VRE for regional Interdisciplinary communities in Southeast Europe and the Eastern Mediterranean

HPC in Albania – Infrastructure and Applications

Name: Neki Frasheri
Position: Professor
Organization: Polytechnic University of Tirana

The VI-SEEM project initiative is co-funded by the European Commission under the H2020 Research Infrastructures contract no. **675121**

Some History – HPC in Albania

- Parallel processing in Albania
  - Participation in European Programmes FP6 and FP7
    - Projects SEE-GRID1/2 (in parallel with regional network SEEREN)
    - First grid clusters in INIMA and UPT, integrated in regional “super-grid” funded by SEE-GRID and Ministry of Education and Science
    - Project SEE-GRID-SCI in UPT, application CHERS for processing of satellite imagery in space-time domain for environmental studies
    - Project HP-SEE for regional high performance processing, application GIM for geophysical (gravity) modeling and inversion
    - Initiative of Ministry of Innovation and P.R. of China Government for a small parallel system in UPT
  - Participation in European Programme H2020
    - Project VI-SEEM for the regional virtual platform
    - Application for wind simulation over rugged terrain
    - Integration of local HPC and cloud systems in the regional platform

HPC infrastructure in FTI.UPT

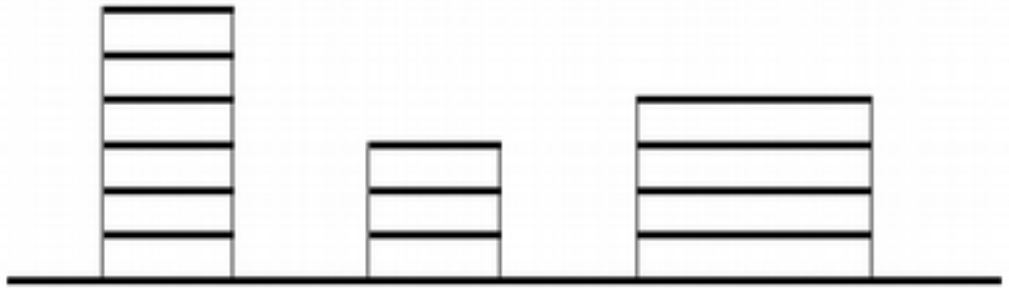
- SUGON/DAWNING blades system
  - 24 nodes with 2x4 cores
  - Two gateway servers
  - 4 GB central memory per node
  - Actually 144 nodes active
  - 1 GBPS switch, NFS available
  - Scientific Linux 6
  - Middleware MPI & Torque
- Storage unit with 16 HDDs, 16 TB capacity
  - To be installed: Fiber Interface
- Difficulties
  - Electricity, maintenance

HPC Access

- SUGON gateway servers
  - Server **mpi1.upt.al** <193.254.1.11> active
  - Server **mpi2.upt.al** <193.254.1.12> backup
- Access protocol: **ssh**
  - Linux command line knowledge necessary
  - Help available from Faculty of Information Technology
  - Prearrangement with Department of Fundamentals of Informatics (DBI) necessary for preparation of developing and running environments
  - Contact <nfrasheri@fti.edu.al>
- Directories **/home** and **/opt** of gateway server are shared with and accessible from all working nodes via NFS

Parallel Programming – OpenMP

- OpenMP is integrated in the **gcc** compiler
  - #include </usr/lib/gcc/x86_64-redhat-linux/4.4.4/include/**omp.h**>
  - **gcc -fopenmp** …
  - Programming with threads in parallel using #pragma directive, example:
    - *#pragma omp parallel for num_threads(NT) private(S)*
      for (i=0; i<n; i++)
      { S=a[i]+b[i]; c[i]=S; }
  - Run the program within a single node with maximum of 8 therads
- Consult OpenMP documentation for details
- Sketch example of parallelization with threads:



Some hints to run the software

- Obtaining runtime
  - Run program via /usr/bin/**time**
    - /usr/bin/time program …
    - Statistics: summary of runtime for all threads, system time, elapsed time, and summary of CPU% for all cores
    -
- Running offline
  - Run program via /usr/bin/**nohup**
    - Logout and login without interrupting execution
    - Standard output redirected to nohup.out
- Example
  - */usr/bin/nohup "/usr/bin/time ./program … & "*

Parallel Programming – MPI

- Two versions of Message Passing Interface – MPI
  - mpich-3.1
  - openmpi-1.10
- Programming
  - /usr/include/mpich-x86_64/**mpi.h**
  - /usr/include/openmpi-x86_64/**mpi.h**
- Compilation
  - /usr/lib64/mpich/bin/mpicc
  - /usr/lib64/openmpi/bin/mpicc
- Update $PATH and LD_LIBRARY_PATH if necessary
- Consult MPI documentation for details

Remember procedures of MPI (1)

- Parallelization procedures
  - MPI_Init (&argc,&argv)
  - MPI_Finalize ()
- Process identification
  - MPI_Comm_size (*comm*,&size)
  - MPI_Comm_rank (*comm*,&rank)
- Broadcasating
  - MPI_Bcast (&buffer,count,datatype,root,*comm*)
- Synchronization
  - MPI_Barrier (*comm*)
- Communicator
  - *comm* = MPI_COMM_WORLD

Remember procedures of MPI (2)

- Distribution of data between processes
  - MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root,comm)
- Collection of data from processes
  - MPI_Gather (&sendbuf,sendcnt,sendtype,&recvbuf,recvcount,recvtype,root,comm)
  - MPI_Allgather (&sendbuf,sendcount,sendtype,&recvbuf,recvcount,recvtype,comm)
  - MPI_Reduce (&sendbuf,&recvbuf,count,datatype,**op**,root,comm)
    where reduction **OP**erators: MPI_MIN, MPI_MAX, MPI_SUM, MPI_PROD, etc.

Running MPI jobs with HYDRA

- Preparatory work: list of nodes names in home directory, example:
  - ~/list-of-nodes
    - node01
    - node02
    - …
- Run program using **mpiexec.hydra** with parameters “-f” and “-np”:
  - *mpiexec.hydra **-f ~/list-of-nodes** **-np NP** **-ppn PP** ./program … &*
  - where ***NP*** – number of processes, ***PP*** – processes per node,
  - and “&” to run in background .
  - Full path /usr/lib64/mpich/bin/mpiexec.hydra
  - Use **nohup** for offline execution

Running MPI jobs with PBS/Torque

- Preparatory work: shell script to run the program:
  - ~/script.sh
    - #!/bin/bash
    - #PBS -l nodes=**NN**:ppn=**8**
    - #PBS -q **batch**
    - #PBS -l walltime=**hh**:00:00
    - /PATH/mpiexec -n NP program …
  - Where: ***NN*** ~ number of requested nodes, "***batch***" ~ the actual active queue, "***hh***" ~ maximal hours requested, "***NP***" ~ number of processes with **8** per node
  - Full "*PATH*":
    - /usr/lib64/mpich/bin/mpiexec
    - /usr/lib64/openmpi/bin/mpiexec
- Run "*hostname*" to check how processes are distributed in nodes

Submitting MPI jobs with PBS

- Submit a job
  - **qsub ./script.sh**
  - JobID number is displayed
  - Output in script.sh.**e**JobID and script.sh.**o**JobID when job terminates
- Check statuis of a job:
  - **qstat [JobID]**
- Delete a job:
  - **qdel JobID**
- List available nodes
  - **Pbsnodes**
- Use **qsub -k eo ...** to get output while running

Hints for good practices

- Evaluate the walltime using small models, and extrapolate this evaluation for real models for better planning of job executions
- Use the "**top**" command to evaluate the required virtual memory, and extrapolate for real models comparing with the physical memory
  - Excess of virtual memory may lead to greater walltime due to swapping
- Evaluate disk space requirements for small models, and extrapolate for real models (the disk storage array is not yet available)
- Do not run jobs if much resources are required, ask for help.

Thank You

- Questions ?