Abel Tips

or: "why the hell doesn't my code run?" – and other stories ...

Armadillo & Lapack

- Abel har ikke Armadillo installert ... ikke overraskende ...
- Har heller ikke Lapack og Blas heller, men har Intels MKL bibliotek.
- Armadillo kan installeres på hjemmeområdet, men det krever visse modifikasjoner til Armadillos compile-skript:
  - Fil: `build_aux/cmake/Modules/ARMA_FindMKL.cmake`
  - På linje 21 er det en liste med paths til steder hvor CMake leter
  - Legg til den som gjelder for Abel
    - Kjør: **`echo $MKLPATH`** for å se denne
    - Husk å kjøre **`module load intel`** først
  - Kjør **`module load cmake`** og deretter **`cmake .`**
  - Kjør **`make install DESTDIR=~`** for å installere armadillo på hjemmeområdet. Den oppretter katalogen '`~/usr`' automatisk.

Init Script

- Det kan lønne seg å ha et init script for å laste de mdouler og sette de variabler du trenger når du logger inn.

  Mitt er som følger:

```
#!/bin/bash
echo "Loading Intel module"
module load intel
module load openmpi.intel

echo "Setting LD_LIBRARY_PATH"
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/usr/lib64
echo $LD_LIBRARY_PATH

echo "Setting CPLUS_INCLUDE_PATH"
export CPLUS_INCLUDE_PATH=$HOME/usr/include
echo $CPLUS_INCLUDE_PATH
```

Makefile

- For å få på plass alle compiler settingene for å kjøre MKL, er det nyttig å bruke Intels MKL Link Line Advisor: http://software.intel.com/sites/products/mkl/MKL_Link_Line_Advisor.html
- Har lagt til en egen seksjon i makefilen for Abel:

```
ifeq ($(NODE),Abel)
    LIBINTEL = -L$(MKLROOT)/lib/intel64 -lmkl_intel_lp64
        -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm
    LIBARMA  = -L$(HOME)/usr/lib64 -larmadillo
    LIBFLAGS = $(LIBINTEL) $(LIBARMA) -lmpi $(LIBADD)
    CFLAGS   = $(DEBUG) -O2 -c -openmp -DMKL_LP64
        -I$(MKLROOT)/include -I$(HOME)/usr/include
    LFLAGS   = $(DEBUG) -openmp
else
    LIBFLAGS = -llapack -lblas -larmadillo $(LIBADD)
    CFLAGS   = $(DEBUG) -O2 -c -fopenmp
    LFLAGS   = $(DEBUG) -fopenmp
endif
```

SBatch & Slurm

- Abel bruker sbatch og slurm for job scheduling.
- Scriptene er relativ greie. Det er bare å følge guiden på hjelpesidene til USIT: http://www.uio.no/english/services/it/research/hpc/abel/
- Et par problemer så langt:
  - OpenMPI jobber som starter på Abels rack 10 eller større (ish) ser ut til å stoppe etter få sekunder. Feilmeldingene sier lite, men det er mulig disse nodene ikke finner Armadillo biblioteket.
  - Løsning: Velg hvilke rack man vil kjøre på i job-filen:

```
#SBATCH --constraint="rack2|rack3|rack4|rack5|
rack6|rack7|rack8|rack9"
```

Nyttige SBatch kommandoer

- Legge en jobb i køen:
  `sbatch JobScript.sh`
- Se status:
  `squeue -u [username]`
- Stoppe en jobb:
  `scancel [jubnummer]`
- Det er smart at jobbene cacher til fil slik at om de går over wall time eller stopper av andre grunner, kan de startes igjen der de slapp. Greit å bruke save() og load() funksjonene i Armadillo for eksempel. I mitt script har jeg:

```
## Run command
cp $SUBMITDIR/*.arma $SCRATCH
mpirun CISD-Abel > $OUTFILE
cp $SCRATCH/*.arma $SUBMITDIR
```

Eksempler på kjøringer

Job	Computer	Dim(Basis)	It.	Cores	Run-time	CPU Hours
P7 Sh6 M0 2s1 ω0.01	TheBeast	4.88×10^5	79	6	05:22:25	32
P7 Sh6 M0 2s1 ω0.1	Gizmo	4.88×10^5	44	8	05:44:00	46
P11 Sh5 M2 2s1 ω0.1	Abel	9.45×10^5	56	160	00:11:22	30
P13 Sh5 M2 2s1 ω0.1	Abel	1.98×10^6	89	160	00:54:38	147
P6 Sh8 M0 2s0 ω0.1	Gizmo	2.46×10^6	49	8	70:21:36	563
P6 Sh8 M0 2s0 ω0.28	Gizmo	2.46×10^6	34	8	41:18:10	330
P6 Sh8 M0 2s0 ω1.0	Abel	2.46×10^6	29	320	01:10:43	377
P6 Sh9 M0 2s0 ω1.0	Abel	8.62×10^6	31	400	06:06:58	2446
P6 Sh10 M0 2s0 ω0.1	Abel	2.65×10^7	53	640	32:08:56	20575
P6 Sh10 M0 2s0 ω1.0	Abel	2.65×10^7	34	640	21:11:13	13560
P11 Sh6 M0 2s1 ω0.1	Abel	5.76×10^7	79	800	14:09:24	11325
P12 Sh6 M0 2s0 ω0.1	Abel	1.49×10^8	73	1280	27:42:36	35469
P12 Sh6 M0 2s0 ω1.0	Abel	1.49×10^8	37	960	17:59:35	17273
P13 Sh6 M0 2s1 ω0.1	Abel	3.13×10^8				

Table 8.11: Run-times for some of the major jobs. The job column describes jobs by number of particles (P), number of shells (Sh), total M (M), total spin times two (2s) and omega (ω). The column labelled "It." lists the number of Lanczos iterations needed for convergence as this number highly affects computation time. All runs have $< 1 \times 10^{-6}$ as convergence criteria.