



Workshop on High Performance Computing & Parallel Programming Concepts

Pankaj Navani, Ravi Teja, Nachiket - Applications Analyst, Cray Inc. (Team @ SERC)



Program Brief



Day 1 - Sat. 10th Sept. 2016

Venue : SERC 4th floor Auditorium

Time : 10:00AM - 5:00PM

- ❑ Introduction
- ❑ SahasraT Architecture & Environment
- ❑ Parallel Programming Models
- ❑ Performance Monitoring and Tuning
- ❑ Lab Sessions

Day 2 - Sat. 17th Sept. 2016

Venue : SERC 4th floor Auditorium

Time : 10:00AM - 5:00PM

- ❑ Intel Processor Architectures
- ❑ Parallelization Techniques
- ❑ Vectorization techniques
- ❑ Thread Level Parallelism
- ❑ Intel Compiler Options – Quick Tweaks
- ❑ Performance and Math Kernel Libraries
- ❑ Accelerated Python for performance computing
- ❑ Lab Session



सहास्रत

Day-1 Agenda



Day 1 - SahasraT Induction For New-Scholars			
Timeline	Duration	Title	Agenda
10:00 - 10:10	10 min.	Opening Remarks - Prof. R. Govindrajan (Chairman, SERC)	Opening Remarks
10:10-10:50	40 min.	SahasraT Architecture - Pankaj Navani	Cray and Brief History of Supercomputing
			System Architecture
			Interconnect, Communications
			Programing Environment
			Scheduling Strategies & Job Submission
			Web / IISc Online Resources
10:50- 11:30	40 min.	Introductions Parallel Programming Models - Ravi Teja	Introduction to Parallel Computing
			Basic Terminology and Concepts
			Memory Architectures
11:30 - 11:50	20 min.	Coffee / Tea Break	
12:00- 13:00	60 min.	Parallel Programming Models Contd..	Multicore and Multimode
			Designing Parallel Programs
			Optimization Techniques with Examples Codes
13:00-14:00	60 min	Lunch Break	
14:00-15:00	60 min.	Performance Monitoring and Tuning - Pankaj Navani	Cray Tools
			Third Party Tools Available On The System
			Some Examples
15:00-15:20	20 min.	Coffee / Tea Break	
15:30-17:00	90 min.	Hands on Session (Pankaj, Ravi Teja and Nachiket)	LAB : Practice / Dummy Codes (Three Examples)

Cray: a long history of supercomputing...



***We build the world's fastest
supercomputers to help solve
"Grand Challenges" in science
and engineering***



**Government
and Defense**



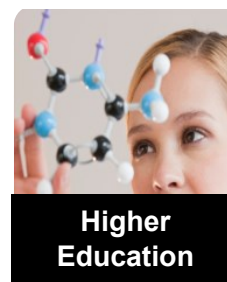
Energy



Manufacturing



Life Sciences



**Higher
Education**



**Financial
Services**



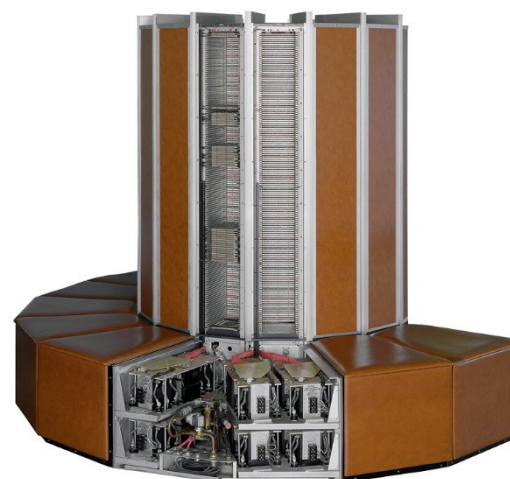
**Earth
Sciences**

Anything that can be simulated needs a Cray

We've Made Incredible Progress



	Zuse Z3	Cray-1		IISc "SaharaT" 8-cabinet Cray-XC40	
Introduced	1941	1976	(vs. Z3)	2015	(vs. Z3)
Flop Rate	$\sim \frac{1}{4} F$	~ 250 MF	$1e9$	~ 1.14 PF	$4.56e15$
Memory Size	~ 176 B	8 MB	$4.7e4$	172 TB	$10.7e12$





Petascale
Cray XC
Systems
Ordered &
Installed

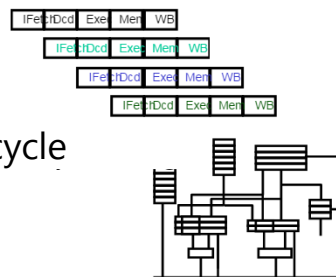
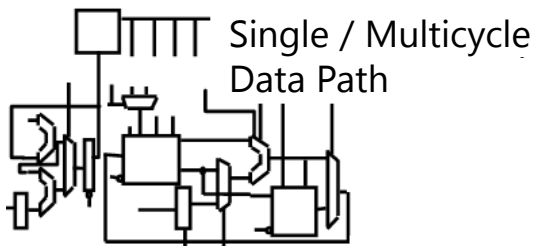
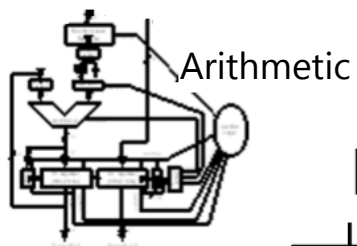
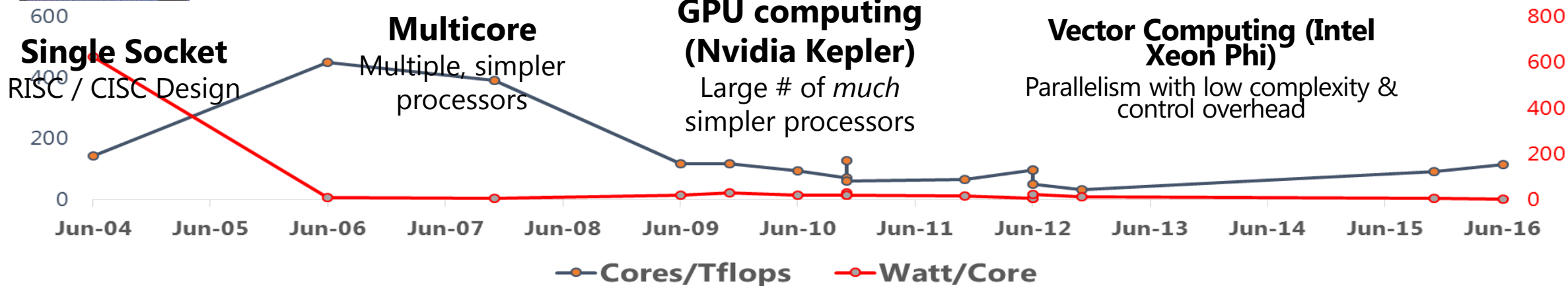
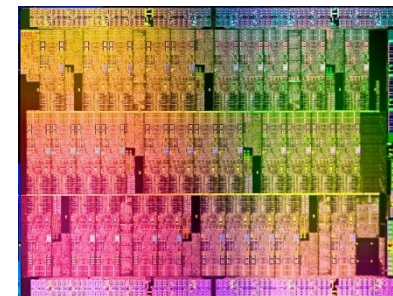
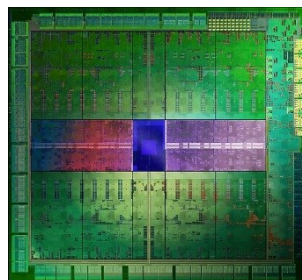
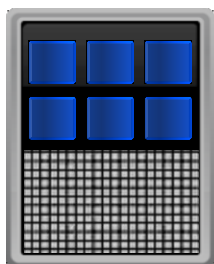
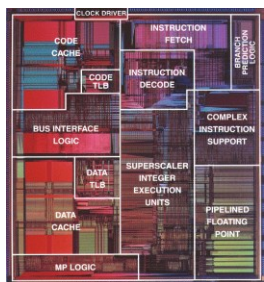


Technology Shifts and Responses...

- It has been 40 years since the first Cray-1 Supercomputer shipped to Los Alamos
- Optimal design has always been dependent on underlying technologies
 - Processors & Memories
 - Storage
 - Interconnects
- Shifts in these technologies can (and will) have large impacts on how systems look and how they are programmed
- Workloads and Users are also changing...



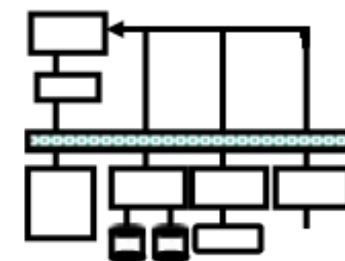
Architectural Response



Pipelining



Memory Hierarchy



IO Optimization

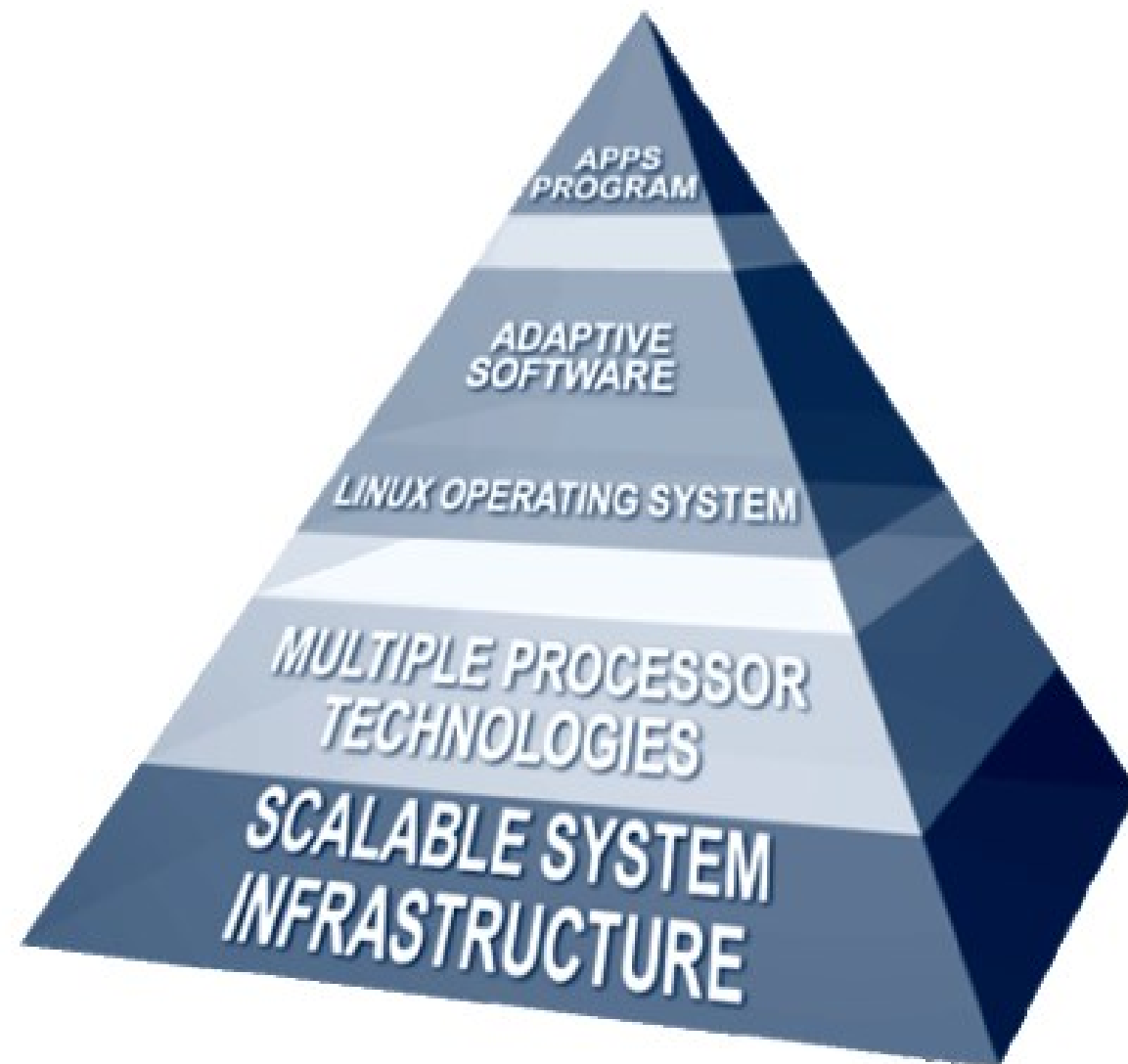
Cray Value Proposition



System Interconnect

Systems Management & Performance Software

Packaging



SahasraT – Interconnect



इसहस्राट

768 Sockets = 9216 cores less than 1 μ s away



2-cabinet Group



Backplane
connections within
chassis



Copper cables
between chassis



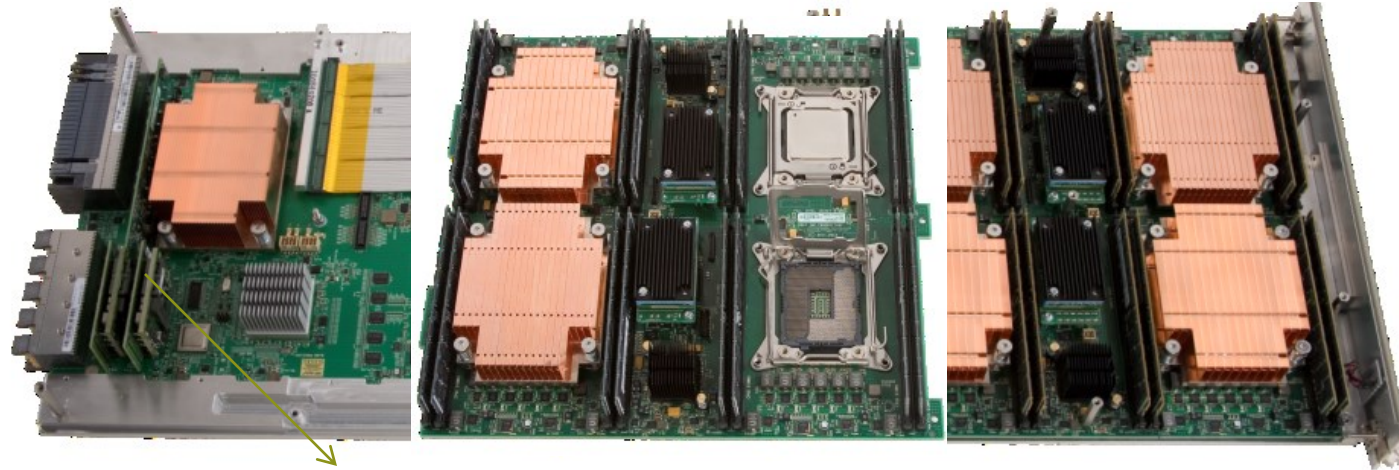
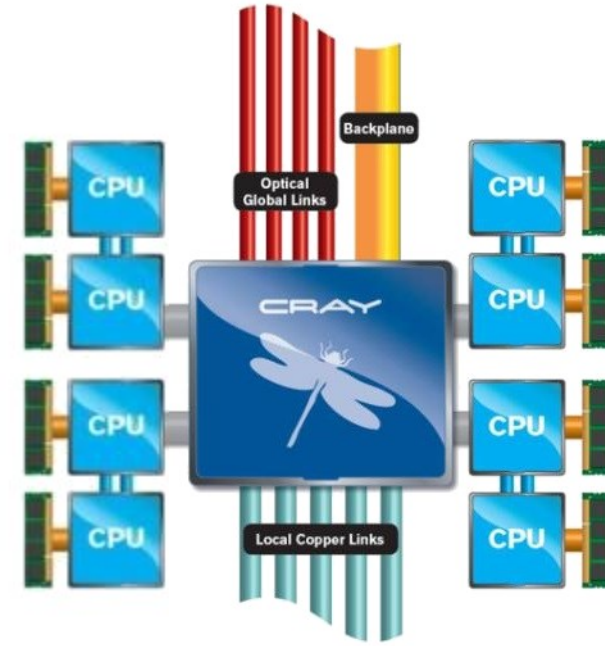
*This basic structure is
repeated in “**SahasraT**”*

इश्वरदत्त

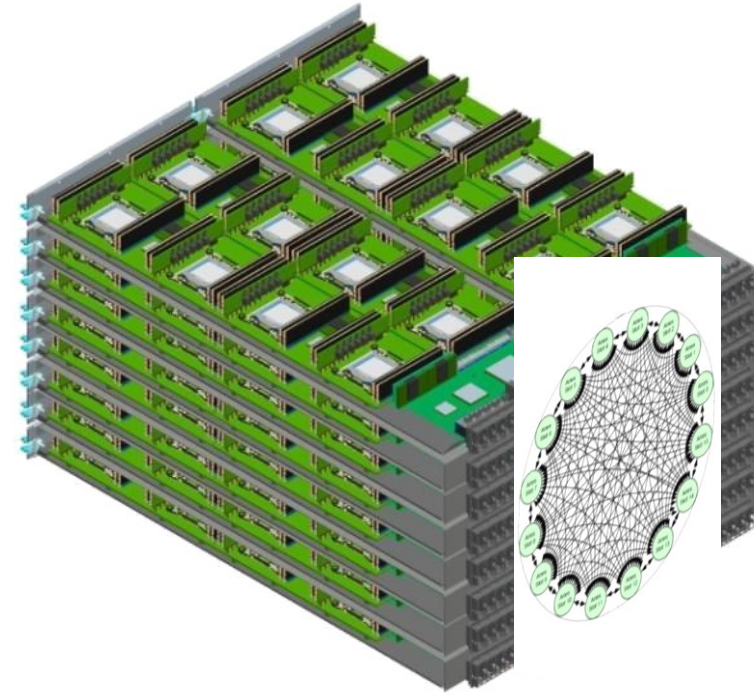
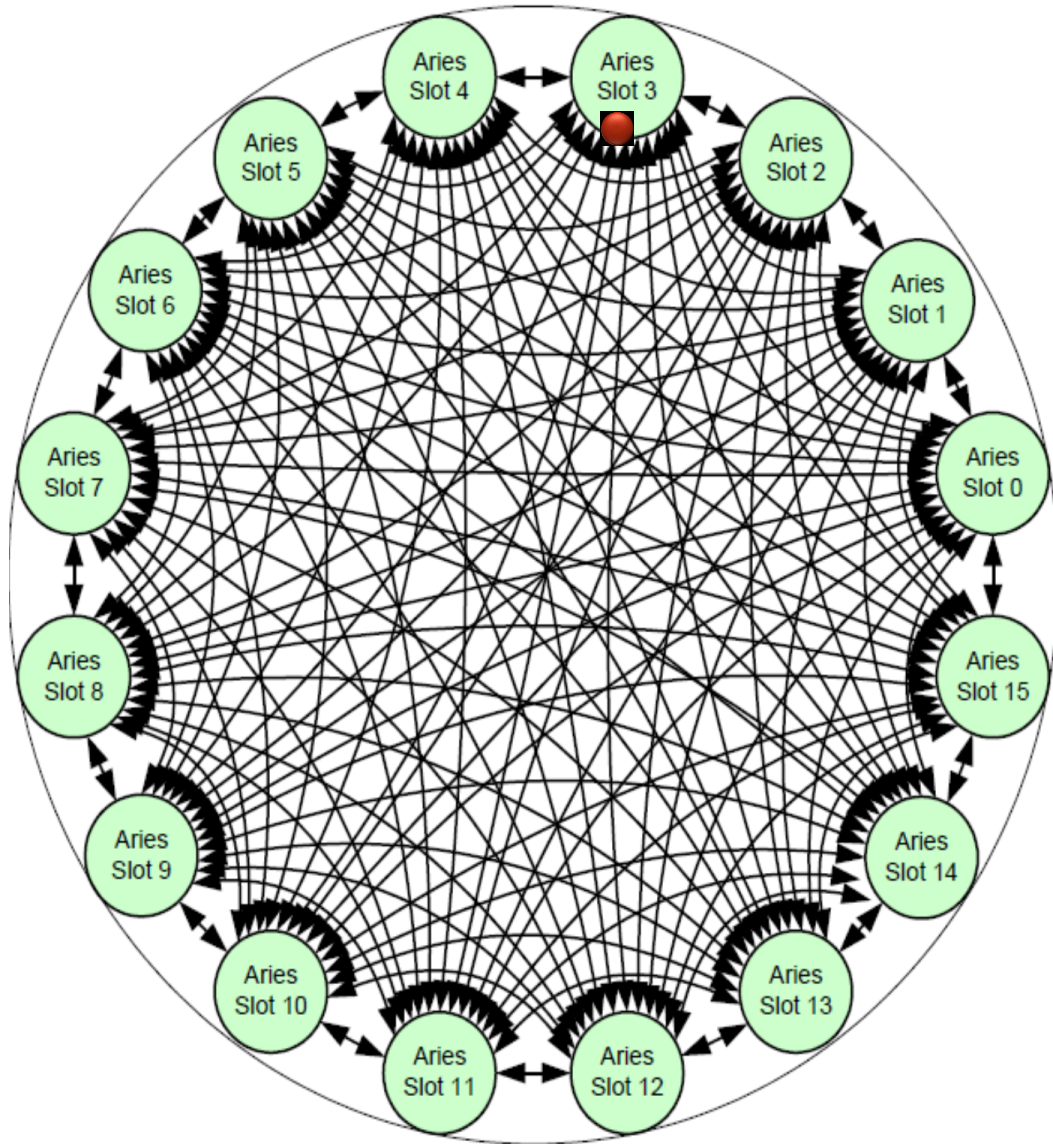
Compute Blade with Aries



- CPU Cluster - Intel Xeon E5-2680v3 @ 2.5GHz (Haswell) based 1376 compute nodes with a total count of 33024 cores (24 cores per node) with a sustained performance of 950 TFLOPS
- GPU Cluster - NVIDIA Tesla K-40 based 44 nodes (2880 cores per node) with a sustained performance of 52TFLOPS
- MIC Cluster - Intel XeonPhi 5120D Knights Corner based 48 nodes with a sustained performance of 28TFLOPS



Cray XC Rank-1 Network – Carried in Backplanes



- Chassis with 16 compute blades
- 128 sockets
- All-to-all within the backplane
- Per Packet Adaptive Routing

Cray XC Rank-2 Network



2-cabinet Group



Backplane
connections within
chassis



Copper cables
between chassis

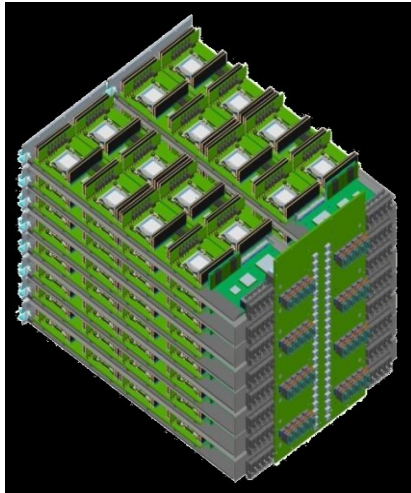


*This basic structure
is repeated in large
systems*

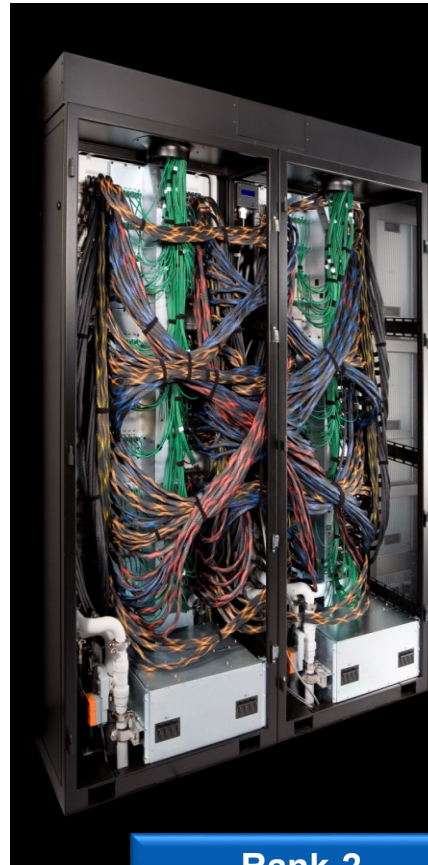
768 Sockets = 9216 cores less than 1 μ s away

इशावर

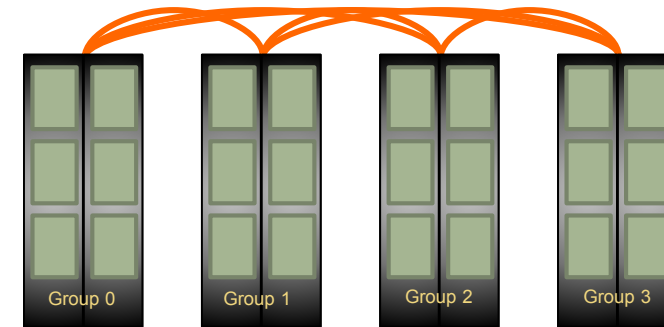
Cray XC Packaging Review



**Rank-1
Chassis**



**Rank-2
2 Cabinet Group**

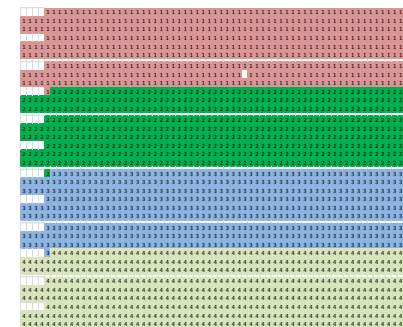


**Rank-3
Between Groups**

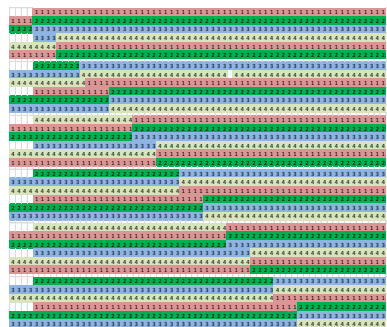
4 nodes = 1 blade; 16 blades = 1 chassis; 3 chassis = 1 cabinet; 2 cabinets = 1 group

Placement Insensitive – Dragon Fly Architecture

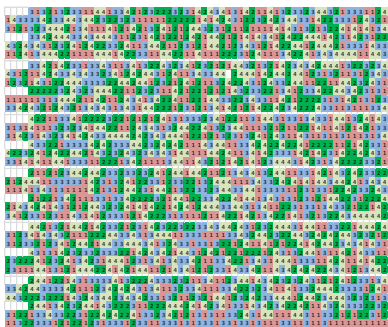
- Example: Sandia miniApp, miniGhost
- Running on 2256 node CSCS system (¼ global bandwidth)
 - Runtime in seconds for 100 cycles



Contiguous Blocks of 512 nodes			
69.0	68.8	68.9	68.9



Random blocks of 64 nodes			
69.4	69.4	69.4	69.5



Random layout of nodes			
70.9	71.0	70.6	70.5

Perfect Placement



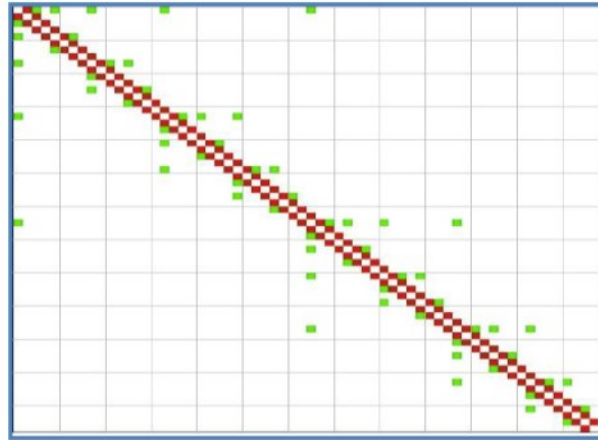
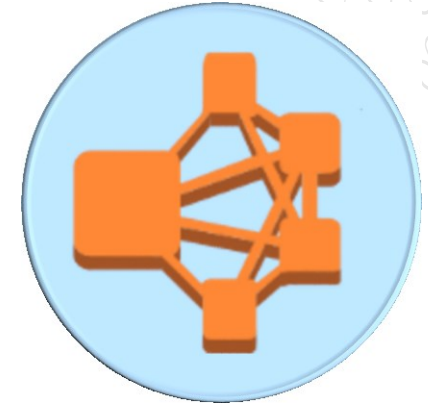
< 3% variance from best-case to worst-case placement

Worst-Case Placement



Connectionless Protocol – Makes it Scalable

CRAY®



Nearest Neighbor

Figure 17. AMG inter-processor communication CrayPat plot

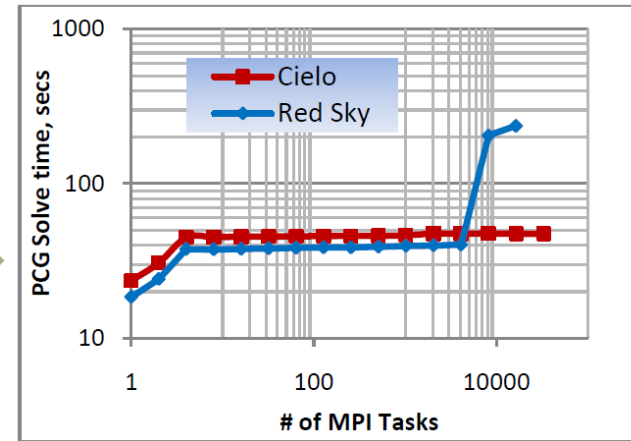
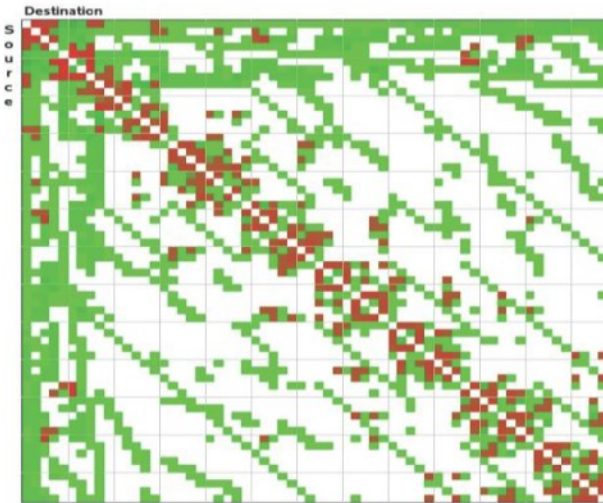


Figure 16. AMG Scaling Performance (Lower is better)



Irregular Pattern

Figure 9. Charon inter-processor communication CrayPat plot

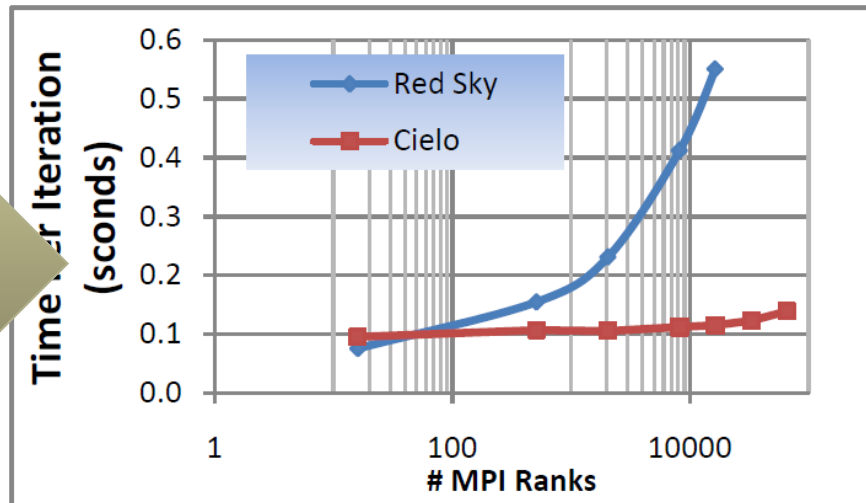
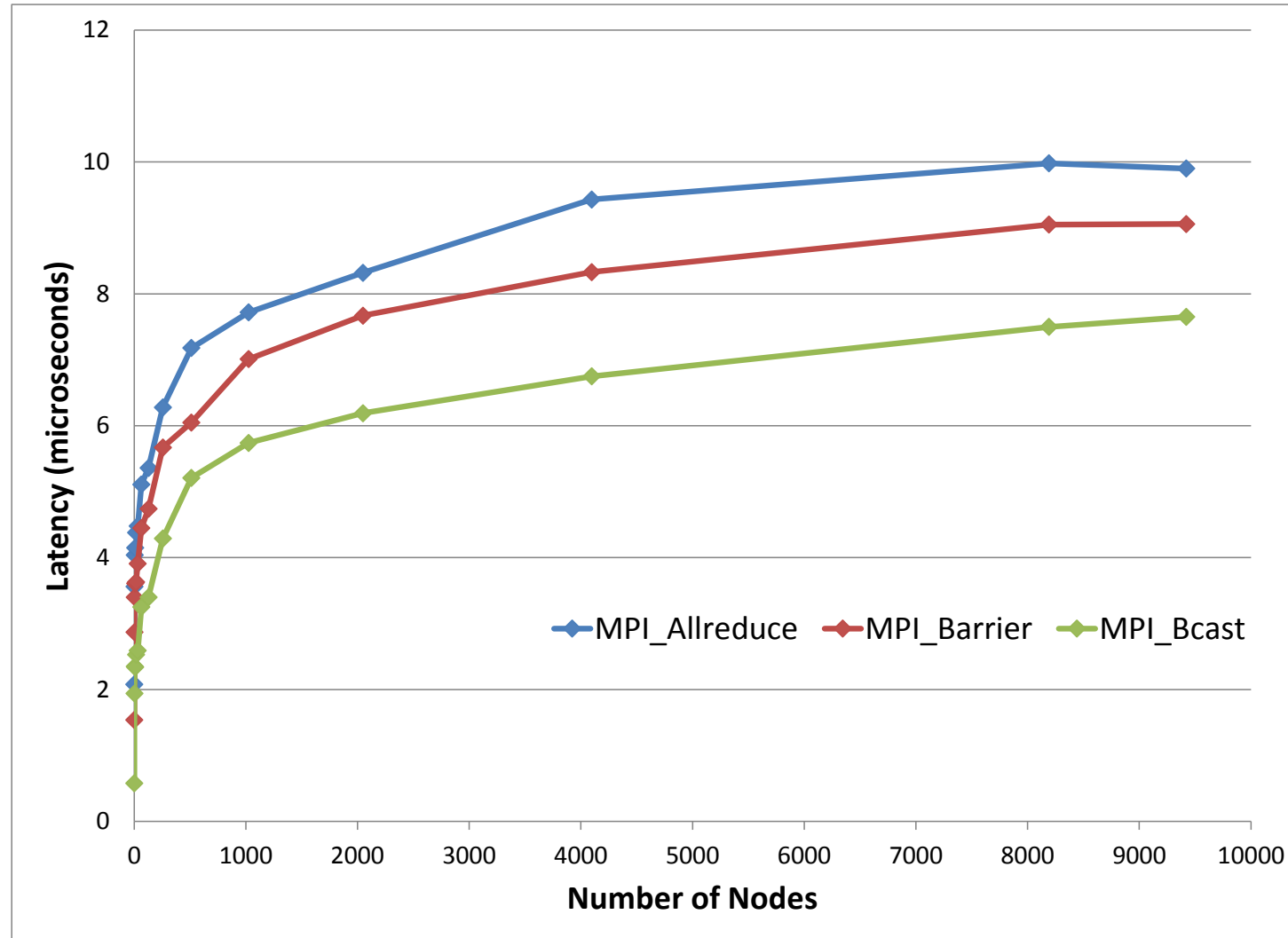


Figure 8. Charon scaling performance (Lower is better)

A Comparison of the Performance Characteristics of Capability and Capacity Class HPC Systems

By Douglas Doerfler, Mahesh Rajan, Marcus Epperson, Courtenay Vaughan, Kevin Pedretti, Richard Barrett, Brian Barrett, Sandia National Laboratories

MPI Collective Latency to 310,440 Ranks



Results are for up to 9420 nodes with 32 MPI ranks per node

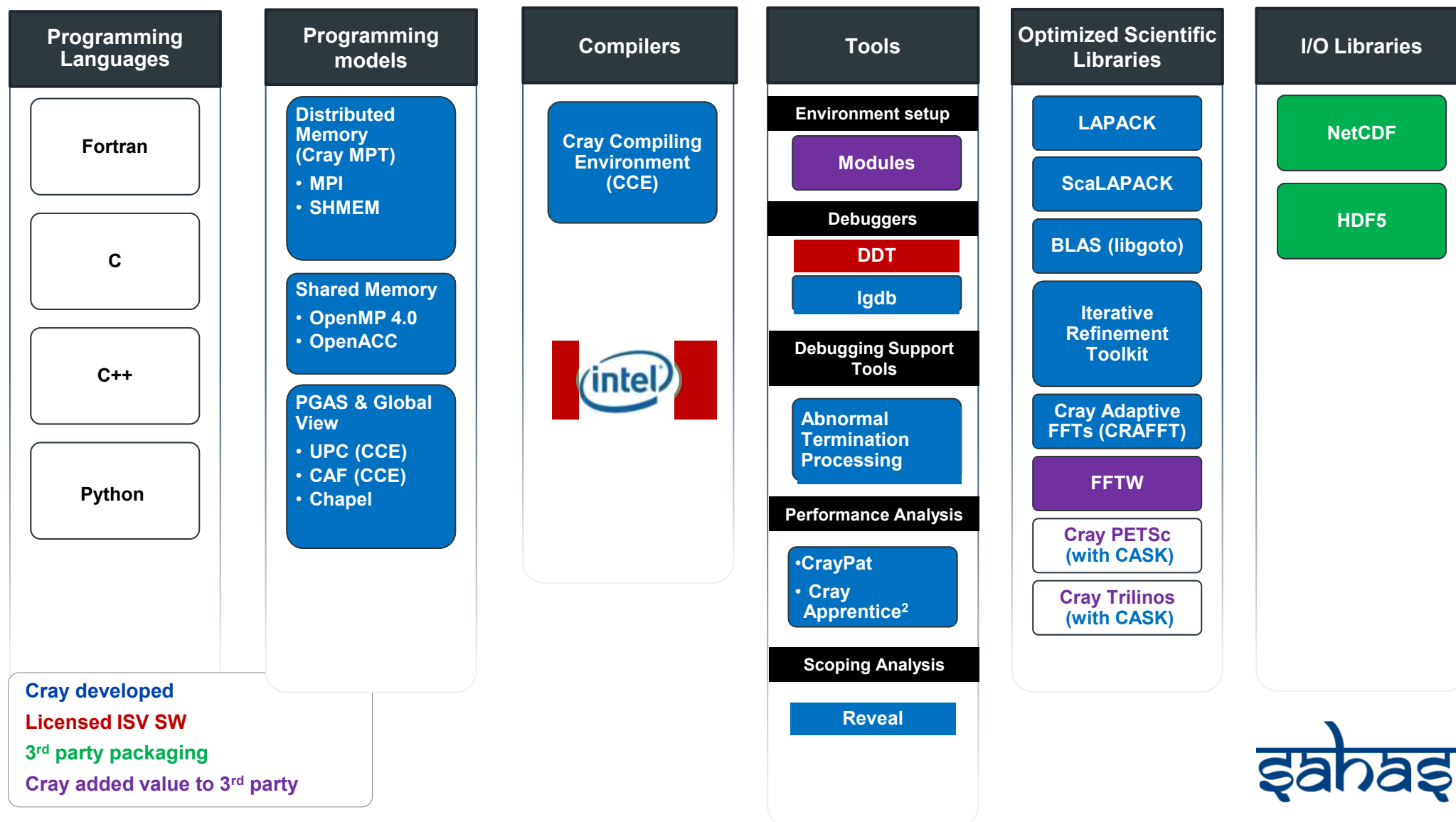
System Management & Performance Software



इवावइरात

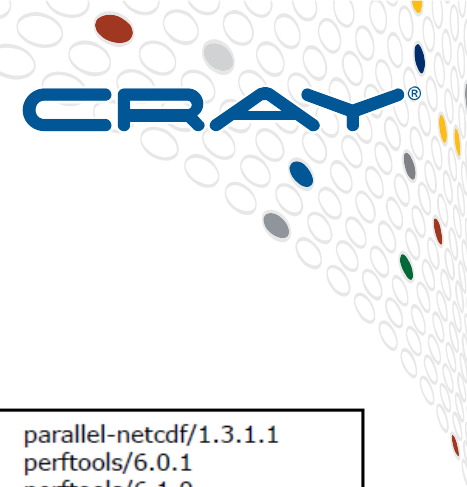
Cray Programming Environment Distribution

Focus on Performance and Productivity



इशावर

\$module avail



CRAY provides multiple (versions of) available compilers, libraries and tools:

- What **application** is really used by the user?
- How many **black-box** users are they?
- How do we know which (buggy) **library/compiler/tool** is used by the user?
- When can we get rid of **old or unused** versions?

PrgEnv-cray/5.0.15	cray-libsci/12.1.01	cray-trilinos/11.2.2.0	parallel-netcdf/1.3.1.1
PrgEnv-cray/5.0.29	cray-mpich/6.0.0	cray-trilinos/11.4.1.0	perftools/6.0.1
PrgEnv-gnu/5.0.15	cray-mpich/6.0.1	craype/1.04	perftools/6.1.0
PrgEnv-gnu/5.0.29	cray-mpich/6.0.2	craype/1.05	perftools/6.1.1
PrgEnv-intel/5.0.15	cray-netcdf/4.2.1.1	craype/1.06	perftools-lite/6.1.0
PrgEnv-intel/5.0.29	cray-netcdf/4.3.0	fftw/2.1.5.6	perftools-lite/6.1.1
atp/1.6.0	cray-netcdf-hdf5parallel/	fftw/3.3.0.2	stat/1.2.1.3
atp/1.6.1	4.2.1.1	fftw/3.3.0.3	stat/2.0.0.0
atp/1.6.2	cray-netcdf-hdf5parallel/	fftw/3.3.0.4	stat/2.0.0.1
atp/1.6.3	4.3.0	hdf5/1.8.11	
ccm/2.2.0-1.0500.37245.2	cray-parallel-netcdf/1.3.1.1	hdf5-parallel/1.8.11	Base-opts/
ccm/2.2.0-1.0500.40544.5	cray-petsc/3.3.05	iobuf/2.0.2	1.0.2-1.0500.40967.2.1.ari
cray-ga/5.1.0.1	cray-petsc/3.3.06	iobuf/2.0.3	cce/8.1.8
cray-ga/5.1.0.2	cray-petsc-complex/3.3.05	iobuf/2.0.4	cce/8.1.9
cray-hdf5/1.8.11	cray-petsc-complex/3.3.06	iobuf/2.0.5	chapel/1.7.0.1
cray-hdf5/1.8.9	cray-petsc-complex/3.4.2.0	lustre-cray_ari_s/	chapel/1.7.0.2
cray-hdf5-parallel/1.8.11	cray-shmem/5.6.4	2.2_3.0.42_0.7.3_1.0500.6	fftw/2.1.5.4
cray-hdf5-parallel/1.8.9	cray-shmem/5.6.5	924.14.1-1.0500.42026.19.	gcc/4.7.3
cray-lgdb/2.0.1	cray-shmem/6.0.0	63	gcc/4.8.0
cray-lgdb/2.0.2	cray-shmem/6.0.1	netcdf/4.3.0	gcc/4.8.1
cray-lgdb/2.0.3	cray-shmem/6.0.2	netcdf-hdf5parallel/4.3.0	intel/13.0.1.117
cray-lgdb/2.1.0	cray-tpsl/1.3.02	ntk/1.5.0	intel/13.1.0.146
cray-lgdb/2.2.0	cray-tpsl/1.3.03	onesided/1.5.0	intel/13.1.3.192
cray-libsci/12.0.00	cray-tpsl/1.3.04	papi/5.0.1	llm-utils/1.0.0
cray-libsci/12.0.01	cray-trilinos/10.12.1.1	papi/5.1.0.2	modules/3.2.6.6
cray-libsci/12.0.02	cray-trilinos/11.0.3.0	papi/5.1.1	modules/3.2.6.7
cray-libsci/12.1.00	cray-trilinos/11.0.3.1	parallel-netcdf/1.3.1	

\$module list

```
pankaj@clogin72:~$ module list
Currently Loaded Modulefiles:
 1) modules/3.2.10.4
 2) nodestat/2.2-1.0502.60539.1.31.ari
 3) sdb/1.1-1.0502.63652.4.25.ari
 4) alps/5.2.4-2.0502.9822.32.1.ari
 5) lustre-cray_ari_s/2.5 3.0.101_0.46.1_1.0502.8871.20.1-1.0502.21481.23.2
 6) udrdg/2.3.2-1.0502.10518.2.17.ari
 7) ugni/6.0-1.0502.10863.8.29.ari
 8) gni-headers/4.0-1.0502.10859.7.8.ari
 9) dmapp/7.0.1-1.0502.11080.8.76.ari
10) xpmem/0.1-2.0502.64982.5.3.ari
11) hss-llm/7.2.0
12) Base-opts/1.0.2-1.0502.60680.2.4.ari
13) cce/8.4.6
14) craype-network-aries
15) craype/2.5.4
16) cray-libsci/16.03.1
17) pmi/5.0.10-1.0000.11050.0.0.ari
18) rca/1.0.0-2.0502.60530.1.62.ari
19) atp/2.0.1
20) PrgEnv-cray/5.2.82
21) nodehealth/5.1-1.0502.65826.9.1.ari
22) pbs/12.2.404.152084
pankaj@clogin72:~$
```

\$module show

```
pankaj@clogin72:~$ module show pbs
-----
/opt/modulefiles/pbs/12.2.404.152084
prepend-path MANPATH /opt/pbs/12.2.404.152084/man
prepend-path PATH /opt/pbs/12.2.404.152084/bin
-----
```

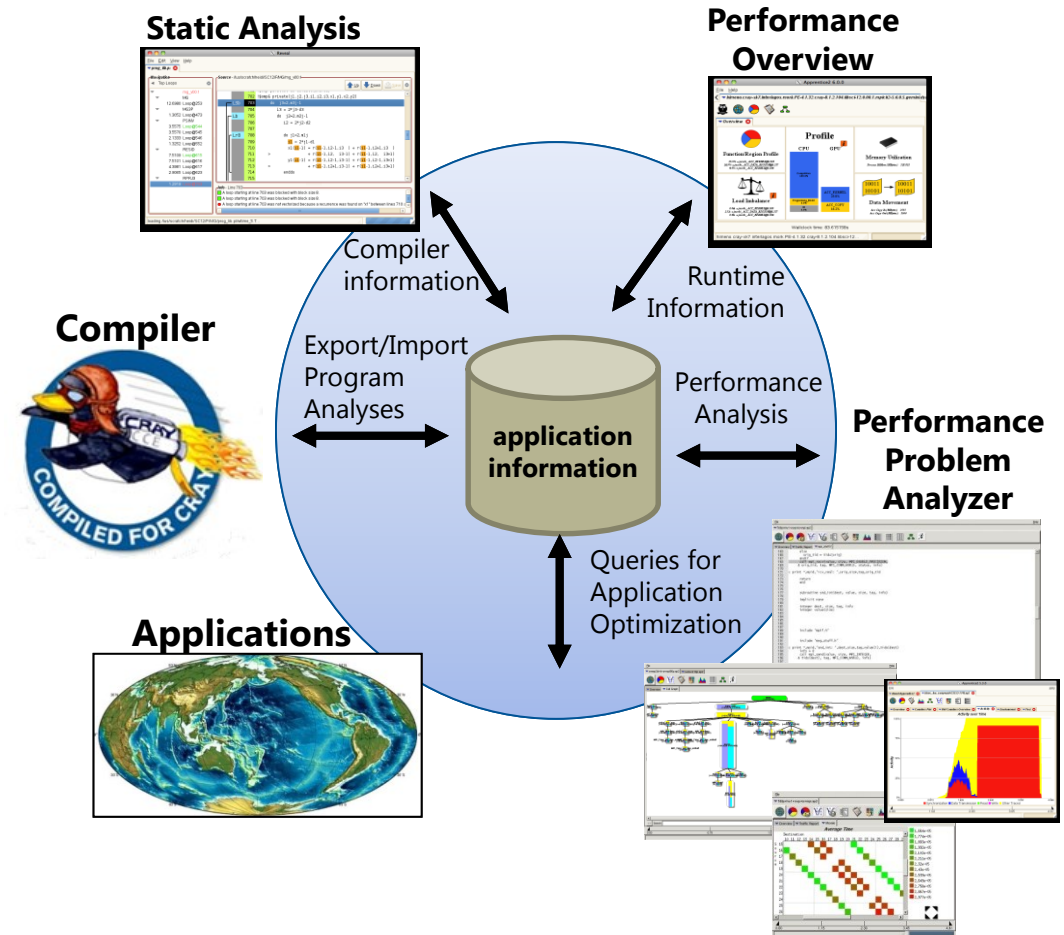
```
pankaj@clogin72:~$ module show cce
-----
/opt/modulefiles/cce/8.4.6:
conflict cce
setenv GCC_X86_64 /opt/gcc/4.8.1/snos
setenv CRAYLMD_LICENSE_FILE /opt/cray/cce/cce.lic
setenv CRAY_BINUTILS_ROOT /opt/cray/cce/8.4.6/cray-binutils
setenv CRAY_BINUTILS_VERSION /opt/cray/cce/8.4.6
setenv CRAY_BINUTILS_BIN /opt/cray/cce/8.4.6/cray-binutils/x86_64-unknown-linux-gnu/bin
setenv LINKER_X86_64 /opt/cray/cce/8.4.6/cray-binutils/x86_64-unknown-linux-gnu/bin/ld
setenv ASSEMBLER_X86_64 /opt/cray/cce/8.4.6/cray-binutils/x86_64-unknown-linux-gnu/bin/as
setenv CRAYLIBS_X86_64 /opt/cray/cce/8.4.6/craylibs/x86-64
setenv FTN_X86_64 /opt/cray/cce/8.4.6/cftn/x86-64
setenv CC_X86_64 /opt/cray/cce/8.4.6/CC/x86-64
setenv CRAY_CXX_IPA_LIBS /opt/cray/cce/8.4.6/CC/x86-64/lib/x86-64/libcray-c++-rts.a
setenv CRAY_FTN_VERSION 8.4.6
setenv CRAY_CC_VERSION 8.4.6
setenv PE_LEVEL 8.4
prepend-path FORTRAN_SYSTEM_MODULE_NAMES ftn_lib_definitions
prepend-path MANPATH /opt/cray/cce/8.4.6/man:/opt/cray/cce/8.4.6/craylibs/man:/opt/cray/cce/8.4.6/CC/man:/opt/cray/cce/8.4.6/craylibs/man
prepend-path NLSPATH /opt/cray/cce/8.4.6/CC/x86-64/nls/En/%N.cat:/opt/cray/cce/8.4.6/craylibs/x86-64/nls/En/%N.cat
prepend-path INCLUDE_PATH_X86_64 /opt/cray/cce/8.4.6/craylibs/x86-64/include
prepend-path PATH /opt/cray/cce/8.4.6/cray-binutils/x86_64-unknown-linux-gnu/bin:/opt/cray/cce/8.4.6/craylibs/x86-64/bin
prepend-path CRAY_LD_LIBRARY_PATH /opt/cray/cce/8.4.6/CC/x86-64/lib/x86-64:/opt/cray/cce/8.4.6/craylibs/x86-64
append-path MANPATH /usr/share/man
-----
```

\$module help

Cray Programming Environment Mission



- It is the role of the Programming Environment to **close the gap** between observed performance and achievable performance
- Provide a **tightly coupled** programming environment with compilers, libraries, and tools that will **hide the complexity** of the system
 - Address issues of **scale and complexity** of HPC systems
 - Target **performance** with **ease of use** based on extended **functionality** and increased **automation**
 - Close **interaction with users**
 - For feedback targeting **performance and functionality enhancements**



Sustained Performance on Real World Applications - Running the largest jobs, Most Nodes, at High Utilization

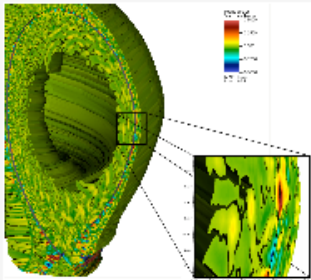


Machine Vitals

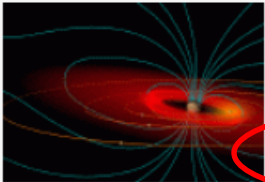


Edison	Cray XC30 Peak TFlop/s: 2,570 (2013)
Peak TFlop/s:	2570
Jobs running:	153
Jobs queued:	275
Cores in use:	128,784 (96%)
Backlog:	0.9 days

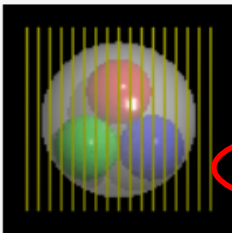
Top Jobs



Center for Edge Physics Simulation: SciDAC-3 Center
Office: Fusion Energy Sciences
Investigator: Choong-Seock Chang
Science Area: Fusion Energy
Cores: 30,720 (Edison)
Core Hours Used: 14,983.2



Computational studies in plasma physics and fusion energy
Office: Fusion Energy Sciences
Investigator: Abhay K. Ram
Science Area: Fusion Energy
Cores: 21,960 (Edison)
Core Hours Used: 330,521.4



Quantum Chromodynamics with four flavors of dynamical quarks
Office: High Energy Physics
Investigator: Doug Toussaint
Science Area: Lattice QCD
Cores: 18,432 (Edison)
Core Hours Used: 90,039.5

Interface | Policies & Job Management



इस वार्ड

How applications are run on a Cray XC

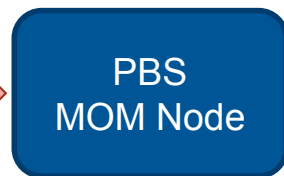
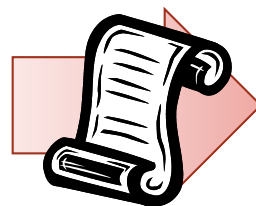
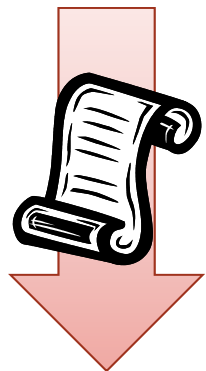
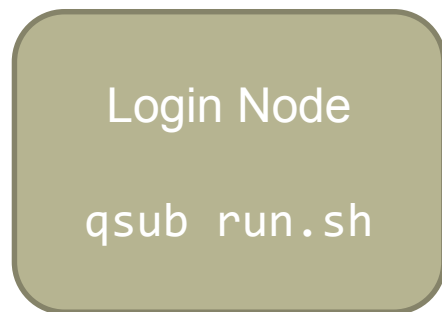
- **The Cray XC is a batch system.**
 - Users submit batch job scripts to the PBS scheduler from a login node for execution at some point in the future. Each job requires resources and a prediction of how long it will run.
 - The scheduler (running on an external server) chooses which jobs to run and allocates appropriate resources
 - The batch system will then execute the user's job script on an a different node than the login node (MOM node).
 - The scheduler monitors the job and kills any that overrun their runtime prediction.

- **User job scripts typically contain two types of statements.**
 1. **Serial commands** that are executed by the MOM node, e.g.,
 - quick setup and post processing commands, e.g., `rm`, `cd`, `mkdir`, etc.
 2. **Parallel executables** that run on compute nodes.
 1. Launched using the `aprun` command.

PBS on the XC40

- **Main PBS commands:**
 - `qsub` – Submit a batch script to SLURM.
 - `aprun` – Run parallel jobs.
 - `qdel`– Signal jobs under the control of SLURM
 - `qstat` – information about running jobs
- **The entire information about your simulation execution is contained in a batch script which is submitted via `qsub`.**
- **The batch script contains one or more parallel job runs executed via `aprun` (job step). Nodes are used exclusively.**
- **The simulations have to be executed on `/mnt/lustre/...`**
- **Useful environment variables:**
 - `PBS_NODEFILE`: “`cat $PBS_NODEFILE | uniq -c | sort`” is a file that shows you which nodes you are running on
 - `PBS_O_WORKDIR`: directory from which `qsub` was run

Running a batch job

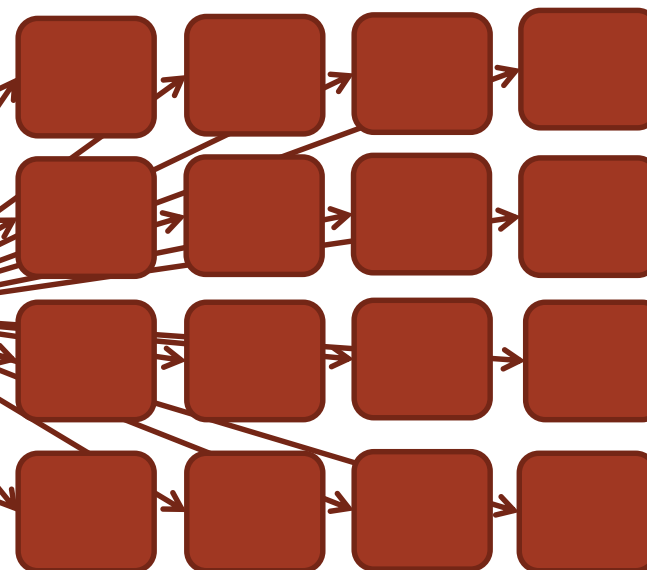


Scheduler
Resources

Serial
Parallel

Example Batch Job Script – run.sh

```
#!/bin/bash
#PBS -l
        select=10:ncpus=24:mpiprocs=24
#PBS -l place=scatter
cd $WORKDIR
aprun -n 240 -N 24 simulation.exe
rm -r $WORKDIR/tmp
```



Cray XC
Compute Nodes

The script will start by default in the directory where `qsub` has been executed. This directory is available in the environment variable `$PBS_O_WORKDIR`

Running an application on the Cray XC - ALPS + aprun



- **ALPS : Application Level Placement Scheduler**
- **aprun is the ALPS application launcher**
 - It **must** be used to run application on the XC compute nodes: interactively or in a batch job
 - If aprun is not used, the application is launched on the MOM node (and will most likely fail).
 - aprun launches groups of Processing Elements (PEs) on the compute nodes (PE == (MPI RANK || Coarray Image || UPC Thread || ..))
 - aprun man page contains several useful examples
 - The 3 most important parameters to set are:

Description	Option
Total Number of PEs used by the application	-n
Number of PEs per compute node	-N
Number of threads per PE (More precise, the “stride” between 2 PEs on a node)	-d

Script example (large queue on SERC system)

```
#!/bin/sh
#PBS -N jobname
#PBS -l select=343:ncpus=24
#PBS -l walltime=24:00:00
#PBS -l accelerator_type="None"
#PBS -j oe
```

! Date stamps at top and bottom of script for reference

```
date
```

! Useful to take note of where job is launched

```
cd $PBS_O_WORKDIR; pwd
```

! Don't necessarily need to load modules at runtime, but

! In case you do (e.g., for dynamic linking):

```
./opt/modules/default/init/sh
```

! (or "source /opt/modules/default/init/csh" for csh)

! Then can do "module load X", "module list" etc.

! Set up to run in the lustre directory /mnt/lustre for any

! parallel application (use diff directory for each run here)

```
RUNDIR=/mnt/lustre/USERNAME/myapp/run.$$
```

```
mkdir -p $RUNDIR
```

```
cd $RUNDIR
```

! Useful info when wondering later where run output might

! have gone

```
pwd
```

! Executable can be in lustre or in home directory! Here, let's copy it to our run directory

```
EXECDIR=/ufs/home/USERNAME/mybuilddir
```

```
cp $EXECDIR/a.out $RUNDIR
```

! Copy any input data you need or symlink it. Large input

! (and output) data files should be on lustre

```
cp /mnt/lustre/USERNAME/INPUTDATA/input_file .
```

! Useful info when looking back at run output

```
export MPICH_ENV_DISPLAY=1
```

```
export MPICH_VERSION_DISPLAY=1
```

! Run the executable. Use timers around aprun as a habit.

! This example uses linux "time" and also calculates walltime in

! a different way.

```
export beg_secs=`date +%s`
```

```
aprun -j 1 -n 8209 -N 24 ./a.out < input_file > output_file
```

```
export end_secs=`date +%s`
```

```
let wallsecs=$end_secs-$beg_secs
```

```
echo "Time taken in seconds is " $wallsecs
```

```
date
```

! Date stamps at top and bottom of script for reference

! Maybe write output_file to stdout if useful and not too huge

```
cat output_file
```

What resources did it use?

- **Can be good to record contents of \$PBS_NODEFILE during batch session to note what nodes were used (though list will be long if use lots of nodes!)**
 - `cat $PBS_NODEFILE | sort | uniq -c`
 - Or look at “apstat -avv apid” when job is running to see placement
- **See upcoming information on Cray Performance Tools**
 - perftools-lite is good place to start
- **For accelerators, environment variables are available to produce job statistics**

More info about my running job.... \$apstat



```
pankaj@clogin72:~> apstat
Compute node summary
  arch config  up   resv   use  avail  down
   XT   1468  1468  1457  1447   11    0

No pending applications are present

Total placed applications: 86
  Apid  ResId    User    PEs  Nodes   Age  State  Command
1155317 461546  aseabwit  240   10 48h11m  run   DNSFLO
1155617 461594  mecrks    240   10 45h39m  run
1155710 461616  aserame   240   10 44h57m  run   DNSFLO
1156577 461741  phypkv    936   39 36h47m  run   ph.x
1160832 461757  chejasj   960   40 1h16m   run   namd2
1160925 461938  mrcakash  216    9 0h20m   run   vasp
1158111 461957  chevkrn    1    1 22h36m  run  ostress-1_sm_
1158150 461963  ipcami    2400  100 22h25m  run  lmp_mpi_rigid
1158151 461965  mecanild 1176   49 22h25m  run   a.out
1158182 461969  chearit    1    1 22h15m  run  ostress-1_sm_
1158236 461977  cheravi    48    2 21h51m  run  nucl_mpi.exe
1158495 462015  cessahoo  480   20 19h51m  run   main.out
1158503 462018  metsoh    240   10 19h49m  run   vasp
1159541 462026  cheantya    1    1 10h07m  run  amplitude-sm_
1158572 462027  cheravi    1    1 19h10m  run  ostress-1_sm_
1159753 462030  mbumasha    1    1 8h11m   run  mdrun_mpi
1158654 462037  ugaka     480   20 18h40m  run   vasp
1158656 462038  mecakhilesh 1176  49 18h40m  run   a.out
1158662 462039  mecksptl   240   10 18h38m  run  lmp_mpi
1158846 462052  chepeter    72    3 17h13m  run  pmemd.MPI
1160561 462066  mbumasha    1    1 3h29m   run  mdrun_mpi
1158864 462069  esdsubha   960   40 17h05m  run   main.out
1158980 462084  esdghan    480   20 15h54m  run   main.out
1159026 462086  chejasj     1    1 15h15m  run  ostress-1_sm_
1159033 462087  esdsubha  2400  100 15h10m  run   main.out
1159051 462091  mecnidhi  1176   49 15h00m  run   a.out
1159067 462095  chearit     1    1 14h54m  run  ostress-1_sm_
1159073 462096  chesnm      1    1 14h49m  run  ostress-1_sm_
1159089 462099  mec dhgy   360   15 14h42m  run  lmp_mpi
1159121 462104  aseraghu   240   10 14h22m  run   DNSFLO
1159210 462119  metsoh     48    2 13h25m  run  lmp_jaguar
1159247 462124  esdveng    480   20 13h01m  run   main.out
```

```
pankaj@clogin72:~> apstat -avv 1158150
Total (filtered) placed applications: 1
  Apid  ResId    User    PEs  Nodes   Age  State  Comm
and
1158150 461963  ipcami    2400  100 22h27m  run  lmp_mpi_rigid

Application detail
Ap[15]: apid 1158150, pagg 0x600000e18, resId 461963, use
r ipcami,
      gid 1039, account 0, time 0, normal
Batch System ID = 181508.sdb
Reservation flags = 0x100000
Application flags = 0x142001
Created at Thu Sep  8 20:54:02 2016
Originator: aprun on NID 6, pid 11519
Number of commands 1, control network fanout 32
Network: cookies 0xc11c0000/0xc11d0000, NTTgran/entries
0/0
Cmd[0]: lmp_mpi_rigid -n 2400 -N 24 -j 1, 2730MB, XT, n
odes 100, exclusive
Placement list entries: 2400
Placement list: 163-164,208-218,233-234,246,252-253,280
-283,292-294,297-303,305-307,313,316,325-326,329-334,349-
350,364-369,372,378-379,392,395,424,472-473,485-489,506-5
07,509-511,516,518-545
pankaj@clogin72:~>
```

\$ xtnodestat -d

```
pankaj@clogin72:~> xtnodestat -d
Current Allocation Status at Fri Sep 09 19:25:08 2016
```

```

C0-0          C1-0          C4-0          C5-0
n3   ajaUaUaUaUaUaUaVajaUa2a2a2a2--   adaVacadadAAadadafa5aVacafaVaf   n3   adboa5a5a5a5bmafa5a5a5bhbobobo   bsbpbbsbsbsbsbuadbsadadadadadbrad
n2   SSajaUaUaUaUaUaUaUaVajaUa2a2a2a2--   SSaVaVaVadadadaVadafadaVacafaVaf   n2   adafa5a5a5a5bmbmafa5a5a5bhbobobo   bsbbspbbsbsbsbtbnadadadadadadbr
n1   SSajaUaUaUaUaUaUaUaVajaUa2a2a2a2--   SSaVaVaVadadadaVadadadaVaVaca9a9   n1   adafa5a5a5a5a5bmbmafa5a5bhbobobo   bsbbspbbsbsbsbsadadadadadadadbsbr
c2n0  ajaUaUaUaUaUaUaUaVajaUa2a2a2a2   a8adaVacadadadadadadaVaVaVa9a9   c2n0  afadboa5a5a5a5a5bmbmafa5a5bhbobobo   bsbbsadbsbsbsbsadadadadadadadbsbr
n3   akaqakauajajazaEaI--aNaTajaj   ahaca6a7a7aVa6a6a8aVaVaVa8a8ad   n3   afbnnbhbhbhbhafafbhbhbhbhbhafbdaf   afadadadadadadadadadadadadadbsbs
n2   SSamakapakatavajayaDaHaLaMaQavav   SSahaca6a7a7aVa7a6aVaVaVa8a8ad   n2   aibnnbhbnaafafafafafhbhbhbhbhbfbhd   bmadaadadadbpadbsadadadadadadbsbs
n1   SSakao--akasafajaxaCaGaK--aPajav   SSahaha6a7a7aVa7a6aVaVaVaVadaVa8   n1   afaiafbnafafafafafafhbhbhbhbhbfbd   bmafadadadadadadbsbsadadadadadbsbs
c1n0  akanakarakajajawaBaFaJ--aOajav   ahaha6a6a7aVa7a6aVa8aVa9a8a8aV   c1n0  afaiafbhafafafafafafhbhbhbhbhbfbd   bmafadadadadadbsbsbsadadadadadbsbs
n3   abaaacacac--aeafaeaeahaiiaiaf   aiaiaVaVaiaiaia3a4a4a5a4a4ah   n3   bhacaibdbibikadbcfafafa8a8aiaf   a5afafafhbhbmbmbma5adbqbmmbmbmafbm
n2   SSSSabaacacac--aeafaeaeahaiiaiaf   SSSSaiaiaVaVaVaiaia3aVa4a5aVa4ah   n2   bhbdbdbhbhbhbikadbcfafafa8a8bhaf   a5a5afafhbhbmbma5bmbqbmmbmbmbmbmh
n1   SSSSaaaaaacac--acafaeaeahaiiaiaf   SSSSaiaiaVaVaVaiaia3aVa4a5a5a4aV   n1   bhbdbdbhbhbhbhbicadbcavafafa8a8bmbh   a5a5afafhbha5bma5bmafmbmbmbmbmbmbm
c0n0  aaaaaaacacSSadafafaeAAaiaiaj   ajaiaVaVaVSSaiaia3a4a5a5a4aV   c0n0  bdbdaibhbhbhbhbSSaiaiafa8a8a8ai   bo5afafbpbhbmbmbmbqadbmbrmbmbmbh
s00112233445566778899aabbccddeeff   00112233445566778899aabbccddeeff   s00112233445566778899aabbccddeeff   00112233445566778899aabbccddeeff

C2-0          C3-0          C6-0          C7-0
n3   aVaVaVaVaVaVaVaVaVa5a3adadadbd   bfbgbgaiadadadadadadbdadbdadbdad   n3   adadadadadadadadadadadadadadadad   bEafafadadadadadadadadadadadad
n2   SSaVaVaVaVaVaVaVaVaVa5a3adadadbd   SSbfbgbgaiadadadadadadbdadbdadbdad   n2   adadadadadadadadadadadadadadadad   bEafafafbkafadadadadadadadadadad
n1   SSbbaVaVaVaVaVaVaVaVaVa3a3adadadbd   SSbfbgbgbgaiadadadadadadadadadadad   n1   adadadadadadadadadadadadadadadad   bpbmafadadadadadadadadadadadad
c2n0  aVaVaVaVaVaVaVaVaVa5a3adadadbd   bfbgbgbgaiadadadadadadadadadadad   c2n0  adadadadadadadadadadadadadadadad   adadadadadadadadadadadadadadadad
n3   ada3adbbba3acbcbaVa3avavbcaVaV   bebbbebebebebebebebebebebebebebe   n3   bbbhbhbhbhbhbhbhbhbhbhbhbhbhbhbhb   bpbpbpafbzbzbpadadadadadadadad
n2   SSada3baacbbba3acbcbaVa3avavbcaVaV   SSbebebebebebebebebebebebebebebe   n2   bbbhbhbhbhbhbhbhbhbhbhbhbhbhbhbhb   btbpaftbzbzbpadadadadadadadadad
n1   SSada3baa8bba3acbcbaVa3avavbcaVaV   SSbebebebebebebebebebebebebebebe   n1   bbbhbhbhbhbhbhbhbhbhbhbhbhbhbhbhb   bibtbtbtbpbzbzbpadadadadadadadad
c1n0  ada3a3avbba3acbaVa3avavbca3   bebebebebebebebebebebebebebebe   c1n0  bbbhbhbhbhbhbhbhbhbhbhbhbhbhbhbhb   adbpbsbtbzbzbpadadadadadadadadad
n3   aVafadadadadadadadadadadadadadad   bdadadadadadadadadadadadadadadbe   n3   bjbjaadbbhbhbhbhbhbhbhbhbhbhbhbhb   bwbwbwadafadbpbpbpbwafadbpafbwbi
n2   SSSSaafadadadadadadadadadadadadad   SSbdadadadadadadadadadadadadadadbe   n2   bjbjbibhbhbhbhbhbhbhbhbhbhbhbhbhb   adbwbbwbmbadbpafbpbxadadadadadad
n1   SSSSaafadadadadadadadadadadadadad   SSbdadadadadadadadadadadadadadadbe   n1   bjbjbjbhbhbhbhbhbhbhbhbhbhbhbhbhb   adadadadadadadadadadadadadadadad
c0n0  aVadafadadSSada3aVadadadadadad   bdadadadadadadadadadadadadadadbe   c0n0  adbjbjbhbhbhbhbhbhbhbhbhbhbhbhbhb   adbwadadadadadadadadadadadadadad
s00112233445566778899aabbccddeeff   00112233445566778899aabbccddeeff   s00112233445566778899aabbccddeeff   00112233445566778899aabbccddeeff
```

Legend:

nonexistent node	S service node
; free interactive compute node	- free batch compute node
A allocated (idle) compute or ccm node	? suspect compute node
W waiting or non-running job	X down compute node
Y down or admin down service node	Z admin down compute node

Available compute nodes: 0 interactive, 10 batch

You have a VERY busy machine...!

Queues on SERC System

```
pankaj@clogin72:~> qstat -q
```

```
server: sdb
```

Queue	Memory	CPU Time	Walltime	Node	Run	Que	Lm	State
phi_nodes	--	--	--	--	0	0	--	D S
ccm_queue	--	--	--	--	0	0	--	D S
temp0	--	--	168:00:0	--	1	0	--	E R
gpu_nodes	--	--	--	--	0	0	--	D S
cpu_nodes	--	--	--	--	0	0	--	D S
batch	--	--	--	--	0	20	--	E R
workq	--	--	--	--	0	0	--	D S
large	--	--	24:00:00	--	0	0	--	E R
medium	--	--	24:00:00	--	8	17	--	E R
small172	--	--	72:00:00	--	15	16	--	E R
small	--	--	24:00:00	--	20	38	--	E R
gpu	--	--	24:00:00	4	30	20	--	E R
mgpu	--	--	24:00:00	24	1	3	--	E R
xphi	--	--	24:00:00	--	2	0	--	E R
idqueue	--	--	02:00:00	--	9	22	--	E R
					86	136		

Batch Strategies and Queues



Queue name: Batch

Queue type: Route

Max_queued_by_each_user: 2

Route destinations: idqueue, small, small72, medium, large, gpu, xphi

=====

Queue Name: idqueue

Queue Type: Execution (This queue is meant for interactive debugging sessions of test runs of codes)

Job type: CPU MPI based/openmp based

Max_job_queued_per_user: 2

Core ranges: 24 – 256 ~ 10 nodes

Max_walltime: 2hrs

Max_user_job_run: 1

Total_job_runs: 32

Batch Strategies and Queues



All the other queues below are for production runs once the code has been verified for correct execution.

Queue Name: small
Queue Type: Execution
Max_job_queued_per_user: 3
Job type: CPU MPI based/openmp based
Core ranges: 24 – 1032
Max_walltime: 24hrs
Max_user_job_run: 2
Total_job_runs: 20

=====

Queue Name: small72
Queue Type: Execution
Max_job_queued_per_user: 1
Job type: CPU MPI based/openmp based
Core ranges: 24 – 1032
Max_walltime: 72hrs
Max_user_job_run: 1
Total_job_runs: 15

=====

Queue Name: medium
Queue Type: Execution
Max_job_queued_per_user: 1
Job type: CPU MPI based/openmp based
Core ranges: 1033 - 8208
Max_walltime: 72hrs
Max_user_job_run: 1
Total_job_runs: 10

Queue Name: large
Queue Type: Execution
Max_job_queued_per_user: 1
Job type: CPU MPI based/openmp based
Core ranges: 8209 - 22800
Max_walltime: 24hrs
Max_user_job_run: 1
Total_job_runs: 4

=====

Queue Name: gpu
Queue Type: Execution
Job Type: Cuda based code/Opencl code/ GPU applications
Max_job_queued_per_user: 5
Core ranges: 1 – 48
Min no. of accelerators (Nvidia): 1
Max no. of accelerators (Nvidia): 4
Max_walltime: 24hrs
Max_user_job_run: 3
Total_job_runs: 30

=====

Queue Name: xphi
Queue Type: Execution
Job Type: intel-xeon phi coprocessor job(offload mode is supported in Cray)
Max_job_queued_per_user: 3
Core ranges: 1 - 480
Max_walltime: 24hrs
Max_user_job_run: 2

SERC Online Resources



The screenshot shows the SERC website with a blue header containing the text 'Supercomputer Education & Research Centre'. Below the header is a navigation bar with links for 'COMPUTING SYSTEMS', 'SOFTWARE', 'PEOPLE', 'SUPPORT', 'TOP SUPERCOMPUTERS OF INDIA', and 'CDS'. A sidebar on the left lists various resources like 'CRAY XC40 - "SAHASRAT"', 'GANGLIA CLUSTER MONITOR', 'LICENSE WATCH', 'USER INFORMATION', 'PUBLICATIONS', 'MATLAB TAH MODEL - NEW', 'SERC WEBMAIL', and 'SATP-SYSTEM ADMINISTRATION TRAINING PROGRAMME'. The main content area is titled 'SAHASRAT' and features a banner image of the IISc building. Below the banner is an 'Introduction' section describing the Cray XC40 system and its components.

Ongoing Training Sessions

- SERC User Training Session(19/01/2015) ([PDF](#)) ([Video](#))
- Cray-Intel Training Sessions
 - Day 1(14/05/2015) ([PDF1](#)) ([PDF2](#)) ([Video](#))
 - Day 2(15/05/2015) ([PDF](#)) ([Video](#))
- Cray Workshop on High Performance Computing Tools
 - Day 1(26/10/2015) ([PDF1](#)) ([PDF2](#)) ([PDF3](#)) ([PDF4](#)) ([PDF5](#))
 - Day 2(27/10/2015) ([PDF1](#)) ([PDF2](#)) ([PDF3](#)) ([PDF4](#)) ([PDF5](#))
- Allinea DDT Workshop (29/02/2016)([PDF](#)) ([Video](#))
- Workshop on Science Using Sahasrat (11/05/2016)
 - A Study with an Earth System Model – Ravi S. Nanjundiah, Prof. ([PDF](#))
 - Defects in Materials – Manish Jain ([PDF](#))
 - High throughput Computational design of Thermoelectric Materials – Abhishek K. Singh ([PPT](#))
 - Cray Roadmap to Exascale – Hee-Sik Kim, Cray APAC ([PDF](#))

Accessing the system

The XC40 has login nodes, through which the user can access the machine and submit jobs. The machine is accessible for login using ssh from inside IISc network (ssh computational_id@sahasrat.serc.iisc.ernet.in). The machine can be accessed after applying for Cray XC40, for which:

Fill the online HPC application form [here](#) and submit at Room no:116, SERC.

HPC Application form must be duly signed by your Advisor/Research Supervisor.

<http://www.serc.iisc.in/facilities/cray-xc40-named-as-sahasrat/>

For any queries, email to [helpdesk_serc](#) or please contact System Administrator, #109,SERC.
Cray Applications Analyst team can be aproaced via SERC System Admin Group

Parallel Computing



इवावश्वत

Agenda

- 1. Introduction to Parallel Computing**
- 2. Basic Terminology and Concepts**
- 3. Memory Architectures**
- 4. Multicore and Multi-node programming**
- 5. Designing Parallel Programs**

What is Parallel Computing?

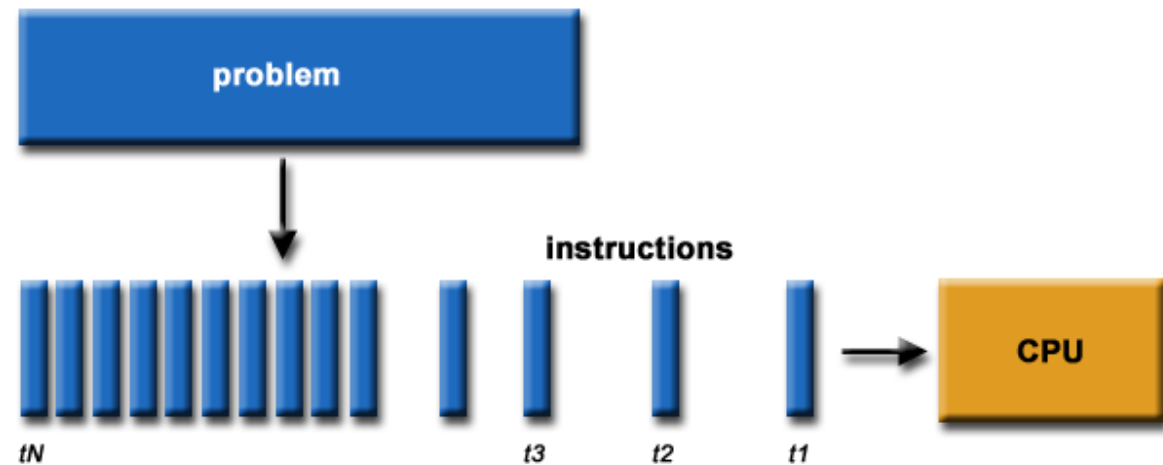


इतिवृत्त

What is parallel computing?

Sequential programming:

- Runs on a single CPU
- Computation is modeled after problems with a chronological sequence of events.
- Processes are run one after another

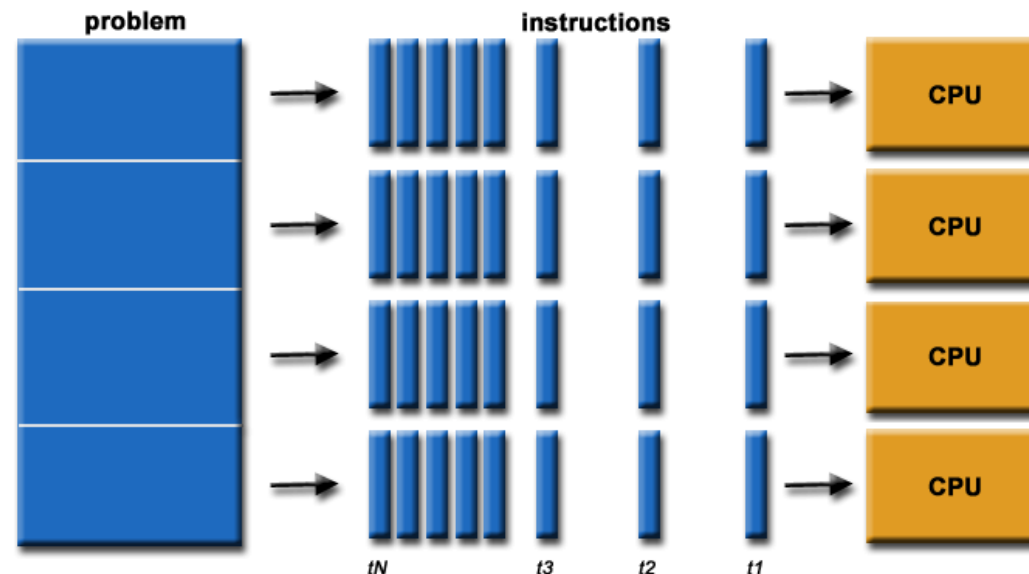




What is parallel computing?

Parallel computing is the use of two or more processors (threads, cores or computers) in combination to solve a single problem.

- Runs on multiple CPUs concurrently
- Computation is modeled into discrete parts that can be solved concurrently
- In each part, processes are run one after another simultaneously on different CPUs



Basic Terminology and Concepts



इवावइवा

Basic Terminology

Instruction :

An order given to a computer processor by a program. Tells the processor what to do

Process :

- **An execution instances of a Program.**
- **Executes in a sequence of Instructions**
- **A process is always stored in the main memory also termed as the primary memory or random access memory (RMA).**
- **Several process associated with a single program**

Source : Operating System Concepts by Abraham Silberschatz , Peter B. Galvin , Greg Gagne

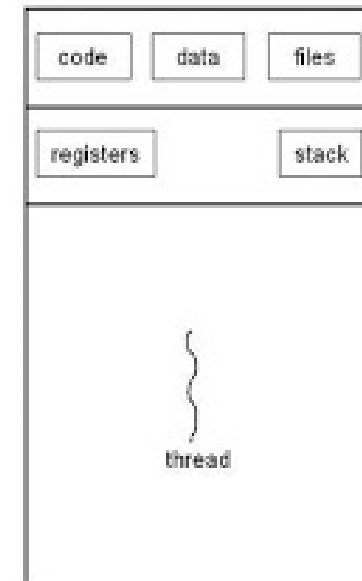
Basic Terminology

Thread

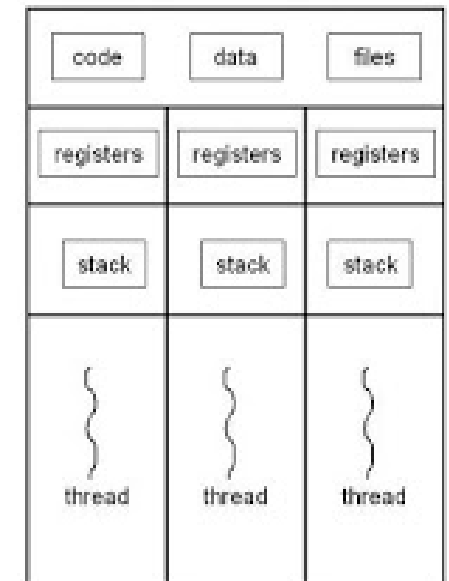
A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.

- Light-weight process
- Executes independently
- A process has only one thread of control
- A process executes Instructions concurrently
- Multiple threads can execute on a multiprocessor systems
- Threads running within a process shares resources such as address space, Stack and other process information
- Threads are created using Posix libraries in C language

Single-threaded process



multithreaded process



Source : Operating System Concepts by Abraham Silberschatz , Peter B. Galvin , Greg Gagne

Basic Terminology

Task

A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.

Parallel Task

A task that can be executed by multiple processors

Serial Execution

Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.

Parallel Execution

Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.

Shared Memory

A computer architecture where all processors have direct access to common physical memory.

Source : https://computing.llnl.gov/tutorials/parallel_comp

Basic Terminology

Distributed Memory

Network based memory access for physical memory that is not common.

Communications

- Parallel tasks typically need to exchange data. Through a shared memory bus or over a network
- The event of data exchange is commonly referred to as communications.

Synchronization

- The coordination of parallel tasks in real time associated with communications.
- Implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point.
- Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

Granularity

In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.

- **Coarse:** relatively large amounts of computational work are done between communication events
- **Fine:** relatively small amounts of computational work are done between communication events

Source : https://computing.llnl.gov/tutorials/parallel_comp

Parallel Computing Concepts



इवावइवात



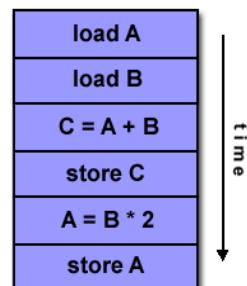
Parallel processing systems achieve parallelism by having more than one processor performing tasks simultaneously. There are many different ways to organize the processors and memory. One of the more widely used classifications is called Flynn's Taxonomy.

- **Flynn's taxonomy** : It is based on Instruction and Data processing. A computer is classified by whether it processes a single Instruction at a time or multiple Instructions simultaneously, and whether it operates on one or multiple Data sets.

Parallel Computing Concepts – Flynn Classification

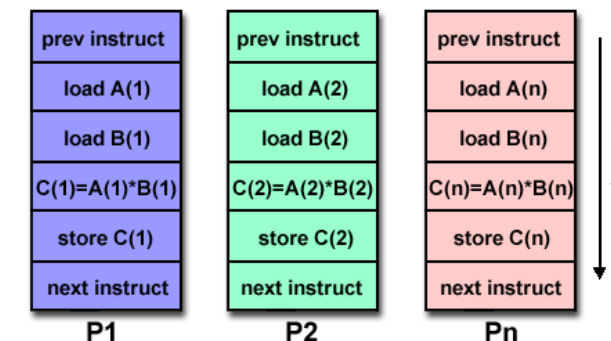
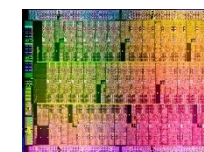
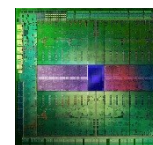
SISD

Single Instruction with Single Data



SIMD

Single Instruction with Multiple Data



MISD

Multiple Instruction with Single Data

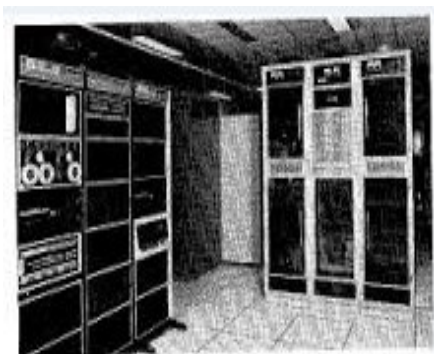
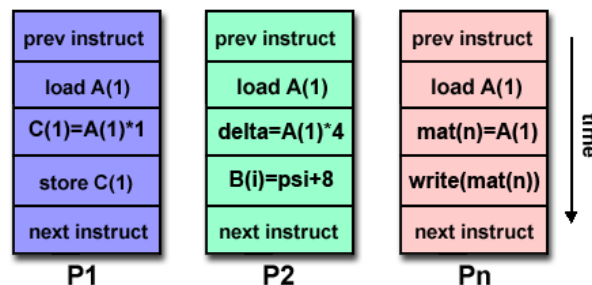
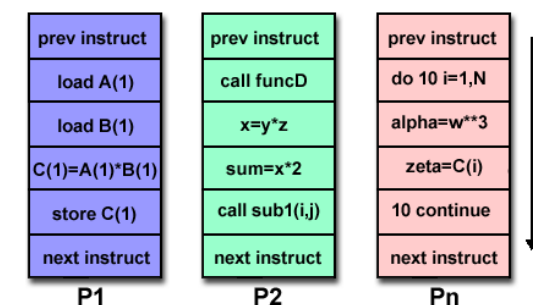


Fig. 1. The Cray T3E multiprocessor.



MIMD

Multiple Instruction with Multiple Data



Memory Architectures



इतिवृत्त

Memory architectures

- **Shared Memory**
- **Distributed Memory**
- **Hybrid Memory**

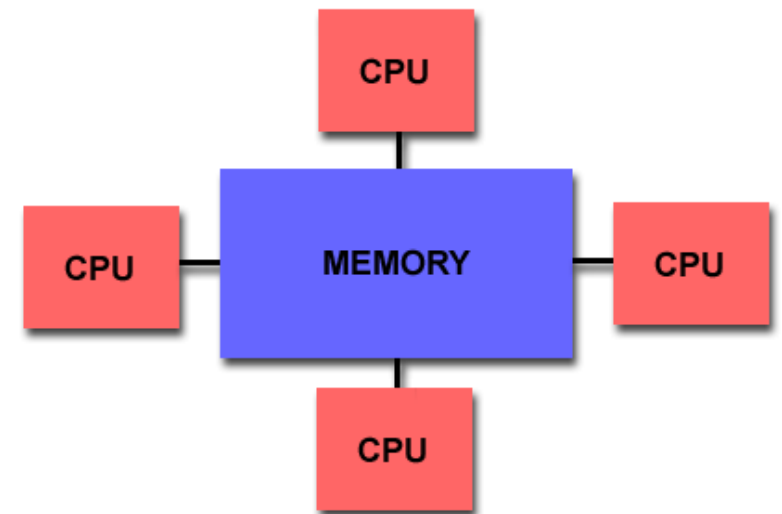


Shared Memory

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: *UMA* and *NUMA*.

Programming Models:

- MPI (Message Passing Interface)
- openMP API (Open Multi-Processing)
- pthreads etc





Shared Memory

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

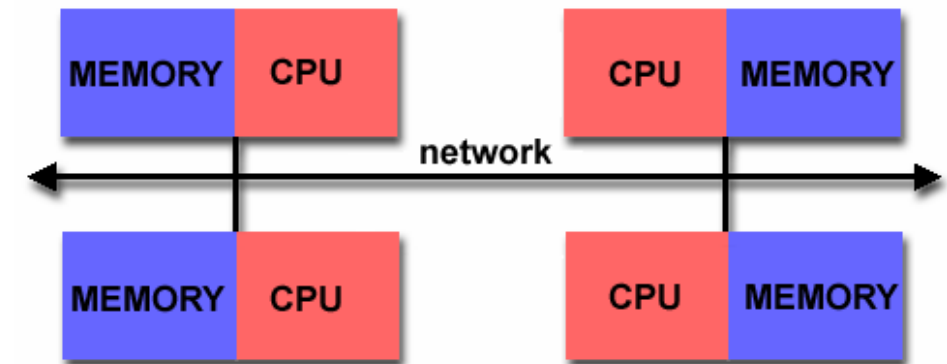
- Lack of scalability between memory and CPUs.
- Adding more CPUs can geometrically increase traffic on the shared memory-CPU
- Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
- Expense: Expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- Distributed memory systems require a communication network to connect inter-processor memory.
- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Each processor has its own local memory, it operates independently.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated.
- The network "fabric" used for data transfer varies widely

Programming Model :

- MPI (Message Passing Interface)



Advantages:

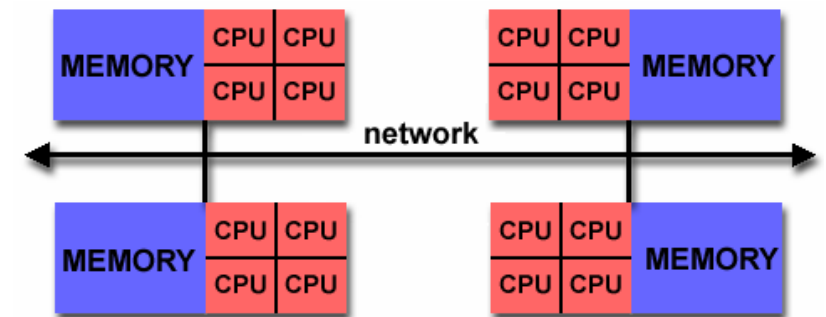
- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency (Changes it makes to its local memory have no effect on the memory of other processors).
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

Disadvantages:

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times

Hybrid Memory

- Combination of both shared and distributed memory architectures.
- The shared memory component is usually a cache coherent SMP machine. Processors on a given SMP can address that machine's memory as global.
- The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.
- Advantages and Disadvantages: whatever is common to both shared and distributed memory architectures.





CRAY XC 40 Login node architecture :

cpuinfo :

- **Processor : Sandy Bridge**
- **No of Sockets : 1**
- **No of Physical Cores : 8**
- **No of Logical Cores : 16**
- **Hyper thread : Enabled**
- **Threads per Core : 2**
- **CPU Clock Rate : 2601 MHz**
- **L1 : 32 KB**
- **L2 : 256 KB**
- **L3 : 20 MB**

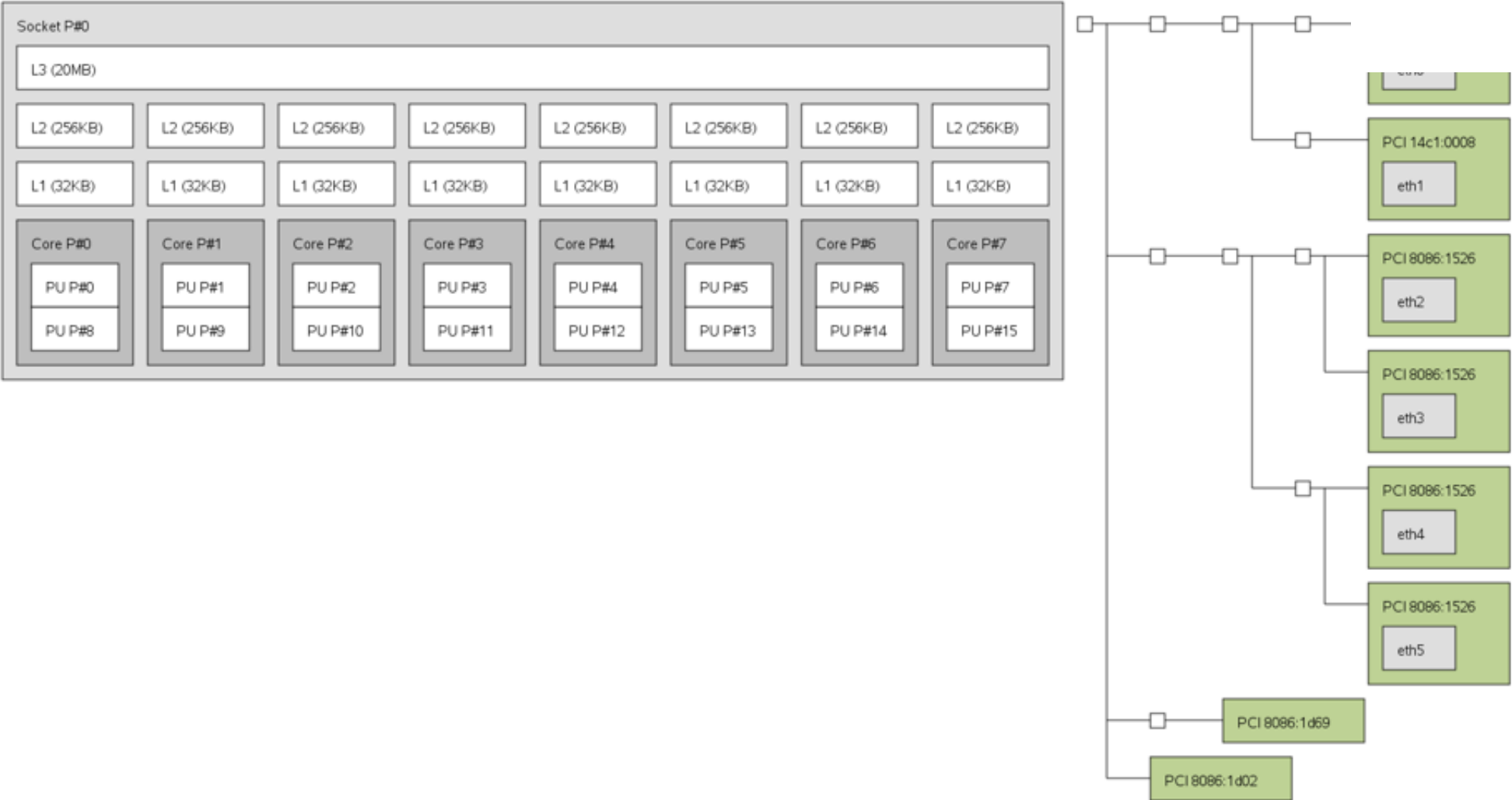
```
processor      : 15
vendor_id     : GenuineIntel
cpu family    : 6
model         : 45
model name    : Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHZ
stepping      : 7
microcode     : 1805
cpu MHz       : 2601.000
cache size    : 20480 KB
physical id   : 0
siblings      : 16
core id       : 7
cpu cores     : 8
apicid        : 15
initial apicid : 15
fpu           : yes
fpu_exception : yes
cpuid level   : 13
wp            : yes
bogomips      : 5200.11
clflush size  : 64
cache_alignment : 64
address sizes : 46 bits physical, 48 bits virtual
power management:
```



Cray XC 40 login node architecture :

lscpu info:

Machine (31GB)





CRAY XC 40 Login node CPU architecture :

cpuinfo :

- Processor : Haswell
- No of Sockets : 2
- No of cores per Socket : 12
- No of Physical Cores : 24
- No of Logical Cores : 48
- Hyper thread : Enabled
- Threads per Core : 2
- CPU Clock Rate : 2501 MHz
- L1 : 32 KB
- L2 : 256 KB
- L3 : 30 MB

```
processor      : 47
vendor_id     : GenuineIntel
cpu_family    : 6
model         : 63
model name    : Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
stepping      : 2
microcode     : 45
cpu MHz       : 2501.000
cache size    : 30720 KB
physical id   : 1
siblings      : 24
core id       : 13
cpu cores     : 12
apicid        : 59
initial apicid : 59
fpu           : yes
fpu_exception : yes
cpuid level   : 15
wp            : yes
bogomips      : 5000.68
clflush size  : 64
cache_alignment : 64
address sizes  : 46 bits physical, 48 bits virtual
power management:
```

Multicore and Multi-node programming



इवावश्वत

Multicore Programming

- **Programming directly on processor cores is painful**
- **Concurrency platforms abstract processor cores**
- **Handles synchronization, communication protocols**
- **Perform load balancing**
- **Uses shared memory architecture**

Example : Open MP

Multicore Programming – Open-MP

- **Stands for Open specifications for Multi-Processing.**
- **API to exhibit multi-threaded and shared memory parallelism.**
- **The API is specified for C/C++ and Fortran**
- **Latest Version : 4.5**

- **Three distinct components.**
 - **Compiler Directives (44)**
 - **Runtime Library Routines (35)**
 - **Environment Variables (13)**

Multicore Programing – openMP Programming Model



- OpenMP is an explicit (not automatic) programming model, offering the programmer full control over parallelization.
- OpenMP uses the fork-join model of parallel execution
- All OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered
- OpenMP programs accomplish parallelism exclusively through the use of threads.
- Threads exist within the resources of a single process. Without the process, they cease to exist.
- Typically, the number of threads match the number of machine processors/cores
- Parallelization can be as simple as taking a serial program and inserting compiler directives
- Inserting subroutines to set multiple levels of parallelism, locks and even nested locks.

Multicore Programming – Open-MP Programming Model

Example openMP :

```
#include <omp.h>
```

```
main ()
```

```
{  
  #pragma omp parallel num_threads(4)  // Compiler Directive  
    // setenv omp_num_threads 4 : Environment Variable  
  {  
    printf( "Hello! My Thread Id is : %d\n" , omp_get_num_thread ( ) );  
    // omp_get_num_thread ( ) : Runtime library routine  
  }  
}
```


Multicore Programming – Open-MP Programming Model



Example openMP :

Compilation : `cc hello.c -openmp -o hello`

Run : `./hello` (Use qusb for Sahasrat)

Output :

```
Hello! My Thread Id is : 2  
Hello! My Thread Id is : 1  
Hello! My Thread Id is : 3  
Hello! My Thread Id is : 4
```

Multi-node Programming

- Based on distributed memory architecture
- Designed for MIMD
- Handle communications between nodes
- Perform load balancing between nodes

Example : MPI

Multi-node Programming – MPI

About MPI

- **Message Passing application programmer Interface**
- **Designed to provide access to parallel hardware**
 - **Clusters**
 - **Heterogeneous networks**
 - **Parallel computers**
- **Provides for development of parallel libraries**
- **Supports C/C++ and Fortran**
- **Message passing**
 - **Point-to-point message passing operations**
 - **One to One Communication**
 - **Collective (global) operations**
 - **One to all, All to one & All to All Communications**

Multi-node Programming – MPI Example

Example MPI :

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>           // Provides basic MPI definitions and types
int main (int argc, char *argv[])
{
    int myrank;             // Process rank
    MPI_Init(&argc, &argv); // Start of MPI
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
                           // Rank of calling process in the communicator
    printf("Hello, I am process %d\n", myrank);
    MPI_Finalize();         // End of MPI Communication
    return 0;
}
```


Multi-node Programming – MPI Example

Example MPI :

Compilation :

```
cc hello_mpi.c -o hello_mpi // cc is a superior command for mpicc/mpiccc
```

Run :

```
mpirun -np 4 ./hello_mpi  
( Use aprun command to PBS script to execute on sahasrat )
```

Output :

```
Hello, I am process 2  
Hello, I am process 1  
Hello, I am process 3  
Hello, I am process 4
```

Designing Parallel Programs



इवावश्वत

Designing a Parallel solution

- **Every sequential problem may have one or more parallel solutions**
- **The best solution may differ from the actual sequential algorithm**
- **Solution is based on underlying hardware architecture, available software resources, Problem and Size of the problem**

Source : <http://www.mcs.anl.gov/~itf/dbpp/text/book.html>

Understand the Problem

- Identify the program's hotspots
- Identify bottlenecks in the program
- Identify Data dependence
- Look for alternative algorithms if possible

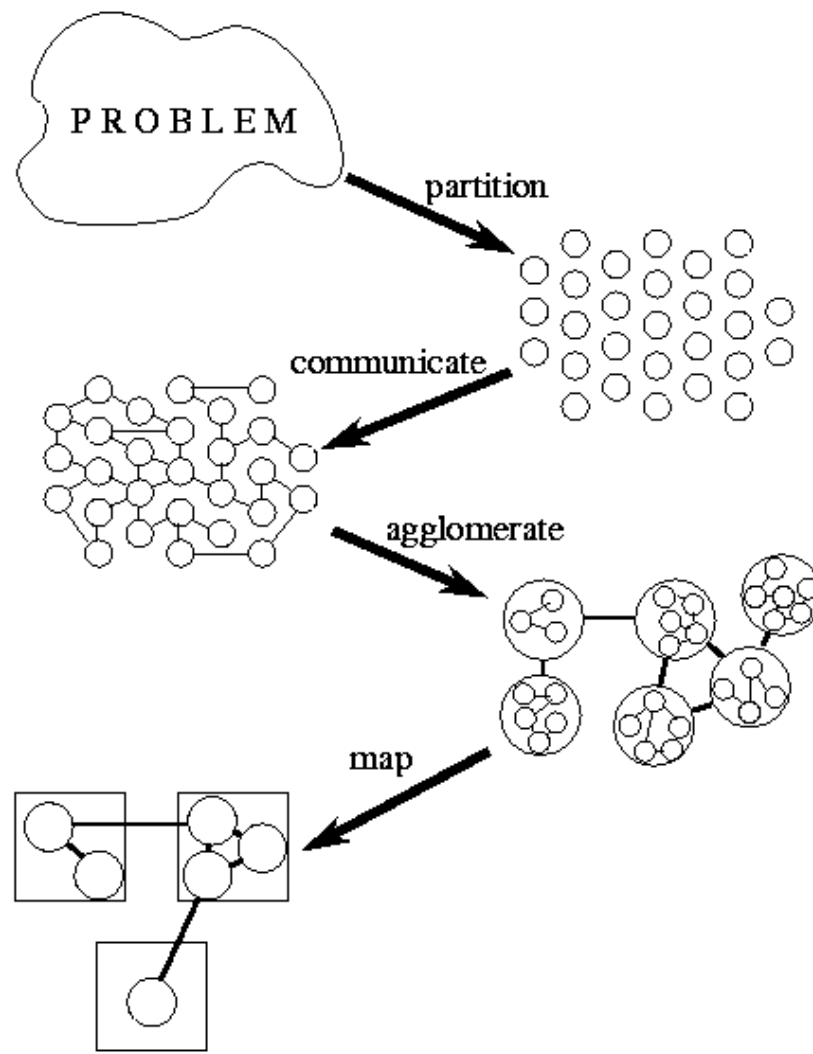
Steps to Parallelization

1. **Understand the Problem**
2. **Partition/ Decomposition** : The computation data set decomposed based on domain and functional
3. **Communication** : The communication required to coordinate task execution
4. **Agglomeration** : Based on the performance requirements and cost, If necessary, tasks are combined into larger tasks to improve performance or to reduce development costs
5. **Mapping** : Each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processor utilization and minimizing communication costs. Mapping can be specified statically or determined at runtime by load-balancing algorithms.

Source : <http://www.mcs.anl.gov/~itf/dbpp/text/book.html>
https://computing.llnl.gov/tutorials/parallel_com



Steps to Parallelization



Source : <http://www.mcs.anl.gov/~itf/dbpp/text/book.html>

Steps to Parallelizing ROMS (Regional Ocean Modeling System)

1. Understand the Problem

ROMS is a free-surface, terrain-following, primitive equations ocean model. Uses topography-following coordinates, and uses curvilinear grids. Grid geometry can be stretched/distorted, but grid is logically Cartesian.

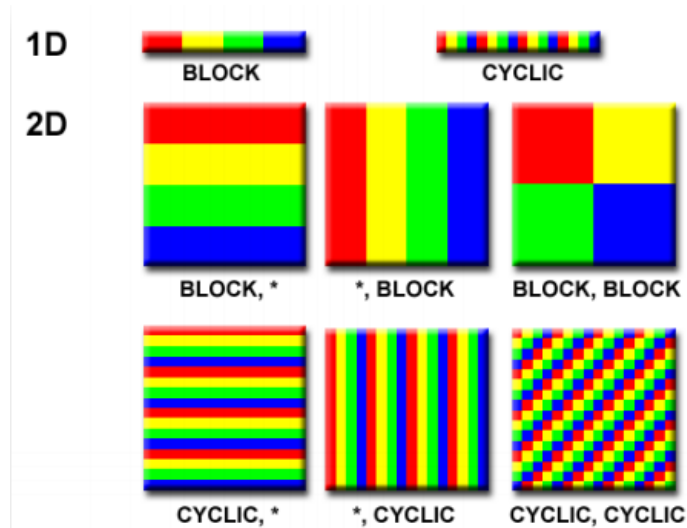
- Based on Fortran language and uses openMP or MPI
- Input file/ Workload/ benchmark size
- Grid Size (4096 x 512)
- MPI standard used

Source : https://www.myroms.org/wiki/Documentation_Portal

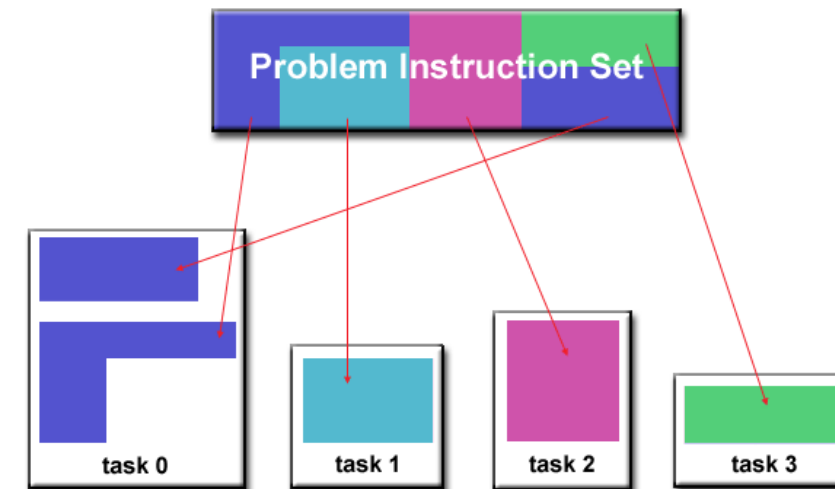
Partition/ Decomposition

- Break the problem into discrete "chunks" of work that can be distributed to multiple tasks.
- Two ways to partition computation among parallel tasks

Domain decomposition

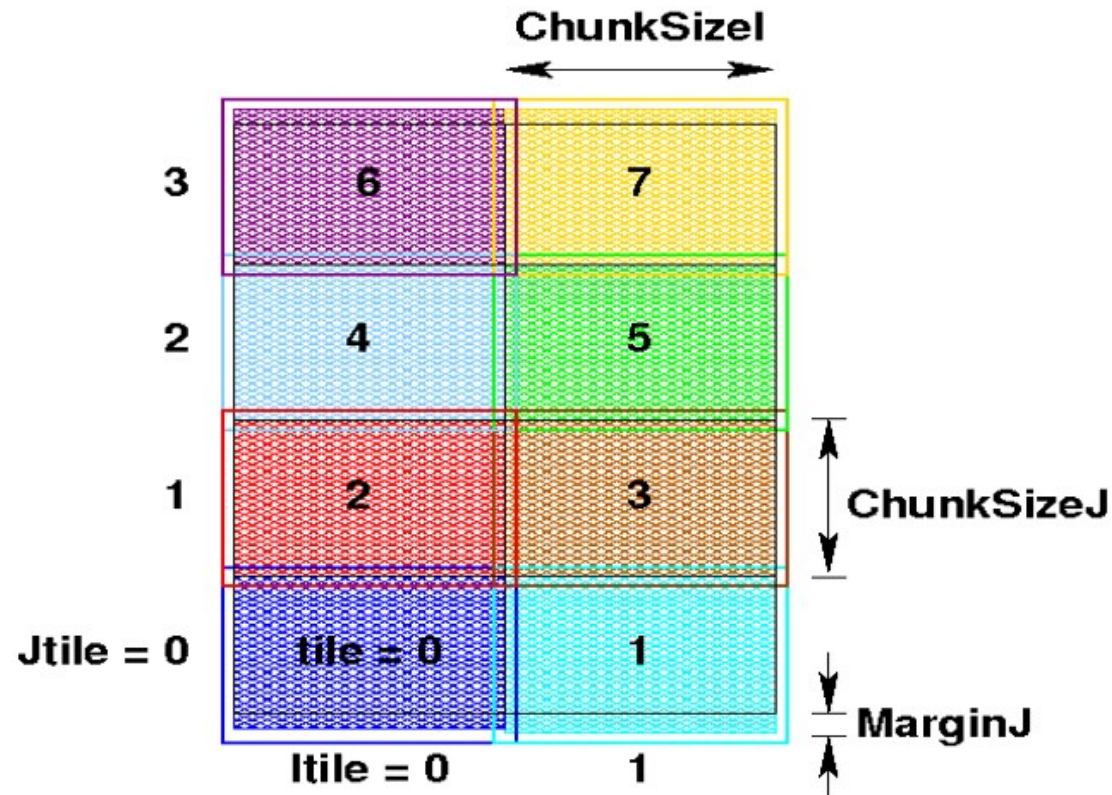


Functional decomposition



Steps to Parallelization

- **Parallel Decomposition via horizontal tiling (vertical not split)**
- **Halo regions around each tile.**
- **Data exchange happens in East-West and North-South direction.**



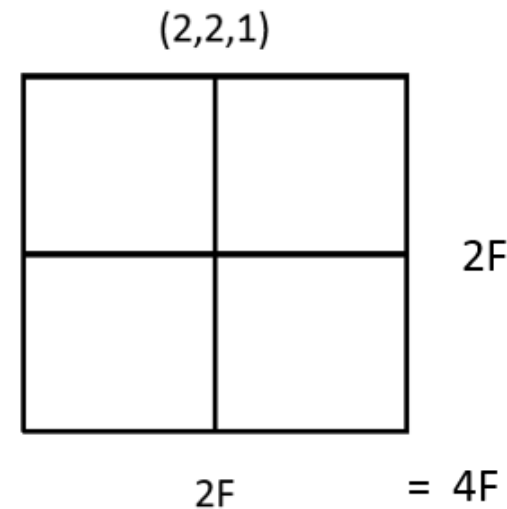
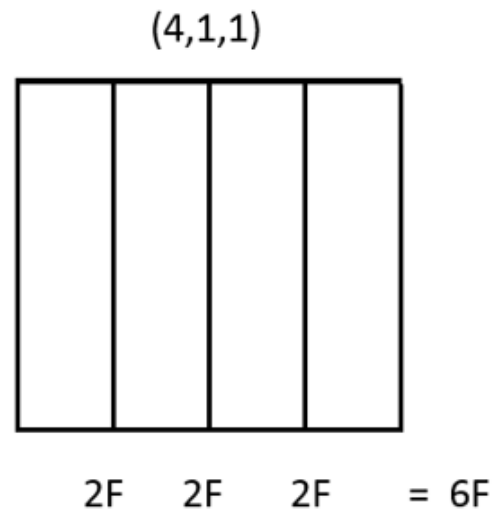
Steps to Parallelization

- **NtileI x NtileJ = 15 x 8**
- **MPI Ranks = 120**
- **Grid Size = 4096 x 512**
- **Each Rank computes of tile size 273.07 x 64**

Partition/ Decomposition

In CFD, Multi dimension decomposition (x,1,1) to (x,y,z)

- Less number of faces always a gain for computation.
- Reduces significant overhead of communication and Computation.
- A mesh with $F \times F$ size



Partition/ Decomposition

In a topology of a mesh is cuboid, of (120,8,8) then

For X,1,1 : (120,1,1)

$$\text{Faces} = 120 \times 2 \times 8 \times 8 = 15380$$

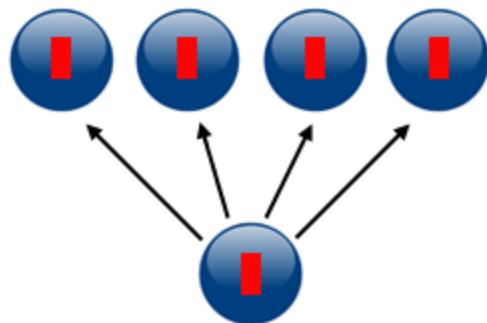
For X,Y,Z : (6,5,4)

$$\text{Faces} = 6 \times 2 \times 8 \times 8 + 5 \times 2 \times 8 \times 20 + 4 \times 2 \times 20 \times 8 = 3648$$

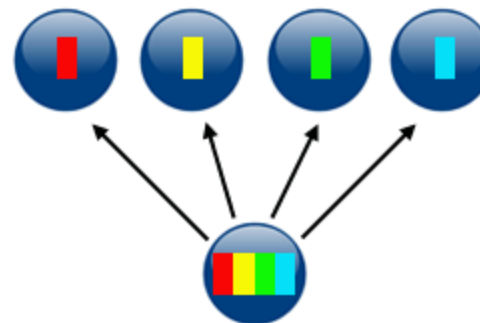
Need of Communication

- **Communications between tasks depends upon the problem**
- **Knowing which tasks must communicate with each other is critical during the design stage of a parallel code**
- **Factors to Consider**
 - **Cost of communications**
 - **Latency vs. Bandwidth**
 - **Synchronous vs. asynchronous communications**
 - **Scope of communications**

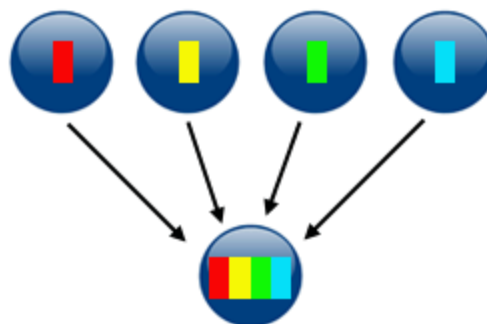
MPI Communication calls:



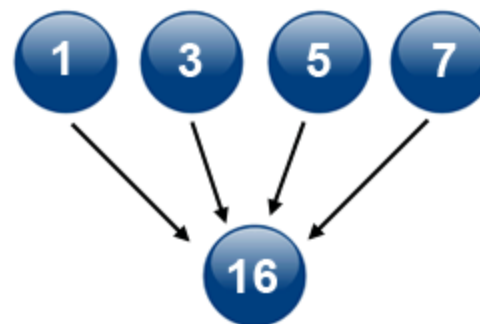
broadcast



scatter



gather

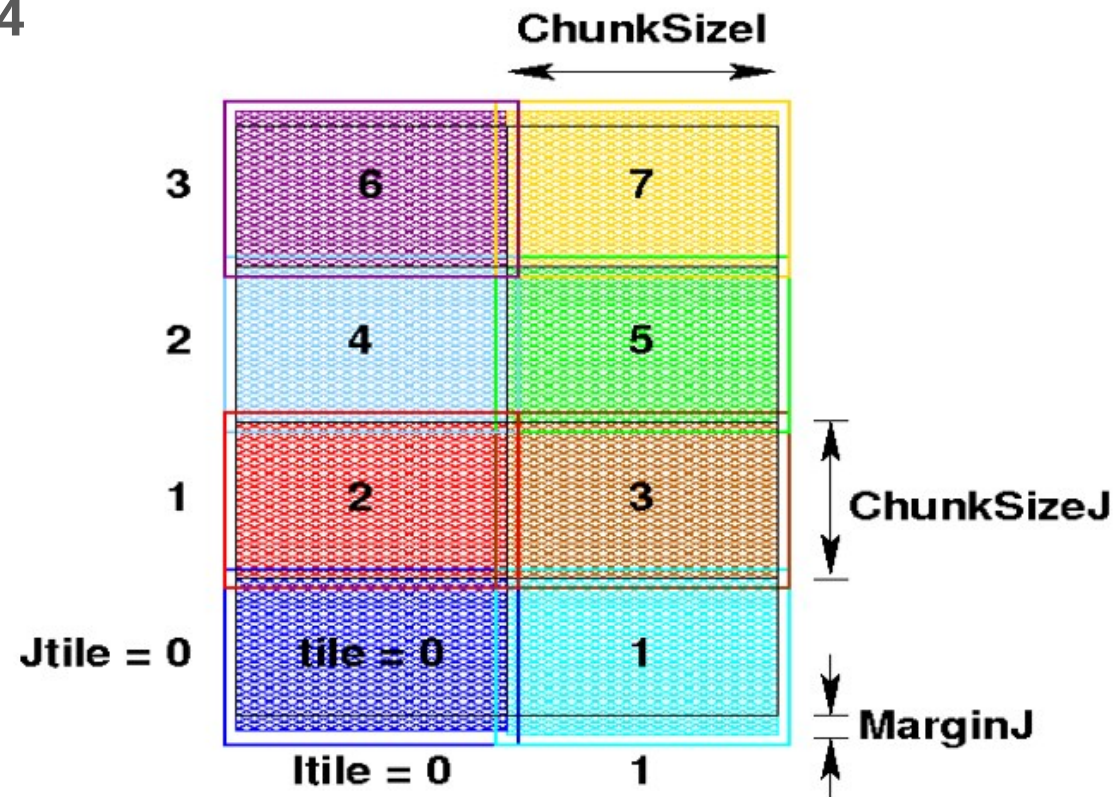


reduction



Steps to Parallelization

- Tile Size : 273.07 x 64

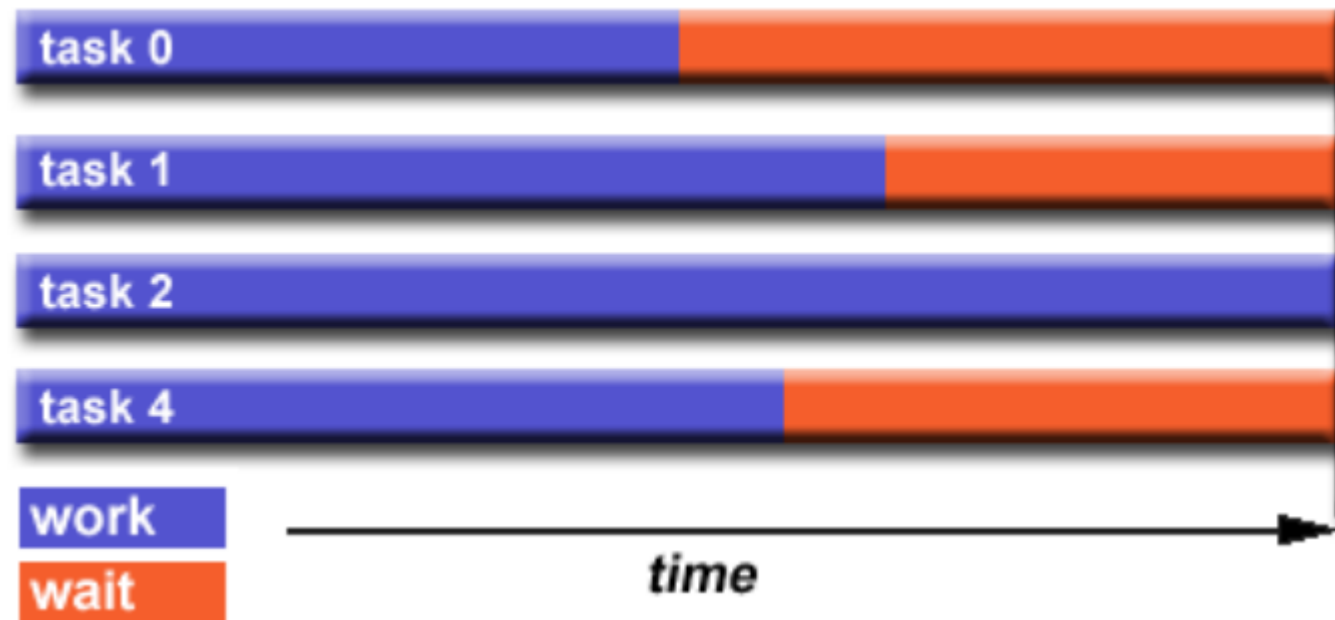




Load Balancing :

Distributing work among tasks

It can be considered a minimization of task idle time



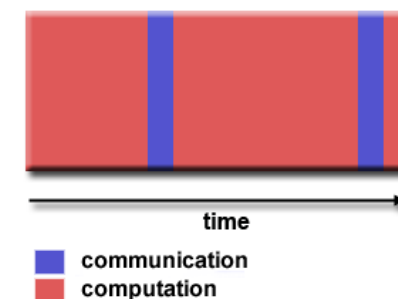


Granularity :

A qualitative measure of the ratio of computation to communication.

Coarse-grain :

- Relatively large amounts of computational work are done between communication events
- High computation to communication ratio
- Implies more opportunity for performance increase
- Harder to load balance efficiently



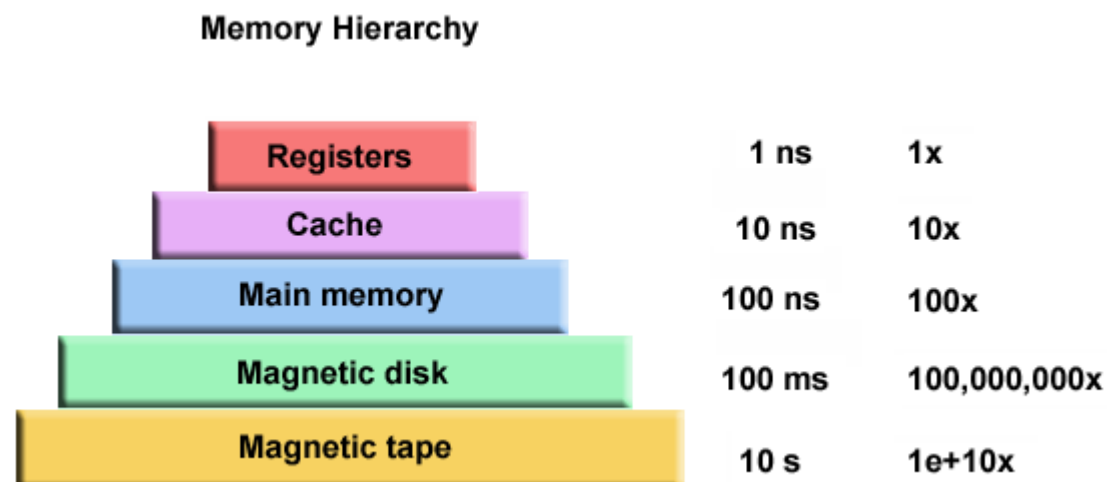
Fine-grain:

- Relatively small amounts of computational work are done between communication events
- Low computation to communication ratio
- Facilitates load balancing
- Implies high communication overhead and less opportunity for performance enhancement
- If granularity is too fine it is possible that the overhead required for communications and synchronization between tasks takes longer than the computation.



I/O :

- Reduce overall I/O as much as possible
- Writing large chunks of data rather than small chunks is usually significantly more efficient.
- Fewer, larger files performs better than many small files.



Optimization techniques



इवावइवात

Simple optimization techniques:



1. Understand Hardware architecture
2. Check bottleneck using profiling tools and debugging tools
Eg : craypat, Allinea DDT, gprof, lgdb etc

For ROMS:

Function	CPU Time	% of Run Time
lmd_skpp_tile	1861.22	11%
t3dmix2_tile	1464.29	8.50%
uv3dmix2_tile	976.551	5.60%
step2d_tile	849.443	5.00%
prsgrd_tile	739.976	4.25%
lmd_vmix_tile	692.027	4.00%
step3d_uv_tile	640.574	3.65%
diag_tile	495.767	2.85%
pre_step3d_tile	490.679	2.80%
step3d_t_tile	405.751	2.30%
rhs3d_tile	383.869	2.20%
omega_tile	256.365	1.45%

Input File	ocean_benchmark3.in
Grid Size	4096 x 512
Compiler Flags	-O -heap-arrays -r8 -ip -fp-model presice

Base line code : 145 Sec

Simple optimization techniques:



3. Use compiler related optimization flags

Eg: -O2/-O3, -parallel, -fp-model fast, -fopt-prefetch, -funroll-loops etc

For ROMS :

Input File	ocean_benchmark3.in
Grid Size	4096 x 512
Compiler Flags	-O2 -heap-arrays -r8 -fp-model precise

After removing -ip flag : 115 Sec

Input File	ocean_benchmark3.in
Grid Size	4096 x 512
Compiler Flags	-O2 -heap-arrays -r8 -fp-model fast=1

After changing -fp-model precise to -fp-model fast=1 : 100 Sec



Simple optimization techniques:

4. Use architecture specific libraries

Eg: Cray's libsci (includes BLAS, CBLAS, BLACS, LAPACK, ScaLAPACK),
Intel's MKL etc

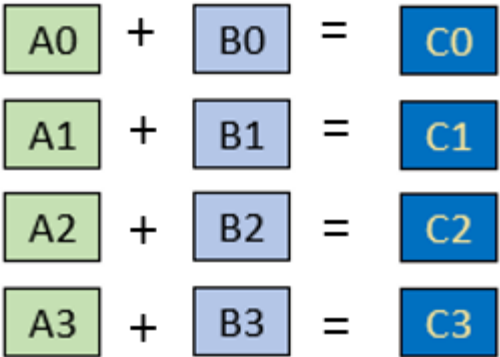


Simple optimization techniques:

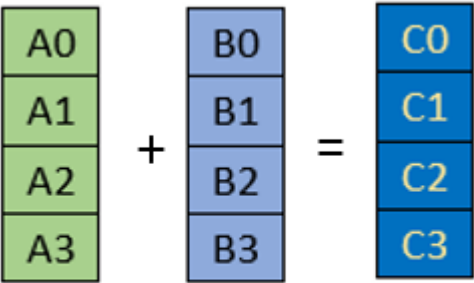
5. Use SIMD length efficiently

For Sandy Bridge : 256 bit vector width
For Haswell : 256 bit vector width

Eg : -mavx, -avx etc



Scalar Operation



SIMD Operation



Simple optimization techniques:

Eg: A typical loop:

```
DO j = JstrV-2, Jendp2
  DO i = IstrU-2, lendp2
    Drhs(i, j) = zeta( i, j, krhs ) + h(i, j)
  END DO
END DO
```

Issues :

- Array accesses not aligned on vector size boundary in memory
- Arrays may overlap - Compiler can't determine at compile time



Simple optimization techniques:

Actual Drhs allocation :

```
real(r8), dimension(IminS:ImaxS,JminS:JmaxS) :: Drhs
```

Modified allocation :

```
real(r8), allocatable, target          :: Drhs_padded(:)
!DIR$ attributes align:64              :: Drhs_padded
real(r8), pointer                      :: Drhs(:, :)

```

```
AlignCue = IstrU - 1
```

```
PadOffset = 8 - (AlignCue - IminS)
```

```
PadRows = 8 - MOD((ImaxS - IminS + 1), 8)
```

```
ArraySize = PadOffset + ((ImaxS - IminS + 1 + PadRows) * (JmaxS - JminS + 1))
```

```
allocate(Drhs_padded(ArraySize))
```

```
Drhs(IminS:(ImaxS + PadRows), JminS:JmaxS) => &
    Drhs_padded((PadOffset + 1):ArraySize)
```



Simple optimization techniques:

6. Replace expensive operations by cheaper operations

Eg :

```
a = tan(x)*(1+sin(y/2)) + tan(z)*(1-sin(y/2));
```

After optimization :

```
s = sin(y/2);  
a = tan(x)*(1+s) + tan(z)*(1-s);
```

Simple optimization techniques:

7. Use Cache efficiently

- Loop blocking/ tiling
- Interchange loops
- Loop unrolling

Simple optimization techniques:



- **Loop blocking/ tiling**

```
void mxm_ref(double *C_Ref, double *A, double *B)
{
    int i, j, k;
    memset((void*)C_Ref, 0, SIZE*SIZE*sizeof(double));
    for (i=0; i<SIZE; i++) {
        for (j=0; j<SIZE; j++) {
            for (k=0; k<SIZE; k++)
                C_Ref[i*SIZE+j] += A[i*SIZE+k] * B[k*SIZE+j];
        }
    }
}
```


Simple optimization techniques:

- **Loop blocking/ tiling**

```
void mxm_block (double *C_Opt, double *A, double *B)
{
    int i, j, k, ib, jb, kb;
    memset((void*) C_Opt, 0, SIZE*SIZE*sizeof(double));

    for (ib=0; ib<SIZE; ib+=BLOCKL2) {
        // Blocking of size BLOCKL2 ( Based on L2 cache size )
        // Loop skips BLOCKL2 Size up to Matrix Size
        for (kb=0; kb<SIZE; kb+=BLOCKL2) {
            for (jb=0; jb<SIZE; jb+=BLOCKL2) {

                for (i=ib; i< min(ib+BLOCKL2, SIZE); i++)
                {
                    // Skipped BLOCKL2 size is computed here
                    for (k=kb; k< min(kb+BLOCKL2,SIZE); k++)
                    {
                        for (j=jb; j<min(jb+BLOCKL2, SIZE); j++) {
                            C_Opt[i*SIZE+j]+=A[i*SIZE+k]*B[k*SIZE+j];
                        }
                    }
                }
            }
        }
    }
}
```

Simple optimization techniques:

- Interchange loops

```
void mxm_loop(double *C_Opt, double *A, double *B)
{
    int i,j,k;
    memset((void*)C_Opt, 0, SIZE*SIZE*sizeof(double));

    for(i=0 ; i<SIZE; i++)    {
        for(k=0; k<SIZE; k++)    {
            // Inter-changed loop ( I,J,K to I,K,J )

            for(j=0; j<SIZE ; j++)    {
                C_Opt[i*SIZE+j]+=A[i*SIZE+k]*B[k*SIZE+j];
            }
        }
    }
}
```

Simple optimization techniques:

In ROMS,

The FC Loop

- Iterating fastest in the k dimension
- Boundary condition set within the loop
- Nested if-else
- Takes **861** seconds across 120 ranks

```
DO j=Jstr, Jend
DO i=Istr, Iend
  FC(i,N(ng)) = 0.0_r8

  DO k=N(ng),1,-1
    depth = [Array Access + Flops]
    IF (Condition1) THEN
      sigma = MIN(s1_depth(i,j), depth)
    ELSE
      sigma = depth
    END IF

    zetahat = [Array Access + Flops]
    zetapar = [Array Access + Flops]

    IF (Condition2) THEN
      wm(i,j) = [Array Access + Flops]
      ws(i,j) = [Array Access + Flops]
    ELSE
      IF (Condition3) THEN
        wm(i,j) = [Array Access + Flops]
      ELSE
        wm(i,j) = [Array Access + Flops]
      END IF
      IF (Condition4) THEN
        ws(i,j) = [Array Access + Flops]
      ELSE
        ws(i,j) = [Array Access + Flops]
      END IF
    END IF

    Rk = [Array Access + Flops]
    Uk = [Array Access + Flops]
    Vk = [Array Access + Flops]
    Ribot = [Array Access + Flops]
    Ritop = [Array Access + Flops]
    FC(i,k-1) = [Array Access + Flops]
  END DO
END DO
END DO
```



Simple optimization techniques:

In ROMS,

After this Change : 91 Sec

With other misc.
optimization : 88 Sec

The Optimized FC Loop

- Iterating fastest in the i dimension
- Boundary condition is set in a new loop
- Nested if-else have been split
- Takes 153 seconds across 120 ranks
- Speedup in this loop is approximately 5.6x
- Provides around 8% gain in overall application runtime.

```
DO j=Jstr, Jend
DO k = 1, N(ng)
DO i = Istr, Iend
depth_p(i, k) =
END DO
END DO

DO k=N(ng),1,-1
DO i = Istr, Iend
sigma = depth_p
IF (Condition1 .AND. (sl_dpth(i, j) .LT. depth_p(i, k))) THEN
sigma_p(i, k) = sl_dpth(i, j)
END IF

zetahat = [Array Access + Flops]
zetapar = [Array Access + Flops]

wm(i, j) = [Array Access + Flops]
ws(i, j) = [Array Access + Flops]
IF ((.NOT. Condition2) .AND. Condition3) THEN
wm(i, j) = [Array Access + Flops]
ENDIF
IF ((.NOT. Condition2) .AND. (.NOT. Condition3)) THEN
wm(i, j) = [Array Access + Flops]
ENDIF
IF ((.NOT. Condition2) .AND. Condition4) THEN
ws(i, j) = [Array Access + Flops]
ENDIF
IF ((.NOT. Condition2) .AND. (.NOT. Condition4)) THEN
ws(i, j) = [Array Access + Flops]
ENDIF
END DO
END DO

DO k=N(ng),1,-1
DO i = Istr, Iend
Rk = [Array Access + Flops]
Uk = [Array Access + Flops]
Vk = [Array Access + Flops]
Ritop = [Array Access + Flops]
Ribot = [Array Access + Flops]
FC(i,k-1) = [Array Access + Flops]
END DO
END DO

DO i = Istr, Iend
FC(i, N(ng)) = 0.0_r8
END DO
END DO
```

Cray Scientific Libraries

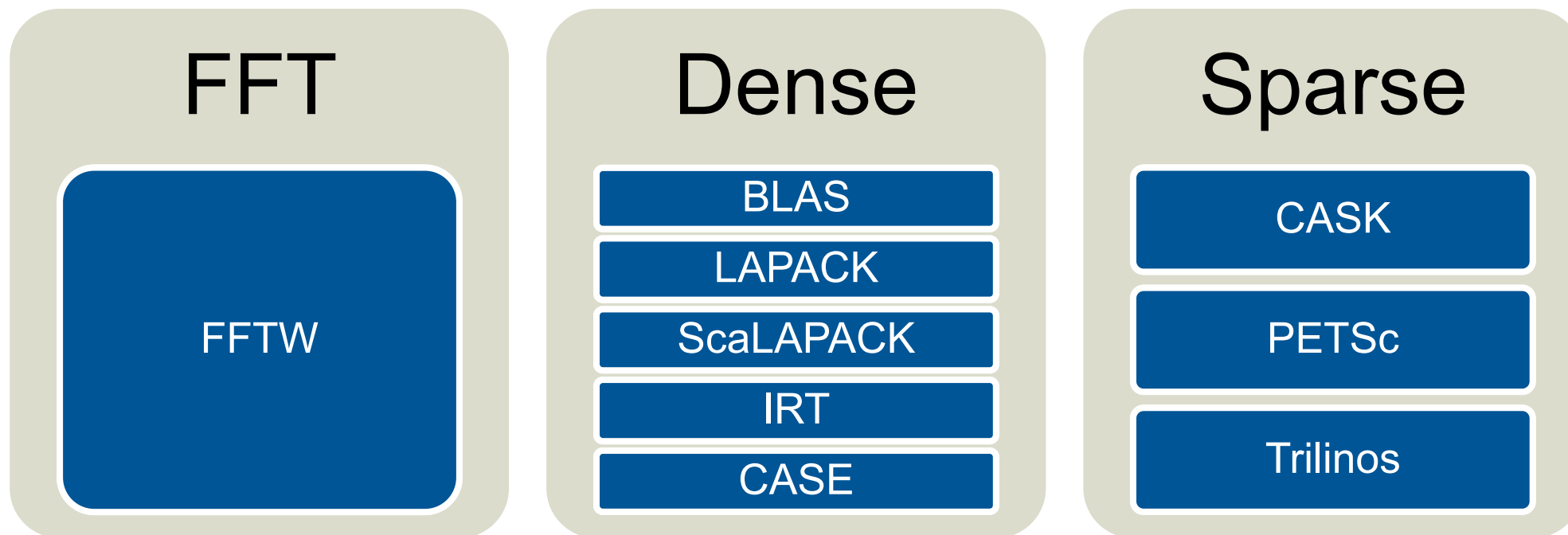


इवावइवा

Cray Scientific Libraries



- Large variety of standard libraries available via modules
 - Optimized for Cray Hardware and also for Haswell processor.



IRT – Iterative Refinement Toolkit
CASK – Cray Adaptive Sparse Kernels
CASE – Cray Adaptive Simplified Eigensolver

What makes Cray libraries special

1. Node performance

- Highly tuned routines at the low-level (ex. BLAS)

2. Network performance

- Optimized for network performance
- Overlap between communication and computation
- Use the best available low-level mechanism
- Use adaptive parallel algorithms

3. Highly adaptive software

- Use auto-tuning and adaptation to give the user the known best (or very good) codes at runtime

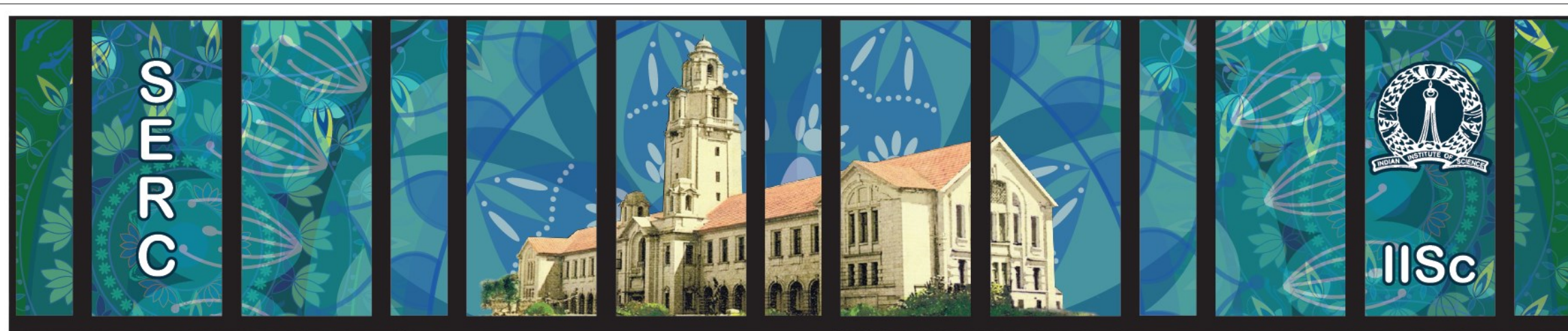
4. Productivity features

- Simple interfaces into complex software

Library Usage Overview.

- **LibSci**
 - Includes BLAS, CBLAS, BLACS, LAPACK, ScaLAPACK
 - Module is loaded by default (`man libsci`)
 - Threading used within LibSci (OMP_NUM_THREADS). If you call within a parallel region, single thread used. More info later on.
- **FFTW**
 - `module load fftw` and `man fftw`
- **PETSc**
 - `module load cray-petsc{-complex}` and `man intro_petsc`
- **Trilinos**
 - `module load cray-trilinos` and `man intro_trilinos`
- **Third Party Scientific Libraries**
 - `module load cray-tpsl` (use online documentation)
- **Iterative Refinement Toolkit (IRT) through LibSci.**
 - `man intro_irt`
- **Cray Adaptive Sparse Kernels (CASK) are used in cray-petsc and cray-trilinos (transparent to the developer).**

Performance Analysis with CrayPat

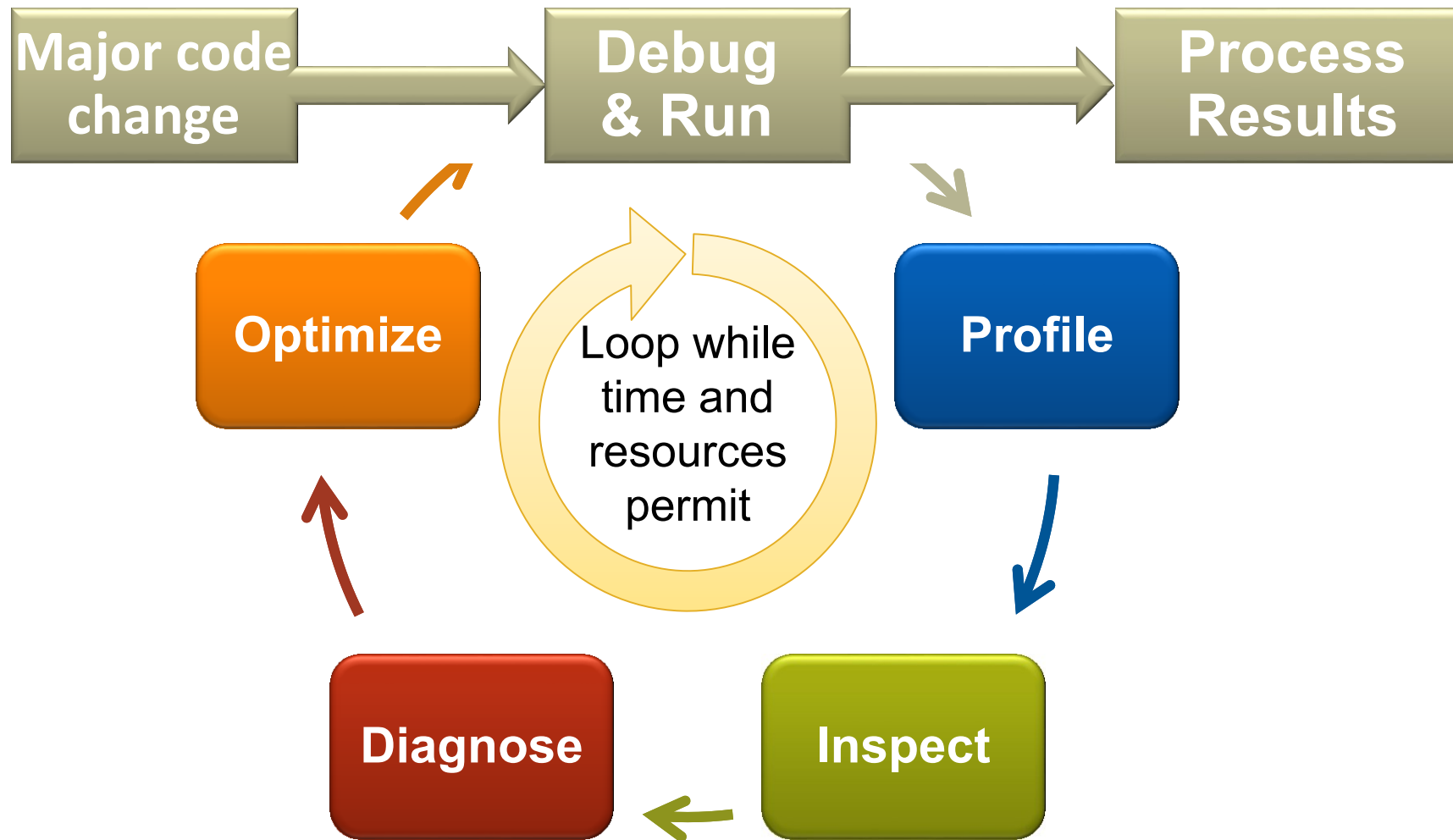


इवावश्वत

Outline

- **Introduction to performance analysis with CrayPat**
 - Different approaches to profiling: Sampling vs. Tracing
 - How to recompile and run your code for CrayPat.
 - Combining Sampling and Tracing: Automatic Performance Analysis
- **Collecting Hardware Performance counters.**

The Optimization Cycle



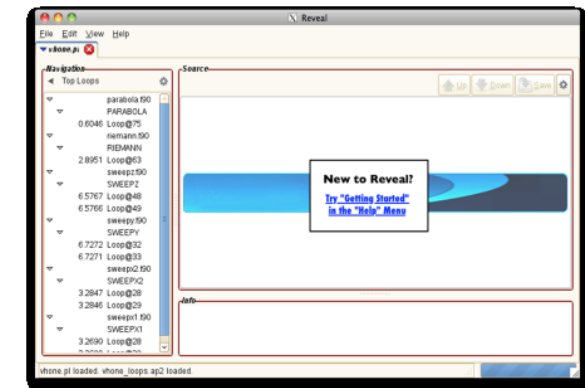
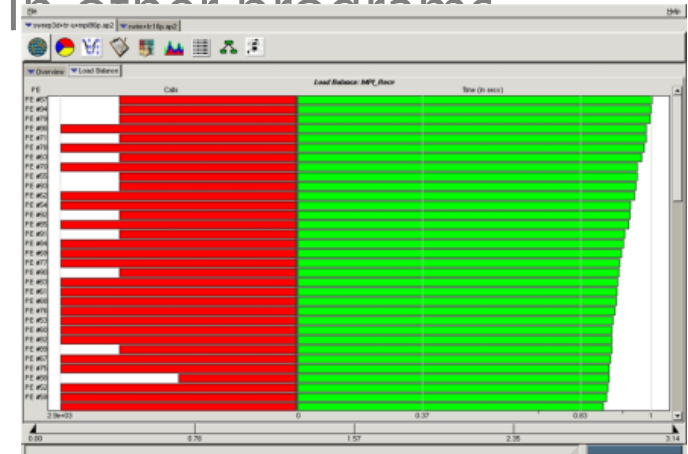
CrayPAT Overview



- **Assist the user with application performance analysis and optimization**
 - Provides concrete suggestions instead of just reporting data.
 - Work on user codes at realistic core counts with thousands of processes/threads
 - Integrate into large codes with millions of lines of code
- **Is a universal tool**
 - Basic functionality available to all compilers on the system
 - Additional functionality available for the Cray compiler (loop profiling)
- **Requires no source code or Makefile modification**
 - Automatic instrumentation at group (function) level such as mpi, io, ...
 - Requires object files and archives for instrumentation and to be compiled with the wrapper scripts while the `perftools` module was loaded.
 - Able to generate instrumentation on optimized code.
 - Creates a new stand-alone instrumented program while preserving the original binary.
- **Is under continuous development – always improving!**

Components of CrayPat

- Available through the **perftools** module:
 - **pat_build** - Instruments the program to be analyzed (command line)
 - **pat_report** - Generates text reports from the performance data captured during program execution and exports data for use in other programs (command line)
 - **Cray Apprentice2** - A graphical analysis tool that can be used to visualize and explore the performance data captured during program execution.
 - **Reveal** - A graphical source code analysis tool that can be used to correlate performance analysis data with annotated source code listings, to identify key opportunities for optimization.
 - **craypat-lite** – Light weight profiling tool.



Components of CrayPat (cont.)



- **grid_order** - Generates MPI rank order information that can be used with the `MPICH_RANK_REORDER` environment variable to override the default MPI rank placement scheme and specify a custom rank placement. (For more information, see the `intro_mpi(3)` man page.)
- **pat_help** - Help system, which contains extensive usage information and examples. This help system can be accessed by entering `pat_help` at the command line.
- The individual components of CrayPat are documented in the following man pages (info on hardware counters will follow):
 - **intro_craypat(1)**
 - **pat_build(1)**
 - **pat_report(1)**
 - **pat_help(1)**
 - **grid_order(1)**
 - **app2(1)**
 - **reveal(1)**

Sampling and Event Tracing

CrayPAT provides two fundamental ways of profiling:

1. Sampling

- By taking regular snapshots of the applications call stack we can create a statistical profile of where the application spends most time.
- Snapshots can be taken at regular intervals in time or when some other external event occurs, like a hardware counter overflowing

2. Event Tracing

- Alternatively we can record performance information every time a specific program event occurs, e.g. entering or exiting a function.
- We can get accurate information about specific areas of the code every time the event occurs
- Event tracing code can be added automatically or included manually through API calls.
- **Automatic Performance Analysis (APA) combines the two approaches.**
- **Loop profiling is a special flavor of event tracing.**

Sampling

Advantages

- Only need to instrument main routine
- Low Overhead – depends only on sampling frequency
- Smaller volumes of data produced

Disadvantages

- Only statistical averages available
- Limited information from performance counters

Event Tracing

Advantages

- More accurate and more detailed information
- Data collected from every traced function call not statistical averages

Disadvantages

- Increased overheads as number of function calls increases
- Huge volumes of data generated

The best approach is *guided tracing*.
e.g., Only tracing functions that are not small (i.e., very few lines of code) and contribute a lot to application's run time.
APA is an automated way to do this.

CrayPat - Full featured application profiling



इवावइवात

Exercise 1: Generate a Sampling Profile



```
> module load perftools
```

- Makes the default version of CrayPAT available
- Subsequent compiler invocations will automatically insert necessary hooks for profiling (not always up-to-date with latest third-party compilers)
- Binaries are *not* automatically instrumented

```
> make clean; make
```

```
> pat_build -S himeno.exe
```

- Builds code with profiling hooks, then instruments the binary
- Result named `himen.exe+pat`

```
> aprun -n 24 ./himen.exe+pat (within PBS script)
```

```
> pat_report -o myreport.txt himeno+pat+* (when PBS job returns)
```

- Running the "+pat" binary creates a data file "*.xf" or a directory in run directory
- pat_report reads that data file and prints lots of human-readable performance data. Creates an *.ap2 file.



Table 2: Profile by Group, Function, and Line

Samp%	Samp	Imb.	Imb.	Group	Function
		Samp	Samp%		Source
					Line
					PE=HIDE
100.0%	2063.0	--	--	Total	
82.3%	1698.0	--	--	USER	
77.2%	1592.2	--	--	jacobi	
3					Himeno/test.samp/himeno.c
4	61.1%	1260.6	32.4	2.9%	line.243
4	7.2%	147.8	19.2	13.2%	line.257
4	4.3%	89.5	17.5	18.7%	line.258
4	4.2%	86.5	8.5	10.2%	line.260
5.1%	105.8	--	--	initmt	
3					Himeno/test.samp/himeno.c
16.4%	338.2	--	--	ETC	
13.8%	284.8	5.2	2.1%	__cray_scopy_HSW	
2.6%	53.5	4.5	8.9%	__cray_sset_HSW	
1.3%	26.6	--	--	MPI	

Top function

Communication not relevant. Threshold of 0.5% can be cancelled with -T option.

Exercise 2: Generate a Tracing Profile



```
> module load perftools
```

- Makes the default version of CrayPAT available.

```
> pat_build -u -g mpi himeno.exe
```

- If your application is already built with perftools loaded you do not have to rebuild when switching the experiment.
- Traces MPI functions calls and functions defined in the program source files

```
> aprun -n 24 ./himen.exe+pat (from within PBS script)
```

```
> pat_report -o myrep.txt himeno+pat+*
```

- Running the "+pat" binary creates a data file or directory
- pat_report reads that data file and prints lots of human-readable performance data. Creates an *.ap2 file.



Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group	Function
						PE=HIDE
100.0%	20.643909	--	--	1149.0	Total	
98.8%	20.395989	--	--	219.0	USER	
91.1%	18.797060	0.115535	0.7%	2.0	jacobi	
7.7%	1.597866	0.006647	0.5%	1.0	initmt	
0.0%	0.000402	0.000167	33.5%	53.0	sendp3	
1.2%	0.239306	--	--	871.0	MPI	
0.7%	0.148981	0.094595	44.4%	159.0	MPI_Waitall	
0.4%	0.085824	0.023669	24.7%	318.0	MPI_Isend	
0.0%	0.004125	0.004316	58.4%	318.0	MPI_Irecv	
0.0%	0.000298	0.000013	4.8%	55.0	MPI_Allreduce	
0.0%	0.000033	0.000013	32.8%	1.0	MPI_Cart_create	
0.0%	0.008614	--	--	59.0	MPI_SYNC	
0.0%	0.006696	0.006627	99.0%	2.0	MPI_Barrier(sync)	
0.0%	0.001802	0.001399	77.6%	55.0	MPI_Allreduce(sync)	
0.0%	0.000061	0.000052	86.3%	1.0	MPI_Init(sync)	
0.0%	0.000056	0.000051	91.7%	1.0	MPI_Finalize(sync)	

User functions

Communication

Synchronisation

Options for Tracing

- **More information is given in the `pat_build` man page**
 - **-u** Create new trace intercept routines for those functions that are defined in the respective source file owned by the user.
 - **-w** Make tracing the default experiment and create new trace intercept routines for those functions for which no trace intercept routine already exists. If `-t`, `-T`, or the trace build directive are not specified, only those functions necessary to support the CrayPat runtime library are traced.
 - **-T tracefunc** Instrument program to trace the function references to tracefunc. This option applies to all user-defined entry points as well as to those that appear in the predefined function groups listed under the `-g` option. Use the `nm` or `readelf` command to determine function names to specify for tracing. If tracefunc begins with an exclamation point (!) character, references to tracefunc are not traced.
 - **-t tracefile** Instrument program to trace all function references listed in tracefile.
- **Only true function calls can be traced. Functions that are inlined by the compiler or that have local scope in a compilation unit cannot be traced.**

Options for Tracing



- More information is given in the `pat_build` man page
 - **-g tracegroup** Instrument the program to trace all function references belonging to the trace function group `tracegroup`. Only those functions actually executed by the program at runtime are traced. A selection of `tracegroup` values is:
 - **blas** Basic Linear Algebra subprograms
 - **netcdf** Network Common Data Form
 - **hdf5** HDF5 I/O library
 - **heap** dynamic heap
 - **io** includes `stdio` and `sysio` groups
 - **lapack** Linear Algebra Package
 - **mpi** MPI
 - **omp** OpenMP API
 - **sysio** I/O system calls
 - **syscall** system calls
- More information on the various `tracegroup` values is given in `$CRAYPAT_ROOT/share/traces` after loading the `perftools` module.

Files generated during regular Profiling

- **a.out+pat+PID-node[s|t].xf: raw data files**
 - Depending on the nature of the program and the environmental conditions in effect at the time of program execution, when executed, the instrumented executable generates one or more data files with the suffix .xf, where:
 - **a.out** is the name of the original program.
 - **PID** is the process ID assigned to the instrumented program at runtime.
 - **node** is the physical node ID upon which the rank zero process executed.
 - **s|t** is a one-letter code indicating the type of experiment performed, either **s** for sampling or **t** for tracing.
 - Use the `pat_report` command to view or dump the .xf file or export it to another file format for use with other applications, i.e. *.ap2 files.
- ***.ap2 files: self contained compressed performance files.**
 - Normally about 5 times smaller than the corresponding set of *.xf files.
 - Only one *.ap2 per experiment compared to potentially multiple *.xf files.
 - Contains the information needed from the application binary and can be reused, even if the application binary is no longer available or if it was rebuilt.
 - Is independent on the version used to generate the ap2 file while the xf files are very version dependent.
 - It is the only input format accepted by Cray Apprentice2 and Reveal.
 - => Can delete the .xf files after you have the ap2 file.

Using pat_report

- **Always need to run `pat_report` at least once to perform data conversion**
 - Combines information from xf output (optimized for writing to disk) and binary with raw performance data to produce ap2 file (optimized for visualization analysis and smaller than raw data)
 - **Instrumented binary must still exist when data is converted!**
 - Resulting ap2 file is the input for subsequent pat_report calls and Reveal or Apprentice²
 - xf files and instrumented binary files can be removed once ap2 file is generated.
- **Generates a text report of performance results**
 - Data laid out in tables
 - Many options for sorting, slicing or dicing data in the tables.
 - > `pat_report -O <table option> *.ap2`
 - > `pat_report -O help (list of available profiles)`
 - Volume and type of information depends upon sampling vs tracing.

Some useful predefined report types:

- **pat_report -O ca+src**
 - Show the callers (bottom-up view) leading to the routines that have a high use in the report and include source code line numbers for the calls and time-consuming statements.
- **pat_report -O load_balance**
 - Show load-balance statistics for the high-use routines in the program. Parallel processes with minimum, maximum and median times for routines will be displayed. Only available with tracing experiments.
- **pat_report -O mpi_callers**
 - Show MPI message statistics. Only available with tracing experiments.

CrayPat-lite



- Light-weight application profiling
- Good place to start!

CrayPat-lite Overview

- Provide automatic application performance statistics at the end of a job. Focus is to offer a simplified interface to basic application performance information for users not familiar with the Cray performance tools and perhaps new to application performance analysis.
- The tool is enabled by loading a module and rebuild

```
> module load perftools-lite  
> make clean && make
```

- Program is automatically relinked to add instrumentation in a.out (**pat_build** step done for the user)
 - .o files are automatically preserved
 - No modifications are needed to a batch script to run instrumented binary, since original binary is replaced with instrumented version
 - **pat_report** is automatically run before job exits.
 - Performance statistics are issued to stdout
 - User can use "classic" CrayPat for more in-depth performance investigation

Steps to Using CrayPat-lite



Access light version of performance tools software

```
> module load perftools-lite
```

Build program

```
> make
```



```
a.out (instrumented program)
```

Run program (no modification to batch script)

```
aprun a.out
```



```
Condensed report to stdout  
a.out*.rpt (same as stdout)  
a.out*.ap2  
MPICH_RANK_XXX files
```

Performance Statistics Available



Job information

- Number of MPI ranks, ...
- Wallclock
- Memory high water mark
- Performance counters (CPU only)

Number of PEs (MPI ranks): 64
Numbers of PEs per Node: 32 PEs on each of 2 Nodes
Numbers of Threads per PE: 1
Number of Cores per Socket: 16
Execution start time: Fri Feb 15 14:42:24 2013

Wall Clock Time: 122.600994 secs
High Memory: 45.70 MBytes

Profile of top time consuming routines with load balance

Sampl%	Sampl	Imb.	Imb.	Group
		Sampl	Sampl%	Function
				PE=HIDE
100.0%	14272.5	-	-	Total
46.0%	6561.4	-	-	USER
5.9%	847.6	155.4	15.7%	jcollocate_core_1_
4.9%	700.3	125.7	15.5%	jintegrate_core_2_
3.8%	544.0	124.0	18.9%	jcollocate_core_2_
3.7%	523.1	73.9	12.6%	jintegrate_core_1_
29.7%	4239.6	-	-	MPI
9.3%	1328.3	198.7	13.2%	jmpi_alltoallv
4.2%	588.5	71.5	10.8%	jmpi_waitall
2.9%	413.8	107.2	20.9%	jmpi_WAITANY
2.9%	408.1	66.9	14.3%	jmpi_Comm_create

Time%	Time	Imb.	Imb.	Calls	Group
		Time	Time%		Function
					PE=HIDE
100.0%	101.961423	-	-	5315211.9	Total
92.5%	94.267451	-	-	5272245.9	USER
75.8%	77.248585	2.356249	3.0%	1001.0	LAMMPS_NS::PairLJCut::compute
6.5%	6.644545	0.105246	1.6%	51.0	LAMMPS_NS::Neighbor::half_bin_newton
4.1%	4.131842	0.634032	13.5%	1.0	LAMMPS_NS::Verlet::run
3.8%	3.841349	1.241434	24.8%	5262868.9	LAMMPS_NS::Pair::ev_tally
1.3%	1.288463	0.181268	12.5%	1000.0	LAMMPS_NS::FixNVE::final_integrate
7.0%	7.110931	-	-	42637.0	MPI
4.8%	4.851309	3.371093	41.6%	12267.0	MPI_Send
1.5%	1.536106	2.592504	63.8%	12267.0	MPI_Wait

Observations and Instructions on how to get more info.

Cray Apprentice2



इवावइवा

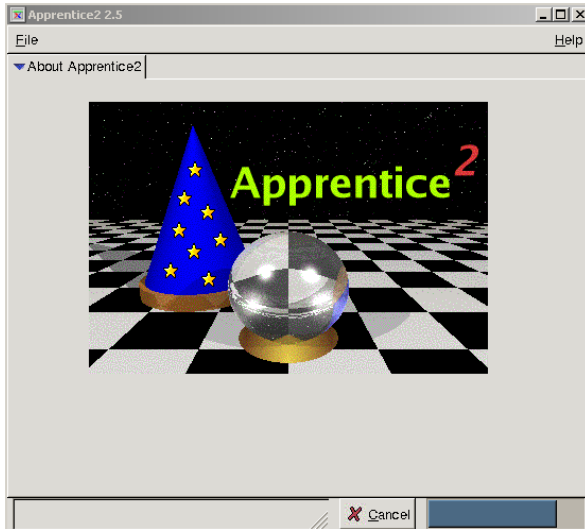
Installing Apprentice2 on Laptop

From a Cray login node

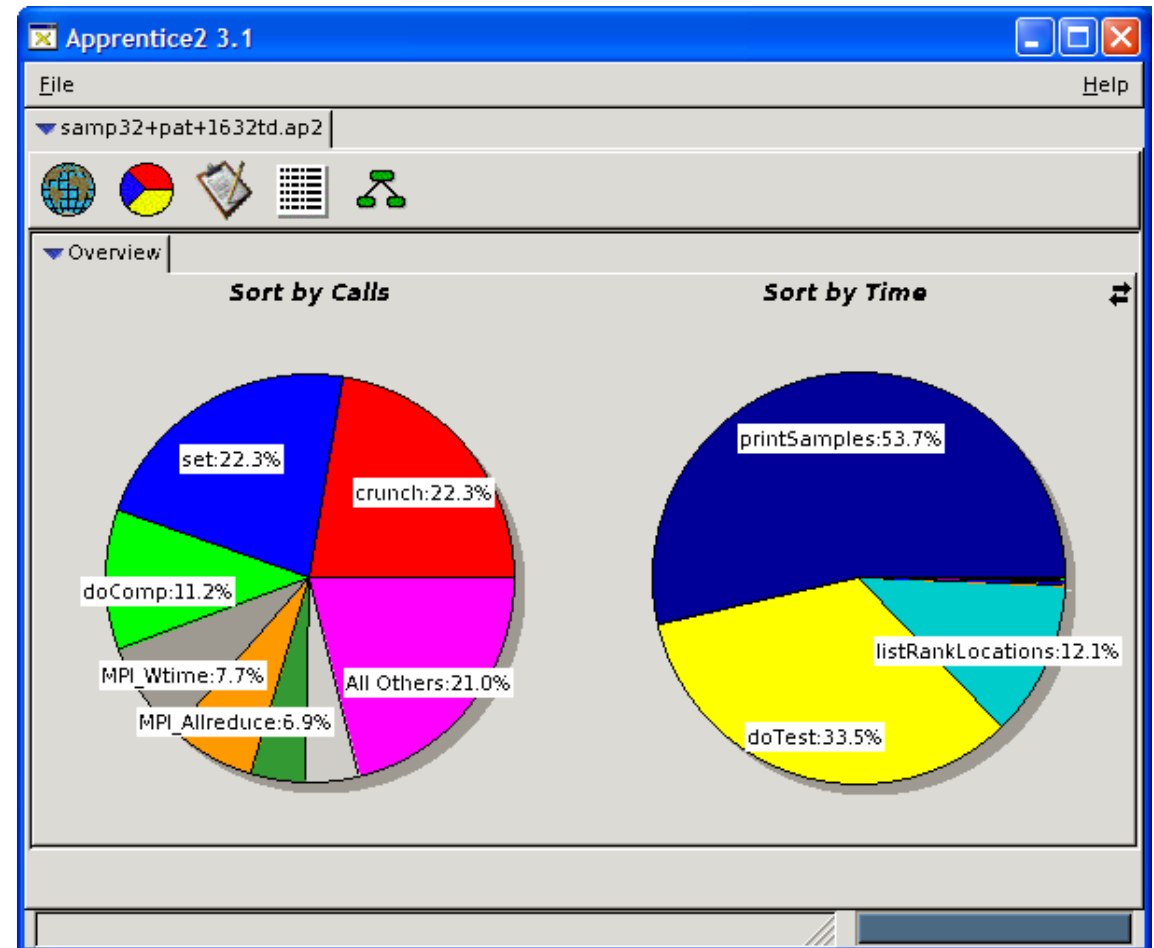
- `> module load perftools`
- **Go to:**
 - `$CRAYPAT_ROOT/share/desktop_installers/`
- **Download .dmg or .exe installer to laptop**
- **Double click on installer and follow directions to install**
- **Of course, can just run app2 from the login prompt instead**

Cray Apprentice2

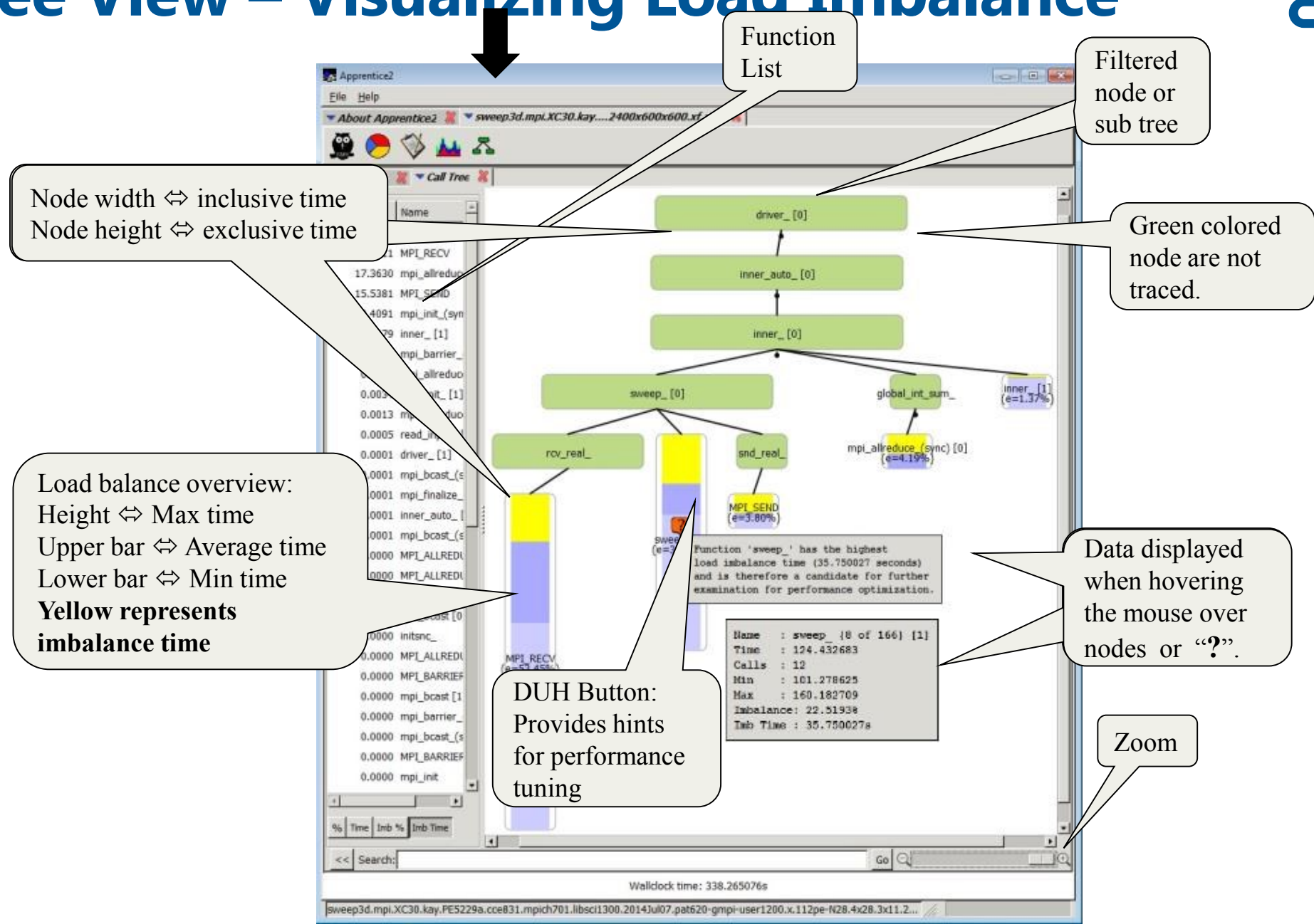
```
% module load perftools  
% app2 program1+pat+180tdo-0000.ap2
```



Many options for viewing Results. See “man app2” or Cray documentation



Call Tree View – Visualizing Load Imbalance

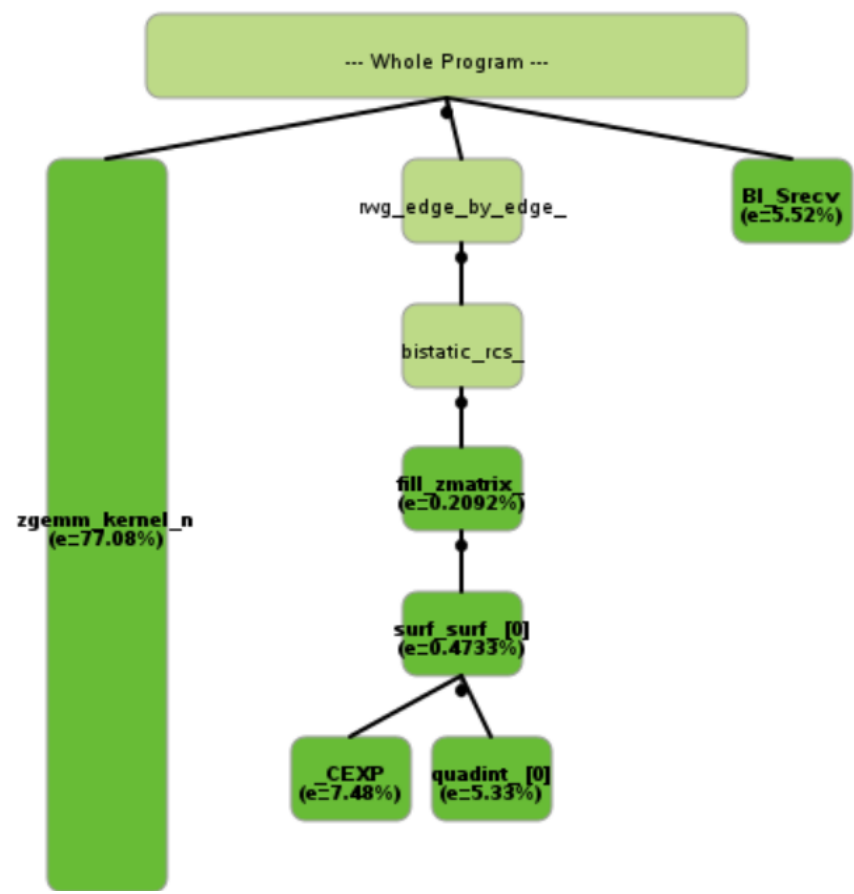


Example – Craypat + Apprentice2



इवावश्वत

Call-Tree – Users Baseline



TLB utilization

Here, 512 x 8-byte double precision floats Therefore any usage of page less than 10X shows poor use. In the 4817 doesn't looked good use considering lower user of the page present in the buffer

D1 cache utilization

Level 1 cache line is 64 contiguous bytes, e.g. 8 x 8-byte doubles
So if every double was used once, expect 8 refs/miss
It Corresponds to hit ratio of 87.5%
95.4 is excellent utilization

D1+D2 cache hit ratio

Should be high (rule of thumb is more than 97%)
99.3% is pretty good ration

D2 cache utilization

Should be put to scrutiny, for performance optimization

TLB utilization

D1 cache hit,miss ratios
D1 cache utilization (misses)
D2 cache hit,miss ratio
D1+D2 cache hit,miss ratio
D1+D2 cache utilization
D2 to D1 bandwidth

4,817.62 refs/miss

95.3% hits

21.30 refs/miss

50.8% hits

97.7% hits

43.29 refs/miss

7,486.647MiB/sec 25,551,258,703,839 bytes

9.41 avg uses

4.7% misses

2.66 avg hits

49.2% misses

2.3% misses

5.41 avg hits

Order of the matrix 77K
Time to Solution 3,259.549389 (3259 sec)

Offered Block Size vs Cache+Performance



16X16-Mat_Blkl_Size - 4,231.546053sec

TLB utilization	5,871.60 refs/miss	11.47 avg uses
D1 cache hit,miss ratios	94.5% hits	5.5% misses
D1 cache utilization (misses)	18.32 refs/miss	2.29 avg hits
D2 cache hit,miss ratio	75.0% hits	25.0% misses
D1+D2 cache hit,miss ratio	98.6% hits	1.4% misses
D1+D2 cache utilization	73.41 refs/miss	9.18 avg hits
D2 to D1 bandwidth	5,094.990MiB/sec	22,588,110,969,719 bytes

Lower TLB Utilization correspond to lower L2 cache hits

D1-D2 cache hit ration is directly proportional to the performance

32X32-Mat_Blkl_Size - 4,234.151549sec

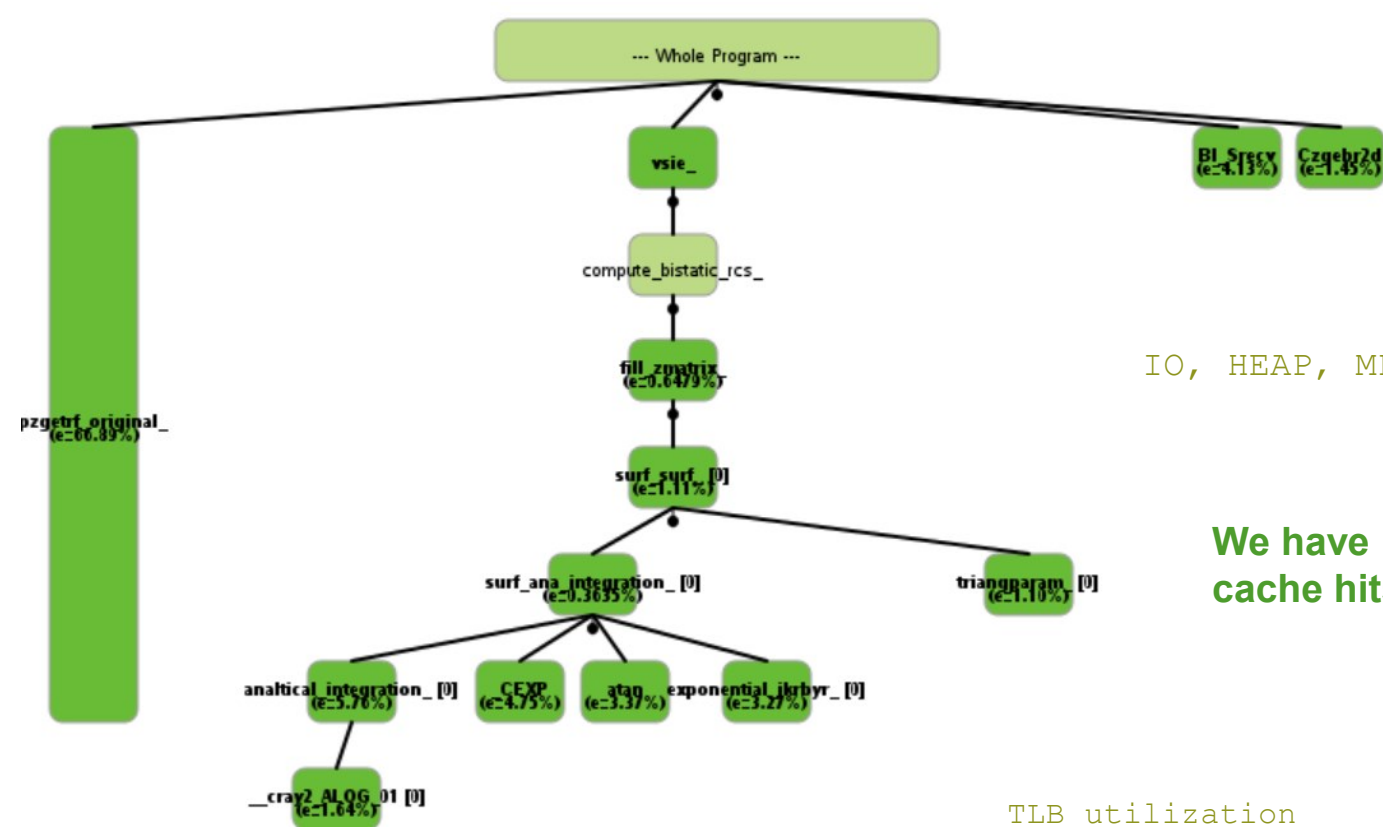
TLB utilization	5,901.81 refs/miss	11.53 avg uses
D1 cache hit,miss ratios	94.5% hits	5.5% misses
D1 cache utilization (misses)	18.25 refs/miss	2.28 avg hits
D2 cache hit,miss ratio	75.1% hits	24.9% misses
D1+D2 cache hit,miss ratio	98.6% hits	1.4% misses
D1+D2 cache utilization	73.18 refs/miss	9.15 avg hits
D2 to D1 bandwidth	5,093.079MiB/sec	22,591,206,456,691 bytes

D2 Cache Miss Ratio- 10% lower ration cause 19% performance hit in the case of 16 and 32 block size matrix

64X64-Mat_Blkl_Size - 3,557.615978sec

TLB utilization	7,120.13 refs/miss	13.91 avg uses
D1 cache hit,miss ratios	95.3% hits	4.7% misses
D1 cache utilization (misses)	21.41 refs/miss	2.68 avg hits
D2 cache hit,miss ratio	85.0% hits	15.0% misses
D1+D2 cache hit,miss ratio	99.3% hits	0.7% misses
D1+D2 cache utilization	142.64 refs/miss	17.83 avg hits
D2 to D1 bandwidth	4,995.377MiB/sec	18,452,159,972,629 bytes

Call-Tree – New Baseline



~30%
more elements / time

IO, HEAP, MPI, PTHREAD, OMP, BLAS, BLACS, SCALAPACK, PBLAS

We have Improved on both the aspects of nonperformance, D2
cache hits and Overall TLB utilization

Order of the matrix 110K
Time to Solution 3,557.615978 (3557 sec)

TLB utilization	7,120.13 refs/miss	13.91 avg uses
D1 cache hit,miss ratios	95.3% hits	4.7% misses
D1 cache utilization (misses)	21.41 refs/miss	2.68 avg hits
D2 cache hit,miss ratio	85.0% hits	15.0% misses
D1+D2 cache hit,miss ratio	99.3% hits	0.7% misses
D1+D2 cache utilization	142.64 refs/miss	17.83 avg hits
D2 to D1 bandwidth	4,995.377MiB/sec	18,452,159,972,629 bytes

Thank You



इवावश्रवत

Contact : pankaj.navani@cray.com & raviteja@cray.com