc -o printinfo printinfo.cpp /usr/lib64/libOpenCL.so -lm -openmp
```

The only reason I can think of to do this is when running benchmarks to compare vector vs. scalar array processing.

The Intel compiler does a great job of automatically vectorizing when it can.

Warning: just because you didn’t deliberately vectorize your code doesn’t mean it didn’t end up vectorized! Use the “-no-ve” flag instead.
The `printinfo` Program Output

<table>
<thead>
<tr>
<th>Number of Platforms</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform #0:</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>NVIDIA-CUDA</td>
</tr>
<tr>
<td>Vendor</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>Version</td>
<td>OpenCL 1.1, CUDA 7.0.18</td>
</tr>
<tr>
<td>Profile</td>
<td>FULL_PROFILE</td>
</tr>
<tr>
<td>Device #0:</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>CL_DEVICE_TYPE_GPU</td>
</tr>
<tr>
<td>Device Vendor ID</td>
<td>0x10de (NVIDIA)</td>
</tr>
<tr>
<td>Device Maximum Compute Units</td>
<td>15</td>
</tr>
<tr>
<td>Device Maximum Work Item Dimensions</td>
<td>3</td>
</tr>
<tr>
<td>Device Maximum Work Item Sizes</td>
<td>1024 x 1024 x 64</td>
</tr>
<tr>
<td>Device Maximum Work Group Size</td>
<td>1024</td>
</tr>
<tr>
<td>Device Maximum Clock Frequency</td>
<td>1071 MHz</td>
</tr>
</tbody>
</table>

Device Extensions:
- cl_khr_byte_addressable_store
- cl_khr_fp64
- cl_khr_local_int32_base_atomics
- cl_khr_local_int32_extended_atomics
- cl_khr_global_int32_base_atomics
- cl_khr_global_int32_extended_atomics
- cl_khr_num_float8
- cl_khr_pragma_unroll
- cl_khr_copy_opts
- cl_khr_gl_sharing
- cl_khr_icd
- cl_nv_compiler_options
- cl_nv_device_attribute_query
- cl_nv_device_attribute_query
- cl_nv_global_int32
- cl_nv_local_int32
- cl_nv_copymodes

15 * 192 = 2880 CUDA cores!