Nutzeranleitung Fortgeschrittene

Kompilieren

> **Problem:** Das Kompilieren auf dem Phoenix-Cluster klappt nicht, z.B. wenn gegen eine Bibliothek im Modul-System gelinkt wird (OpenCV).

Dies kann daran liegen, dass viele Bibliotheken (Libraries) mit dem optimierten Intel Compiler kompiliert wurden. Denn viele der installierten Bibliotheken sind mit dem neuesten Intel Compiler (icc) mit AVX 2 Optimierungen kompiliert.

In diesem Fall kann es daher helfen, den **Intel Compiler** (icc) zu verwenden.

Das kann im Idealfall hohe Leistungszuwächse - 50% und mehr - z.B. auf neuen Xeon CPUs bringen, und meistens läuft der Code, der mit dem icc kompiliert wird, schneller. Gerade die *auto vectorization* funktioniert gut, da der icc von sich aus Schleifen umwandelt, um die CPU Eigenschaften besser nutzen zu können.

Das kann aber dazu führen, dass der **GNU Compiler** (gcc) Probleme mit Bibliotheken hat, die für den Intel Compiler stark optimierten Code nutzen. Deshalb muss der Kommandozeile/Shell mitgeteilt werden, dass diese den icc (bzw. icpc für C++) statt den gcc nutzen soll, falls dieses Problem auftritt.

Das wird mit folgenden Befehlen getan:

```
module load intel-studio-2019
source /cluster/share/intelenv.sh
```

Anschließend muss, wie üblich, der Befehl

```
cmake
```

ausgeführt werden.

>

Mehr Informationen, insbesondere auch für in Fortran geschrieben Code, kann unter https://software.intel.com/en-us/articles/performance-tools-for-software-developers-building-hdf5-with-intel-compilers gefunden werden.

Wichtiger Hinweis: Beim Fortan Compiler von Intel (ifort) darf unter Standard-Einstellungen Probleme eine Zeile maximal 132 Zeichen umfassen!

GLIBC

Um neuere Versionen von glibc zu verwenden, ist es leider nicht möglich, einfach das Modul zu laden. Stattdessen werden dem Compiler Flags übergeben, mit denen die Pfade für das Finden der Bibliotheken sowie des Loaders der dynamischen Bibliotheken gesetzt werden. Hier ein kurzes Beispiel:

```
g++ -o My_Program -Wl,--rpath=/cluster/lib/glibc/2.29/lib -Wl,--dynamic-linker=/cluster/lib/glibc/2.29/lib/ld-linux-x8
```

Gaussview

Um Gaussview zu nutzen, muss wie folgt vorgegangen werden. Via ssh -X für eine grafische Oberfläche auf Phoenix einloggen. Dann einen Visualisierungsknoten allokieren. Dann den Namen des allokierten Knotens in Erfahrung bringen und sich auf diesem Knoten einloggen. Dort dann gaussview mit gv starten.

```
ssh -X user@phoenix.hlr.rz.tu-bs.de
sbatch gaussview.job
   Submitted batch job 100990
squeue | grep 100990
   100990       vis gaussvie raskrato  R       0:14      1 vis01
ssh -X vis01
gv
```

Die Jobfile zum Allokieren eines Visualisierungsknoten.

gaussview.job

```
#!/bin/bash -l

#SBATCH --partition=vis
#SBATCH --nodes=1
#SBATCH --time=1:00:00
#SBATCH --job-name=gaussview
#SBATCH --ntasks-per-node=20
```

```
module load software/molecules/gaussian
sleep 1h
```

Python Module

Auf dem Cluster sind eine Reihe von Python Module bereits vorinstalliert. Diese können eingesehen werden, wenn Sie mit *python2* oder *python3* deren „Interpreter“ beitreten und dort die folgende Anweisung *help(„modules“)* ausführen.

Sollte ein Modul benötigt werden, welches dort nicht aufgeführt ist, gibt es zwei Möglichkeiten. Sollten Sie nachweisen können, dass eine große Nutzergruppe Bedarf an diesem bestimmten Modul hat, so wenden Sie sich an das HLR-Service-Team. Da es allerdings zu aufwendig wäre und den Betrieb stören würde, das gesamte Cluster ständig mit neuen Python-Packeten zu befüttern, müssen Sie ansonsten auf virtuelle Umgebungen zurückgreifen.

Für Python2 gestaltet sich dies relativ einfach:

```
virtualenv venv //Erstellt im aktuellen Arbeitsverzeichnis einen Ordner mit dem Namen "venv", in dem sich eine le
source venv/bin/activate //Damit wird die Python-Umgebung geladen, dies kann auch in einem Job-File gemacht w
deactivate //Damit verlässt man die aktuelle Pyton-Umgebung.
```

Solange Sie sich in der Python Umgebung befinden, wird dies durch ein Voranstellen Ihres Names in der Kommandozeile verdeutlicht. Das sieht dann z.B. so aus:

```
(venv)[testuser@login01 ~]
```

In der virtuellen Umgebung können Sie sich nun nach Herzenslust mittels **pip** jedes Modul beschaffen. Sollten Sie keine Verwendung mehr für eine Umgebung haben, lässt sie sich einfach durch das Löschen des Ordners entfernen.

Für Python3 muss aktuell ein kleiner Umweg gemacht werden, bevor virtuelle Umgebungen anlegt werden können:

```
module load python/3.7
python3 -m pip install --user virtualenv //Hiermit wird virtualenv für sie als Nutzer installiert ohne Konflikte zu Pyth
python3 -m virtualenv venv //Damit stellen sie sicher, dass Python3 in der virtuellen Umgebung eingerichtet wird.
```

Wir hoffen mit dem nächsten Update von Python diesen Schritt obsolet zu machen.

Sie können natürlich fragen, wieso Sie nicht immer *pip install –user* verwenden, um sich virtuelle Umgebungen zu sparen. Und in der Regel haben Sie Recht, aber es gibt immer wieder Konflikte zwischen verschiedenen Modulen und so garantieren Sie, dass Sie für jedes Projekt eine konfliktfreie Umgebung schaffen können.

> Zum Schluss noch eine Anmerkung zu Python-Bibliotheken, die mit C-Code daherkommen: Der Intel-Kompiler wird von pip nicht akzeptiert. Stattdessen laden Sie daher bitte den gcc-

> Compiler über das entsprechende Modulefile „comp/gcc/8.3.0“ oder welche Version Sie auch immer verwenden möchten.

Revision #5
Created 12 March 2024 14:00:42 by Michael Giemsa
Updated 22 May 2024 16:33:46 by Michael Giemsa