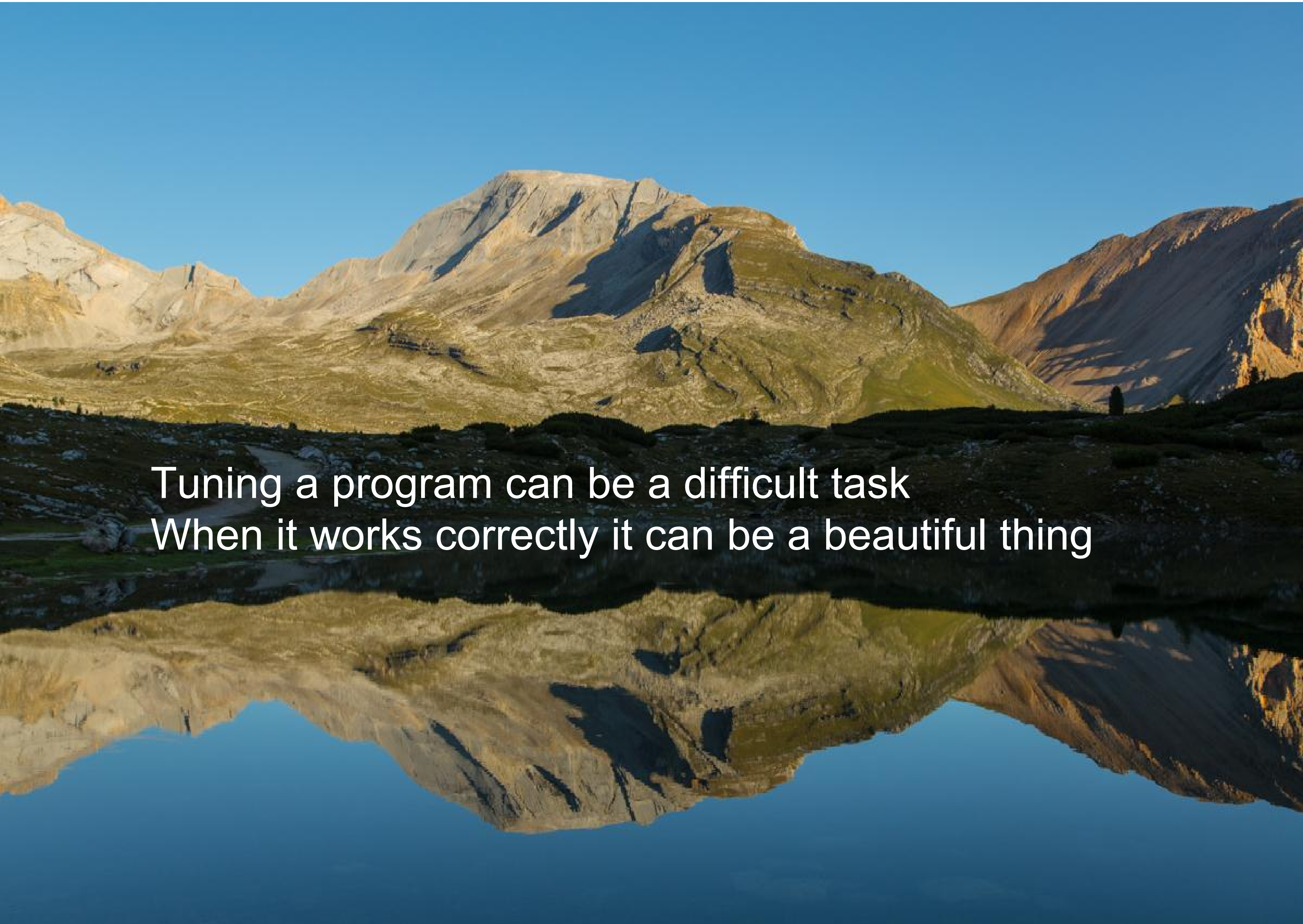
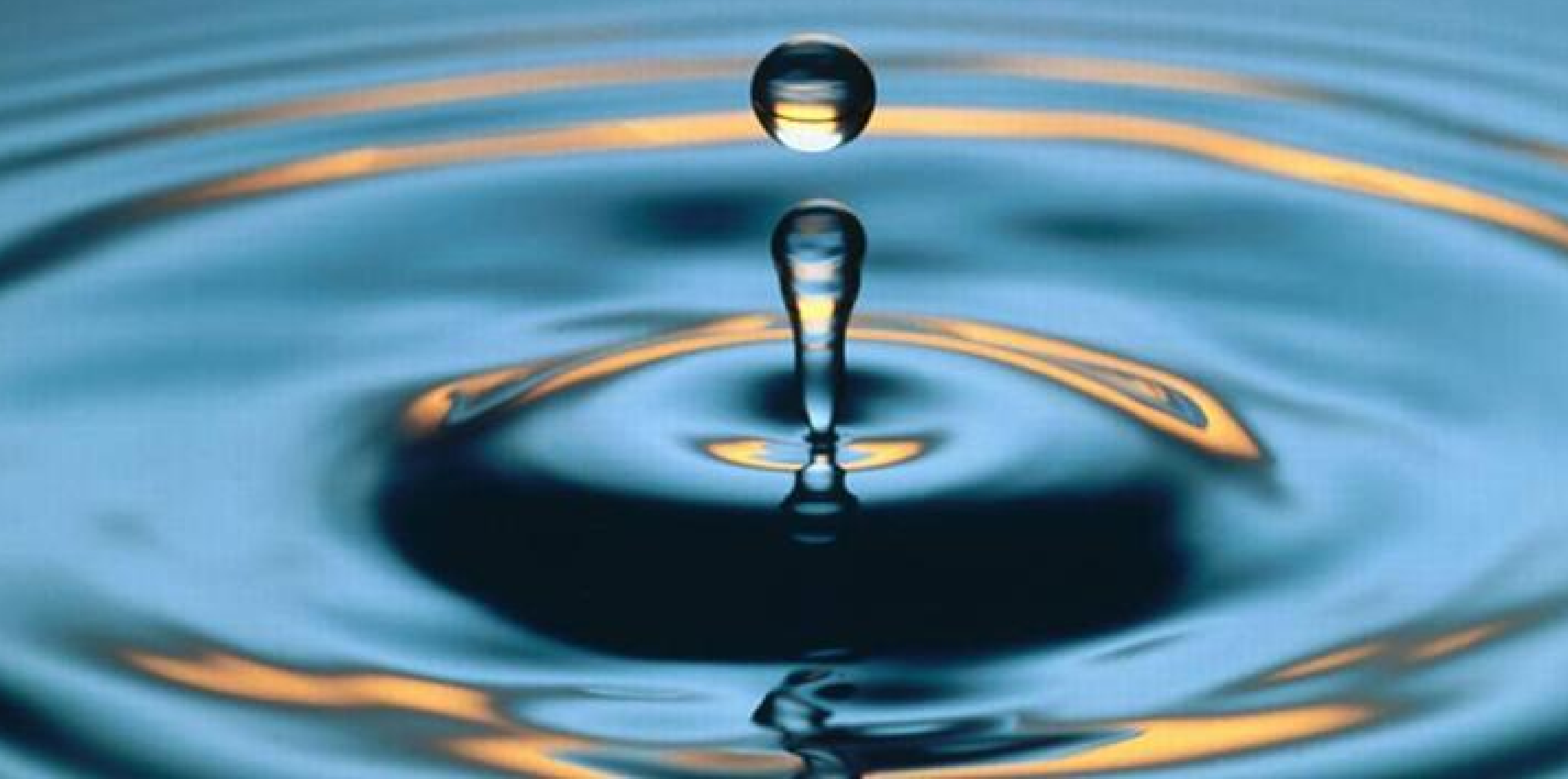
Slurm Processes Isolation

Slurm 2014 User Group

Bill Brophy, Bull
Martin Perry, Bull
Moe Jette, SchedMD
Yiannis Georgiou, Bull
Matthieu Hautreux, CEA


Tuning a program can be a difficult task
When it works correctly it can be a beautiful thing

Even minor disturbances (Slurm processes) can cause a ripple
Resource Specialization of Slurm processes attempts to address this issue



Motivations

- Studies have demonstrated that Operating system (OS) noise can have a major **negative** impact on the performance of parallel jobs [1,2,3]
  - Interference on individual cores --> Desynchronization in collective communication tasks --> Degraded application performance
  - In Many Core Architectures the problem may be more visible
- Sources of interference preventive productive work on compute nodes
  - OS Services
  - Network Interfaces
  - Kernel daemons

[1] Sarp Oral et al. Reducing Application Runtime Variability on Jaguar XT5 in Cray User Group 2010
[2] Hakkan Akkan et al. Understanding and isolating the noise in the Linux kernel. IJHPCA 27(2): 136-146 (2013)
[3] Zero Overhead Linux, Tilera White Paper 2011

Background

- Isolation of system processes on specific cores in each compute node and preventing applications from using those cores in some cases made a significant improvement in job performance [3]

- Slurm introduced support for Core Specialization **at the job level** on CRAY systems (Slurm 14.03.0pre6)
  - `CoreSpecPlugin=core_spec/cray`
  - `–Core-Spec= <count>` option supported in `salloc`, `srun` & `sbatch`

[3] Sarp Oral et al. Reducing Application Runtime Variability on Jaguar XT5 in Cray User Group 2010

The Development Project

- Bull proposed and implemented a project to provide resource specialization on conventional Linux clusters

  - Introduced **system-level resource specialization**
  - Confine Slurm compute node daemons (slurmd, slurmstepd) to a specific number or set of cores so that they do not interfere with application processes (confined on other cores)
  - Limit these processes to a specific amount of memory
  - New configuration parameters to control resource specialization

The Design Approach

- The Slurm administrator specifies the number of cores, or a list of specific cores, and the memory specialization limit (if desired), for each node using new node configuration parameters in slurm.conf.
  - Different nodes may have different numbers/lists of reserved cores and different memory limits.

- These parameters are applied **by default to all jobs** using the nodes.
  - Individual jobs may override the default parameters and allocate the reserved cores, using a command line option.
- Supported for SelectType=select/cons_res.

The Design Approach (cont.)

- Core specialization only makes sense if Slurm jobs are confined to their allocated resources, to prevent them from executing on the specialized CPUs
- Required configuration option to enable Core specialization
  - `TaskPlugin=task/cgroup in slurm.conf`
  - `ConstrainCores=yes in cgroup.conf`

- Without these options core specialization will have no effect and a warning message will be logged
- Similar approach to what is done when CPU frequency scaling is requested

Core Specialization Configuration and Usage

- The number of CPUs or a specific list of CPUs to specialize can be designated as part of the **node definition** using new parameters in the slurm.conf
  - `CoreSpecCount=<number of cores>`
  - `CPUSpecList=<comma separated list of CPU IDs>`

- CoreSpecCount and CPUSpecList are mutually exclusive.
- Size of the memory limit can be designated for each node in slurm.conf
  - `MemSpec=<memory limit in MB>`

Core Specialization Configuration and Usage

- If resources specialization is defined, individual jobs may override the default parameters and allocate the reserved resources using the command line option.
  - --core-spec=0 in srun/salloc/sbatch
  - AllowSpecResourceUsage = 1 in slurm.conf

- Default values for Linux systems
  - No Core specialization on any node
  - No Memory specialization limit on any node

- “scontrol show node” was enhanced to display the new parameters.

Implementation Details: Slurmd side

- Modifications upon slurmd startup

  - Recognizes & validates the new resource specialization configuration options
  - Determines which machine CPU IDs will be specialized
  - Invokes new functions to establish cgroups
    - init_system_cpuset_cgroup
    - init_system_memory_cgroup
  - Invokes new functions to establish specialization values for the node
    - set_system_cgroup_cpus
    - set_system_cgroup_mem_limit
  - Invokes new functions to attach itself to the system cpuset & system memory cgroups
    - attach_system_cpuset_pid
    - attach_system_memory_pid

Implementation Details: Slurmd side

- Slurmd Initialization
  - Processes resource specialization values
    - Determines specific CPUs for specialization
    - Determines memory limits
  - Establishes cgroups
    - init_system_cpuset_cgroup
    - init_system_memory_cgroup
  - Establish specialization values for the node
    - set_system_cgroup_cpus
    - set_system_cgroup_mem
  - Attaches itself to cgroups
    - attach_system_cpuset_pid
    - attach_system_memory_pid

Slurmd passes resource specialization information to Slurmctld in Node Registration Message

Implementation Details: Slurmctld side

- Modifications were made to node registration message handler
  - Invokes new function to build a core bitmap representing the node's specialized cores
  - node_spec_bitmap is a new member of node_record structure
- Resource selection logic was modified to exclude allocation of specialized cores on all nodes allocated to jobs

Implementation Details: Slurmctld side

- Slurmctld
  - Node Registration Message Processing
    - Builds core bitmap for node's specialized cores
    - Includes node_spec_bitmap in node_record
  - Resource Allocation Logic
    - Uses node_spec_bitmaps
    - Excludes specialized cores if usage not allowed
    - Allows allocation if Job can use specialized cores

Implementation Details: Slurmstepd side

- Modifications were made to slurmstepd startup
  - Invokes Core Specialization function to attach itself to the system cgroups
- Slurmstepd detaches automatically from cgroups when it terminates

Implementation Details: Slurmstepd side

Slurmstepd

Initialization

Attaches itself to the system cgroup created by slurmd

Termination

Automatically detaches from the cgroup

Example of usage

```
[root@ctld ~]$cat /etc/slurm/slurm.conf|grep CoreSpec
NodeName=mo[73-80] Procs=16 Sockets=2 CoresPerSocket=8 ThreadsPerCore=1
State=UNKNOWN RealMemory=30076 CoreSpecCount=1

[root@server]$scontrol show node=mo80
NodeName=mo80 Arch=x86_64 CoresPerSocket=8 CPUAlloc=0 CPUErr=0 CPUTot=16 CPULoad=4.
69 Features=(null)    Gres=(null)  NodeAddr=mo80 NodeHostName=mo80    CoreSpecCount=1
CPUSpecList=15
…
[root@ctld ~]$srun -N8 -n120 ./xhpl&
[root@mo80 ~]$ps -aux|grep slurm
27018
27872
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/system/cpus
15
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/system/tasks
27018
27872
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/uid_0/job_165/step_0/cpus
0-14
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/uid_0/job_165/step_0/tasks
27877
27878
...
[root@mo73 ~]# ps -u root -o pid,cpuid,comm
27018   15 slurmd
27872    15 slurmstepd
27877    0 xhpl
27878    1 xhpl
```

Example of usage

```
[root@ctld ~]$cat /etc/slurm/slurm.conf|grep Allow
AllowSpecResourcesUsage=1

[root@ctld ~]$srun --core-spec=0 -N8 -n120 ./xhpl&
[root@mo80 ~]$ps -aux|grep slurm
27018
27872
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/system/cpus
15
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/system/tasks
27018
27872
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/uid_0/job_166/step_0/cpus
0-15
[root@mo80 ~]$cat /cgroup/cpuset/slurm_mo80/uid_0/job_166/step_0/tasks
27877
27878
...
```

Conclusions

- Initial tests with HPLinpack on 8 nodes (16 cores per node) did not show any actual difference in performance.
  - This is due to the small scale of the application, the small number of cores and the type of MPI job.
- Experiments planned on larger scale and larger number of cores per node.
- Developments to ensure that overhead and noise will be as small as possible in upcoming architectures
- In many cores architectures (MIC) there is a real value in isolating slurm processes upon particular resources (cores, memory).
  - perhaps even other system processes

Architect of an Open World™