Login from PC: 1st exercise (on Altix UV)

Login at PC

- Power on your PC (if off)
- PC will **automatically** log you into local course account
  - this account is different from the account used on parallel machine!
- Auf Desktop
  in “Ordner für Datentransfers”:
  - Click on **Xming** to start X Windows Server (necessary for opening console or editor windows from Altix server)
  - Click on **Putty** to start ssh session
  - Click on **Matlab** to start Matlab session

Login on Altix compute server:

- Enter host name
  LXLogin2*.LRZ.de* (or LXLogin[1|3|4].LRZ.de)
  into Putty host field and click *Open*.
- Accept & save host key [only first time]
- Enter user name (***a2c06aa***) and password (*see white board*) into opened console window
- Start an **xterm**
- On this xterm, go to the Altix UV with
  ssh -Y uv2
- Open additional console by entering
  xterm &
  in original console

Your working directory:

cd <*wdir*>/*number of PC*
***with** leading zero*

Settings for <*wdir*> are given on the next slides

Add in .bshrc before course: source /etc/profile.d/modules.sh

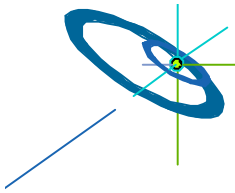
MPI on SGI Altix UltraViolet at LRZ, Garching/Munich

- Compute server: uv2 (accessed via **ssh –Y uv2** from LXLogin2.lrz.de)
- Shell: bash (is already the login-shell)
- Working directory: **cd ~/MPI/*#nr***
- Editing:
  - vi, vim, emacs, or joe
- Compilation:
  - **mpif90** –o *my_prog* *my_prog**.f90*** or *my_prog**.f***
  - **mpicc** –o *my_prog* *my_prog**.c***
- Execution:
  - ulimit –s 40000 (if you want to use, e.g., "*float[10000000] x*" in your prog.c)
  - **mpirun** –np *number_of_cpus* **./***my_prog* (Maximum: **4** processes!)
- Local Description at LRZ:
  - **http://www.lrz.de/services/compute/linux-cluster/** and then click on the MPI link
  - **man mpi**
  - there is also a man-page for each MPI routine

module load emacs

For MPI-3.0 (incl. MPI shared memory) and new mpi_f08:
module unload mpi.mpt intel/15.0
module load intel/14.0 mpi.ompi/1.8

Interactively only during the course!



Login from PC: 2nd exercise (SuperMIC)

Login on SuperMIC frontend:

- Enter host name
  training.srv.mwn.de
  into Putty host field and click *Open*.
- Accept & save host key [only first time]
- Enter user name (***a2c06aa***) and password (*see white board*) into opened console window
- Start an **xterm**
- On this xterm, go to the SuperMIC with
  ssh -Y supermic.smuc.lrz.de
- Open additional console by entering
  xterm &
  in original console

Your working directory:

cd *<wdir>/number of PC*
***with** leading zero*

SuperMIC requires a batch job:

- Generate SSH key pair without passphrase:
  ssh-keygen
- Submit 1-node sleep job with:
  cp ~/MPIOMP/course/job.ll ./
  llsubmit job.ll
  (5 hours; for more,adapt script)
- Monitor job with:
  llq –u $USER
  and find out the node name
- Log into node:
  ssh <node-name-without-ib>
- Compiling works only on the frontend; compile your code on the frontend and run it on the node
- **module load likwid/4.0**
  for likwid

Add in .bshrc before course: source /etc/profile.d/modules.sh

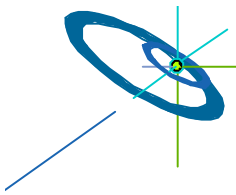
Pure MPI pinning (Intel MPI) – selection

- **`I_MPI_PIN={0,1}`**
  Switch off/on MPI affinity (default = on)

- **`I_MPI_PIN_PROCESSOR_LIST=<proclist>`**
  Set core IDs to run on.

  Example:
  `I_MPI_PIN_PROCESSOR_LIST=0-7` # 1st socket on SuperMIC

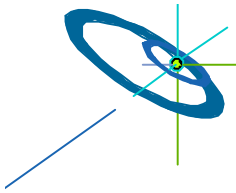
- **`I_MPI_DEBUG=4`**
  Print (among other things) process-to-core mapping



Intel MPI+OpenMP hybrid pinning – selection

Define a “domain” for the multi-threaded MPI processes:

- **`I_MPI_DOMAIN=...`**
  - **`core | socket | node | cache`**
    MPI process spans the specified entity (cache=largest cache)
  - **`omp[:[scatter | compact ]]`**
    MPI process spans as many logical cores as **`OMP_NUM_THREADS`**, with scattered or compact distribution
  - **`<n>[:[scatter | compact ]]`**
    ditto, but the number of logical cores is n
  - **`[m1,...,mn]`** (brackets included)
    specify for each MPI process a bit mask (in hexadecimal), numbering according to BIOS. Example:
    `[0x000F,0x00F0,0x0F00,0xF000]`
    → this provides full control!



Intel `KMP_AFFINITY` environment variable: Pinning OpenMP

- **`KMP_AFFINITY`**=[<*modifier*>,...]<*type*>[,<*permute*>][,<*offset*>]
  - **modifier**
    - **granularity=<*specifier*> takes the following specifiers: fine, thread, and core**
    - **norespect**
    - **noverbose**
    - **proclist={<*proc-list*>}**
    - **respect**
    - **verbose**
    - **Default:** noverbose,respect,granularity=core
  - **type (required)**
    - **compact**
    - **disabled**
    - **explicit (`GOMP_CPU_AFFINITY`)**
    - **none**
    - **scatter**

OS processor IDs

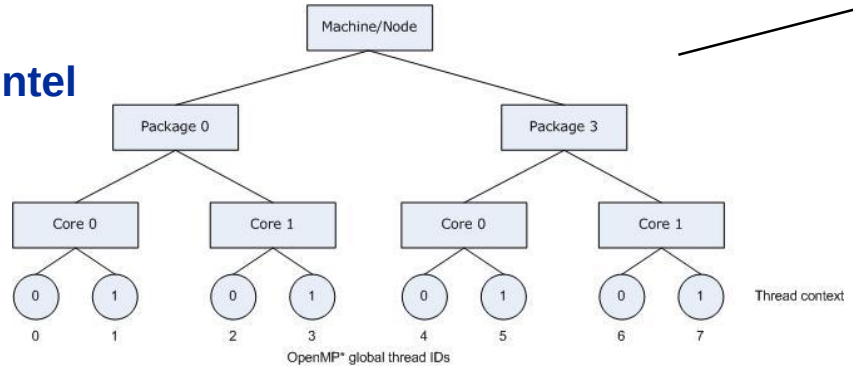
Respect an OS affinity mask in place

- **`KMP_AFFINITY=verbose,none`** to list machine topology map

Intel `KMP_AFFINITY` examples

- KMP_AFFINITY=granularity=fine,compact

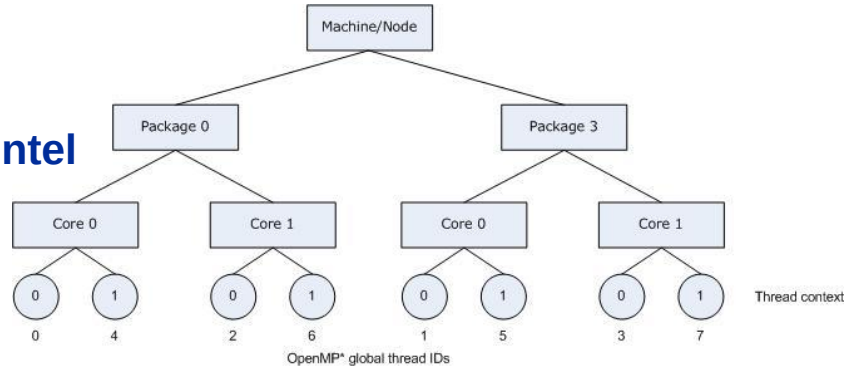
(c) Intel



Package means chip/socket

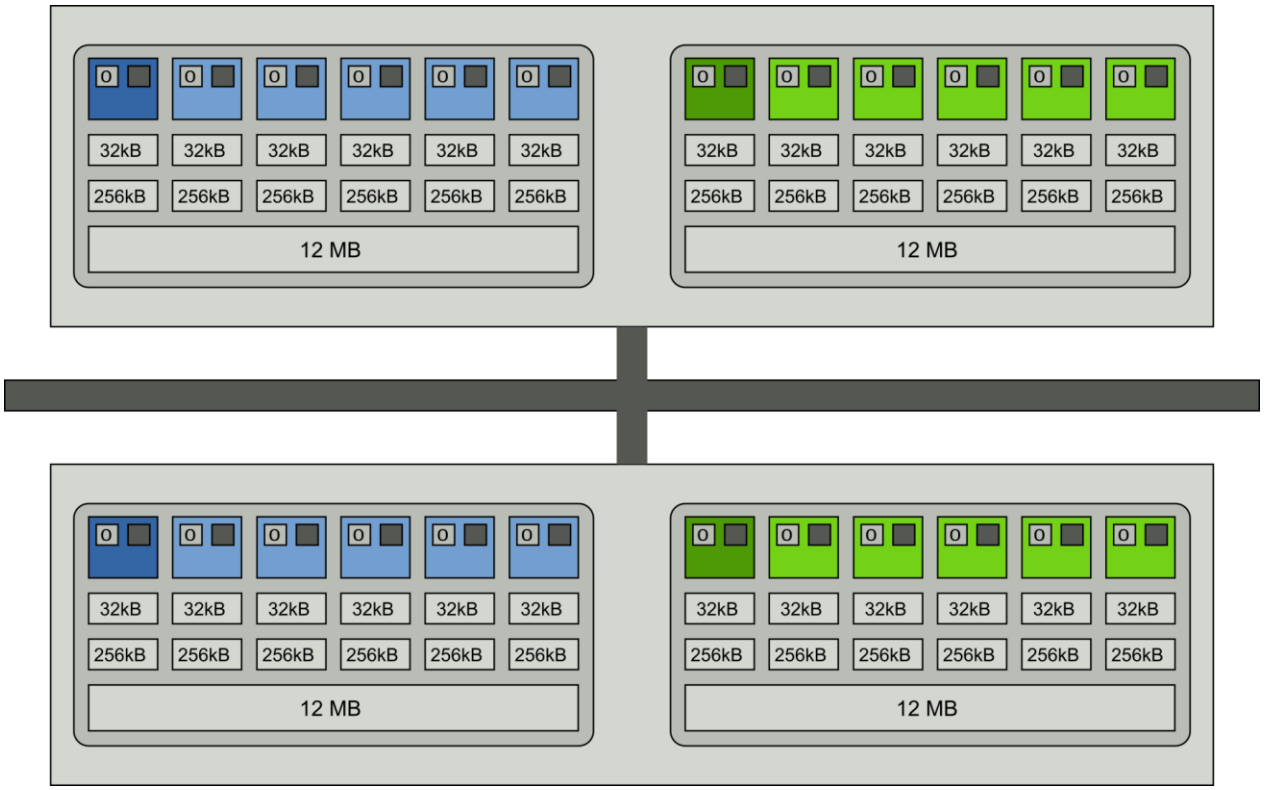
- KMP_AFFINITY=granularity=fine,scatter

(c) Intel



Example:

1 MPI process per socket



Intel MPI+compiler:

```
OMP_NUM_THREADS=6 mpirun -ppn 2 -np 4 \
     -env I_MPI_PIN_DOMAIN socket -env KMP_AFFINITY scatter ./a.out
```