Introduction to Parallel Programming

Martin Čuma
Center for High Performance Computing
University of Utah
m.cuma@utah.edu

Overview

- Types of parallel computers.
- Parallel programming options.
- How to write parallel applications.
- How to compile.
- How to debug/profile.
- Summary, future expansion.

Parallel architectures

Single processor:

- SISD – single instruction single data.

Multiple processors:

- SIMD - single instruction multiple data.
- MIMD – multiple instruction multiple data.
  - Shared Memory
  - Distributed Memory
- Current processors combine SIMD and MIMD
  - Multi-core CPUs w/ SIMD instructions (AVX, SSE)
  - GPUs with many cores and SIMT

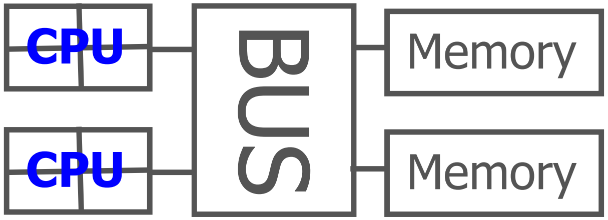
Shared memory

- All processors have access to local memory
- Simpler programming
- Concurrent memory access
- More specialized hardware
- CHPC :

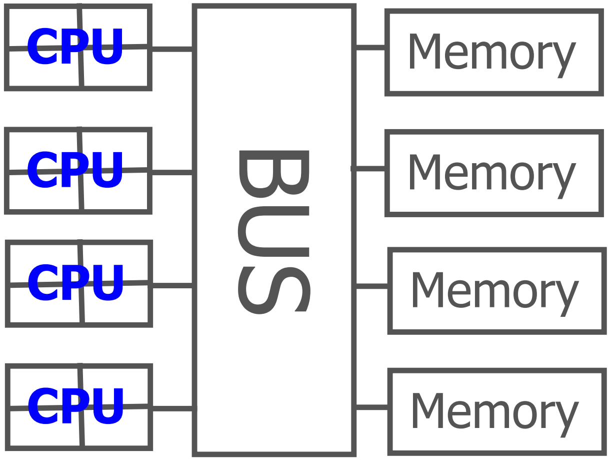
  Linux clusters 12, 16, 20, 24 core nodes

  GPU nodes

Dual quad-core node

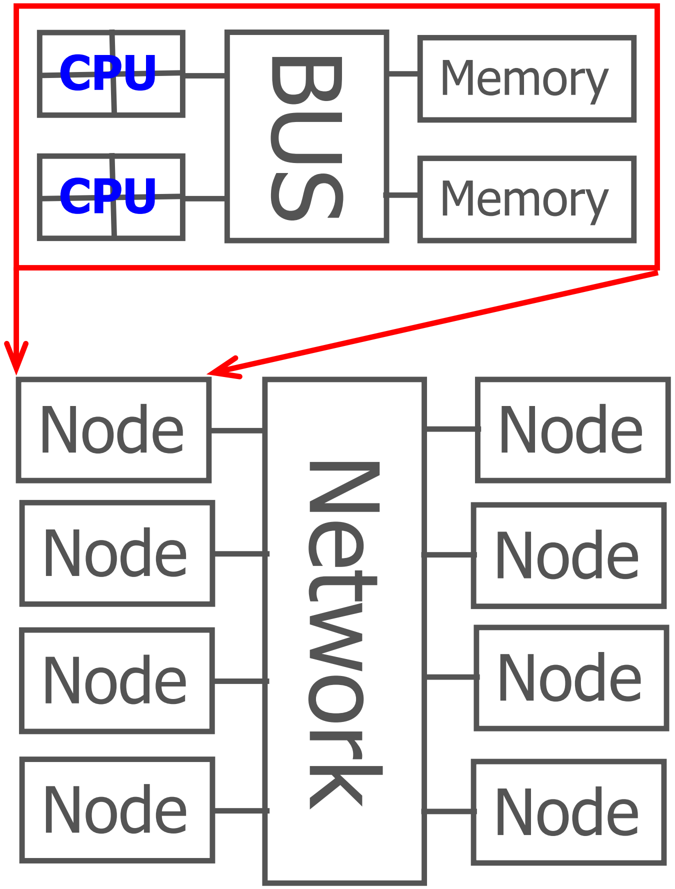


Many-core node (e.g. SGI)



Distributed memory

- Process has access only to its local memory
- Data between processes must be communicated
- More complex programming
- Cheap commodity hardware
- CHPC: Linux clusters



8 node cluster (64 cores)

Parallel programming options

Shared Memory

- Threads – POSIX Pthreads, OpenMP (CPU, MIC), OpenACC, CUDA (GPU)
  - Thread – own execution sequence but shares memory space with the original process
- Message passing – processes
  - Process – entity that executes a program – has its own memory space, execution sequence

Distributed Memory

- Message passing libraries
- Vendor specific – non portable
- General – MPI, PVM, language extensions (Co-array Fortran, UPC. …)

OpenMP basics

- Compiler directives to parallelize
  - Fortran – source code comments

    `!$omp parallel/!$omp end parallel`
  - C/C++ - #pragmas

    `#pragma omp parallel`
- Small set of subroutines
- Degree of parallelism specification
  - `OMP_NUM_THREADS` or `omp_set_num_threads(INTEGER n)`

MPI Basics

- Communication library
- Language bindings:
  - C/C++ - `int MPI_Init(int argv, char* argc[])`
  - Fortran - `MPI_Init(INTEGER ierr)`
- Quite complex (100+ subroutines)

  but only small number used frequently
- User defined parallel distribution

MPI vs. OpenMP

- Complex to code
- Slow data communication
- Ported to many architectures
- Many tune-up options for parallel execution

- Easy to code
- Fast data exchange
- Memory access (thread safety)
- Limited usability
- Limited user’s influence on parallel execution

Program example

- saxpy – vector addition: $\bar{z} = a\bar{x} + \bar{y}$
- simple loop, no cross-dependence, easy to parallelize

```
subroutine saxpy_serial(z, a, x, y, n)
integer i, n
real z(n), a, x(n), y(n)

do i=1, n
  z(i) = a*x(i) + y(i)
enddo
return
```

OpenMP program example

```
subroutine saxpy_parallel_omp(z, a, x, y, n)
integer i, n
real z(n), a, x(n), y(n)

!$omp parallel do
do i=1, n
   z(i) = a*x(i) + y(i)
enddo
return
```

```
setenv OMP_NUM_THREADS 16
```

MPI program example

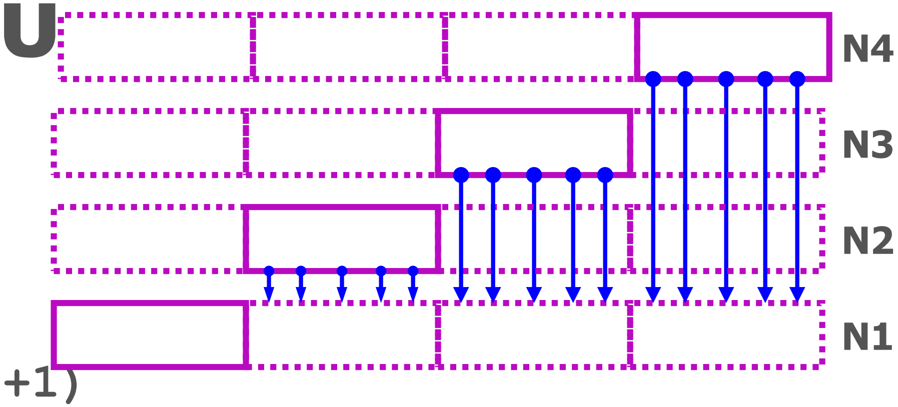
```
subroutine saxpy_parallel_mpi(z, a, x, y, n)
integer i, n, ierr, my_rank, nodes, i_st, i_end
real z(n), a, x(n), y(n)

call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,my_rank,ierr)
call MPI_Comm_size(MPI_COMM_WORLD,nodes,ierr)
i_st = n/nodes*my_rank+1
i_end = n/nodes*(my_rank+1)

do i=i_st, i_end
   z(i) = a*x(i) + y(i)
enddo
call MPI_Finalize(ierr)
return
```

z(i) operation on 4 processes

z(1 … n/4)	z(n/4+1 … 2*n/4)	z(2*n/4+1 … 3*n/4)	z(3*n/4+1 … n)

MPI program example

- **Result on the first CPU**



```
include "mpif.h"
integer status(MPI_STATUS_SIZE)
if (my_rank .eq. 0 ) then
  do j = 1, nodes-1
    do i= n/nodes*j+1, n/nodes*(j+1)
      call MPI_Recv(z(i),1,MPI_REAL,j,0,MPI_COMM_WORLD,
&     status,ierr)
    enddo
  enddo
else
  do i=i_st, i_end
     call MPI_Send(z(i),1,MPI_REAL,0,0,MPI_COMM_WORLD,ierr)
  enddo
endif
```

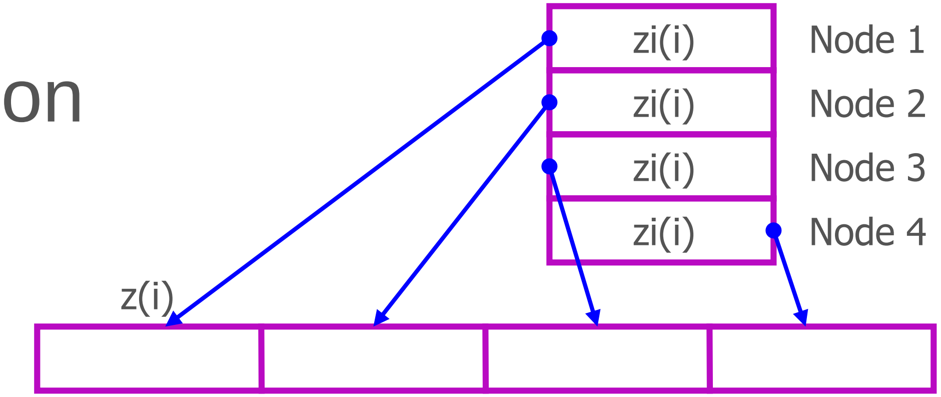
Data | Count

Sender

Recipient

MPI program example

- Collective communication



```
real zi(n)
j = 1
do i=i_st, i_end
   zi(j) = a*x(i) + y(i)
   j = j +1
enddo
call MPI_Gather(zi,n/nodes,MPI_REAL,z,n/nodes,MPI_REAL,
&                   0,MPI_COMM_WORLD,ierr)
```

Send data

Receive data

Root process

- Result on all nodes

```
call MPI_AllGather(zi,n/nodes,MPI_REAL,z,n/nodes,
&                   MPI_REAL,MPI_COMM_WORLD,ierr)
```

No root process

Clusters - login

- First log into one of the clusters

`ssh lonepeak.chpc.utah.edu` – Ethernet

`ssh ember.chpc.utah.edu` – Ethernet, InfiniBand

`ssh kingspeak.chpc.utah.edu` – Ethernet, InfiniBand

- Then submit a job to get compute nodes

```
sbatch -N 2 -n 24 -p ember -A chpc -t 1:00:00
--pty=/bin/tcsh -l
sbatch script.slr
```

- Useful scheduler commands

`sbatch` – submit a job

`scancel` – delete a job

`squeue` – show job queue

Security Policies

- No clear text passwords use ssh and scp
- You may not share your account under any circumstances
- Don’t leave your terminal unattended while logged into your account
- Do not introduce classified or sensitive work onto CHPC systems
- Use a good password and protect it

Security Policies

- Do not try to break passwords, tamper with files etc.
- Do not distribute or copy privileged data or software
- Report suspicions to CHPC (security@chpc.utah.edu)
- Please see http://www.chpc.utah.edu/docs/policies/security.html for more details

Compilation - OpenMP

- Different switches for different compilers, –openmp, –fopenmp or –mp

  e.g. `pgf77 -mp source.f -o program.exe`
- Nodes with up to 20 cores each
- Further references:

  Compilers man page – `man ifort`

  Compilers websites

  http://www.intel.com/software/products/compilers

  http://gcc.cnu.org

  http://www.pgroup.com/doc/

Compilation - MPI

- Two common network interfaces
  - – Ethernet, InfiniBand
- Different MPI implementations
  - – MPICH - Ethernet, InfiniBand
  - – OpenMPI – Ethernet, InfiniBand
  - – MVAPICH2 - InfiniBand
  - – Intel MPI – commercial, Ethernet, InfiniBand

Compilation - MPI

- **Clusters** – MPICH, OpenMPI, MVAPICH2, Intel MPI

  `/MPI-path/bin/mpixx source.x –o program.exe`

  `xx` = cc, cxx, f77, f90; icc, ifort for Intel MPI

- `MPI-path` = location of the distribution

  `/uufs/chpc.utah.edu/sys/installdir/mpich/std` MPICH Ethernet, InfiniBand

  `/uufs/$UUFSCELL.arches/sys/installdir/openmpi/std` OpenMPI Ethernet, InfiniBand

  `/uufs/$UUFSCELL.arches/sys/installdir/mvapich2/std` MVAPICH2 InfiniBand

  `/uufs/chpc.utah.edu/sys/installdir/intel/impi/std` Intel MPI Ethernet, InfiniBand

  = must specify full path to `mpixx` (`/MPI-path/bin`) or source the appropriate MPI distribution using modules

Running a parallel job – Clusters

- MPICH Interactive batch

```
sbatch -N 2 -n 24 -p ember -A chpc -t 1:00:00
--pty=/bin/tcsh -l
… wait for prompt …
module load intel mpich2
mpirun -np $SLURM_NTASKS program.exe
```

- MPICH Batch

```
sbatch -N 2 -n 24 -p ember -A chpc -t 1:00:00
--pty=/bin/tcsh -l
```

- OpenMP Batch

```
sbatch -N 1 -n 1 -p ember -A chpc -t 1:00:00
--pty=/bin/tcsh -l
setenv OMP_NUM_THREADS 12
program.exe
```

Compiling and running a parallel job – desktops

- Use MPICH or OpenMPI, MPICH is my preferred

```
module load mpich
mpixx source.x -o program.exe
```

xx = cc, cxx, f77, f90; icc, ifort for Intel MPI

- MPICH2 running

```
mpirun -np 4 ./program.exe
```

- OpenMP running

```
setenv OMP_NUM_THREADS 4
./program.exe
```

Single executable across desktops and clusters

- MPICH, MVAPICH2 and Intel MPI are cross-compatible using the same ABI
  - Can e.g. compile with MPICH on a desktop, and then run on the cluster using MVAPICH2 and InfiniBand
- Intel and PGI compilers allow to build "unified binary" with optimizations for different CPU platforms
  - But in reality it only works well under Intel compilers
- On a desktop

```
module load intel mpich2
mpicc -axCORE-AVX2 program.c -o program.exe
mpirun -np 4 ./program.exe
```

- On a cluster

```
srun -N 2 -n 24 ...
module load intel mvapich2
mpirun -np $SLURM_NTASKS ./program.exe
```

- https://www.chpc.utah.edu/documentation/software/single-executable.php

Debuggers

- Useful for finding bugs in programs
- Several free
  - `gdb` – GNU, text based, limited parallel
  - `ddd` – graphical frontend for gdb
- Commercial that come with compilers
  - `pgdbg` – PGI, graphical, parallel but not intuitive
  - `pathdb, idb` – Pathscale, Intel, text based
- Specialized commercial
  - `totalview` – graphical, parallel, CHPC has a license
  - `ddt` - Distributed Debugging Tool
- How to use:
- `http://www.chpc.utah.edu/docs/manuals/software/par_devel.html`

Debuggers - parallel

- Parallel debugging more complex due to interaction between processes
- Totalview is the debugger of choice at CHPC
- Expensive but academia get discount
- How to run it:
  - compile with `-g` flag
  - automatic attachment to OpenMP threads
  - extra flag (`-tv`) to mpirun/mpiexec
- Details:

`http://www.chpc.utah.edu/docs/manuals/software/totalview.html`

- Further information

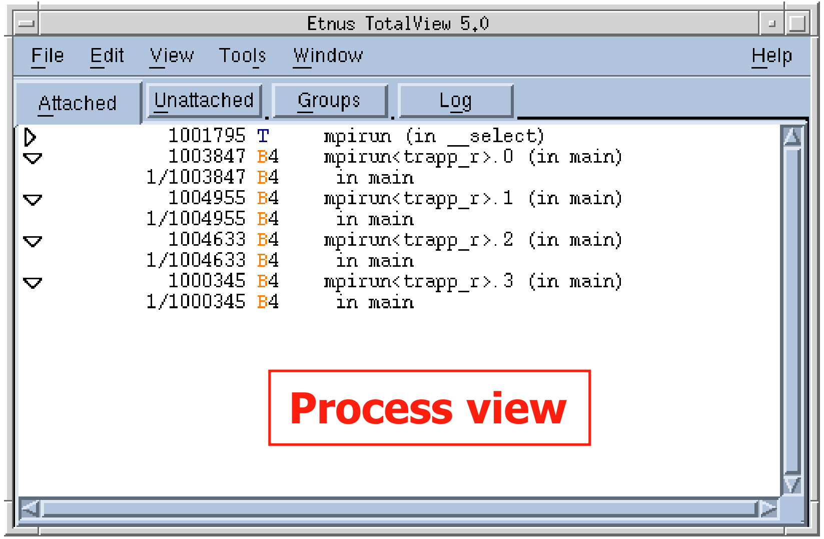
`http://www.roguewave.com/products-services/totalview`

TOGETHER WE REACH

THE UNIVERSITY OF UTAH™

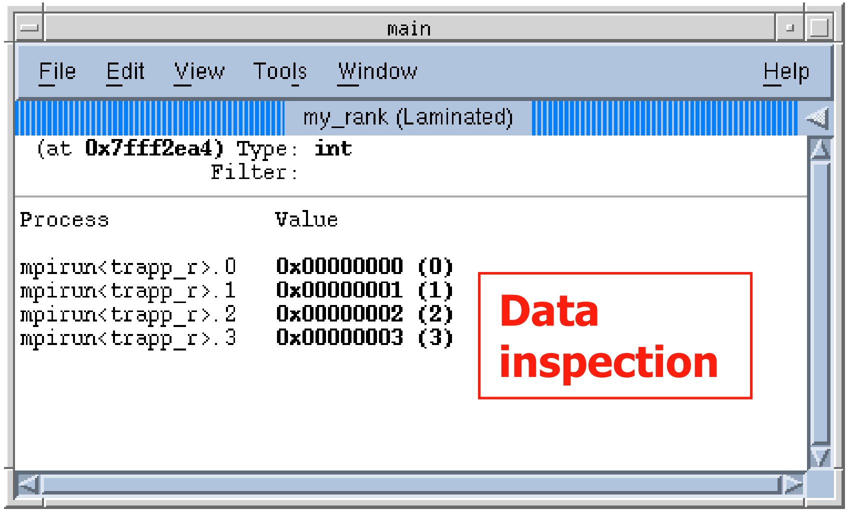
Center for High-Performance Computing

Debuggers – parallel



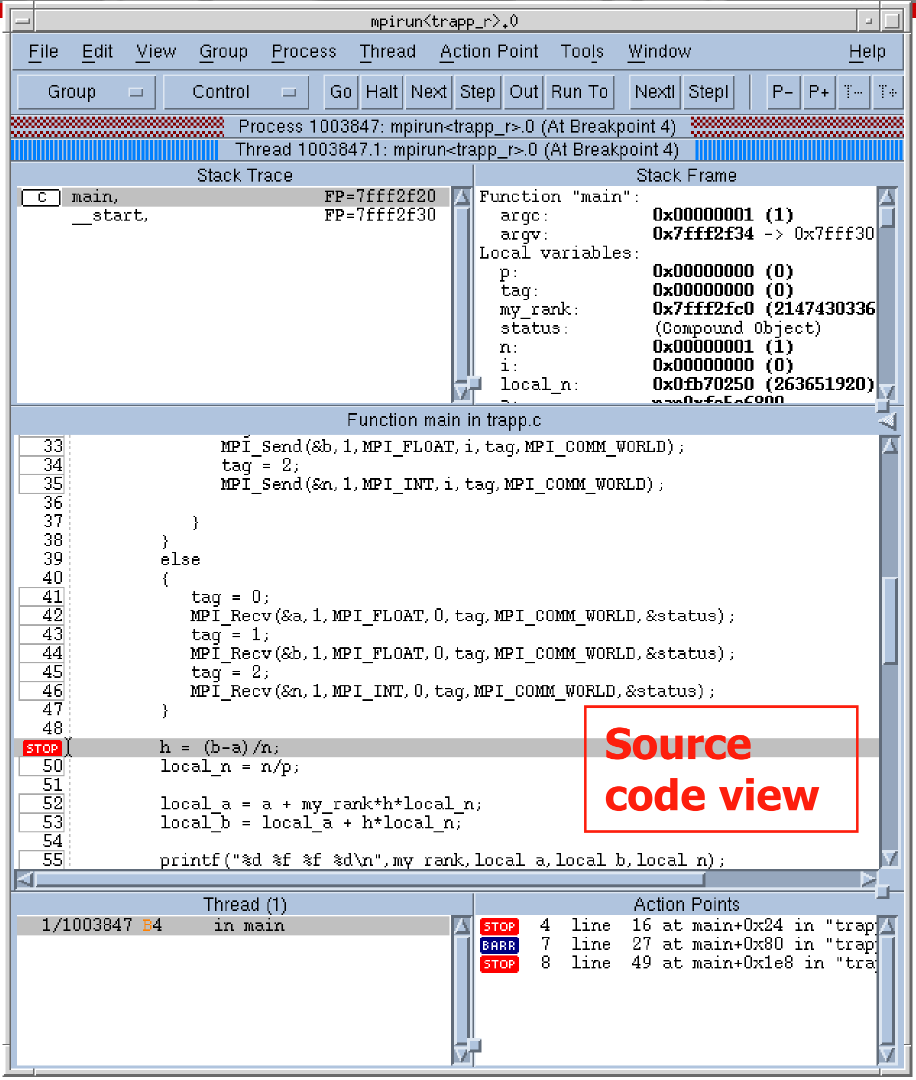
```
   1001795 T    mpirun (in __select)
   1003847 B4   mpirun<trapp_r>.0 (in main)
 1/1003847 B4    in main
   1004955 B4   mpirun<trapp_r>.1 (in main)
 1/1004955 B4    in main
   1004633 B4   mpirun<trapp_r>.2 (in main)
 1/1004633 B4    in main
   1000345 B4   mpirun<trapp_r>.3 (in main)
 1/1000345 B4    in main
```

Process view



```
(at 0x7fff2ea4) Type: int
             Filter:

Process            Value

mpirun<trapp_r>.0  0x00000000 (0)
mpirun<trapp_r>.1  0x00000001 (1)
mpirun<trapp_r>.2  0x00000002 (2)
mpirun<trapp_r>.3  0x00000003 (3)
```

Data inspection



Process 1003847: mpirun<trapp_r>.0 (At Breakpoint 4)

Thread 1003847.1: mpirun<trapp_r>.0 (At Breakpoint 4)

```
C  main,              FP=7fff2f20
   __start,           FP=7fff2f30
```

```
Function "main":
  argc:               0x00000001 (1)
  argv:               0x7fff2f34 -> 0x7fff30
Local variables:
  p:                  0x00000000 (0)
  tag:                0x00000000 (0)
  my_rank:            0x7fff2fc0 (2147430336
  status:             (Compound Object)
  n:                  0x00000001 (1)
  i:                  0x00000000 (0)
  local_n:            0x0fb70250 (263651920)
```

```
                MPI_Send(&b,1,MPI_FLOAT,i,tag,MPI_COMM_WORLD);
                tag = 2;
                MPI_Send(&n,1,MPI_INT,i,tag,MPI_COMM_WORLD);

            }
        }
        else
        {
            tag = 0;
            MPI_Recv(&a,1,MPI_FLOAT,0,tag,MPI_COMM_WORLD,&status);
            tag = 1;
            MPI_Recv(&b,1,MPI_FLOAT,0,tag,MPI_COMM_WORLD,&status);
            tag = 2;
            MPI_Recv(&n,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
        }

STOP    h = (b-a)/n;
        local_n = n/p;

        local_a = a + my_rank*h*local_n;
        local_b = local_a + h*local_n;

        printf("%d %f %f %d\n",my_rank,local_a,local_b,local_n);
```

```
1/1003847 B4     in main
```

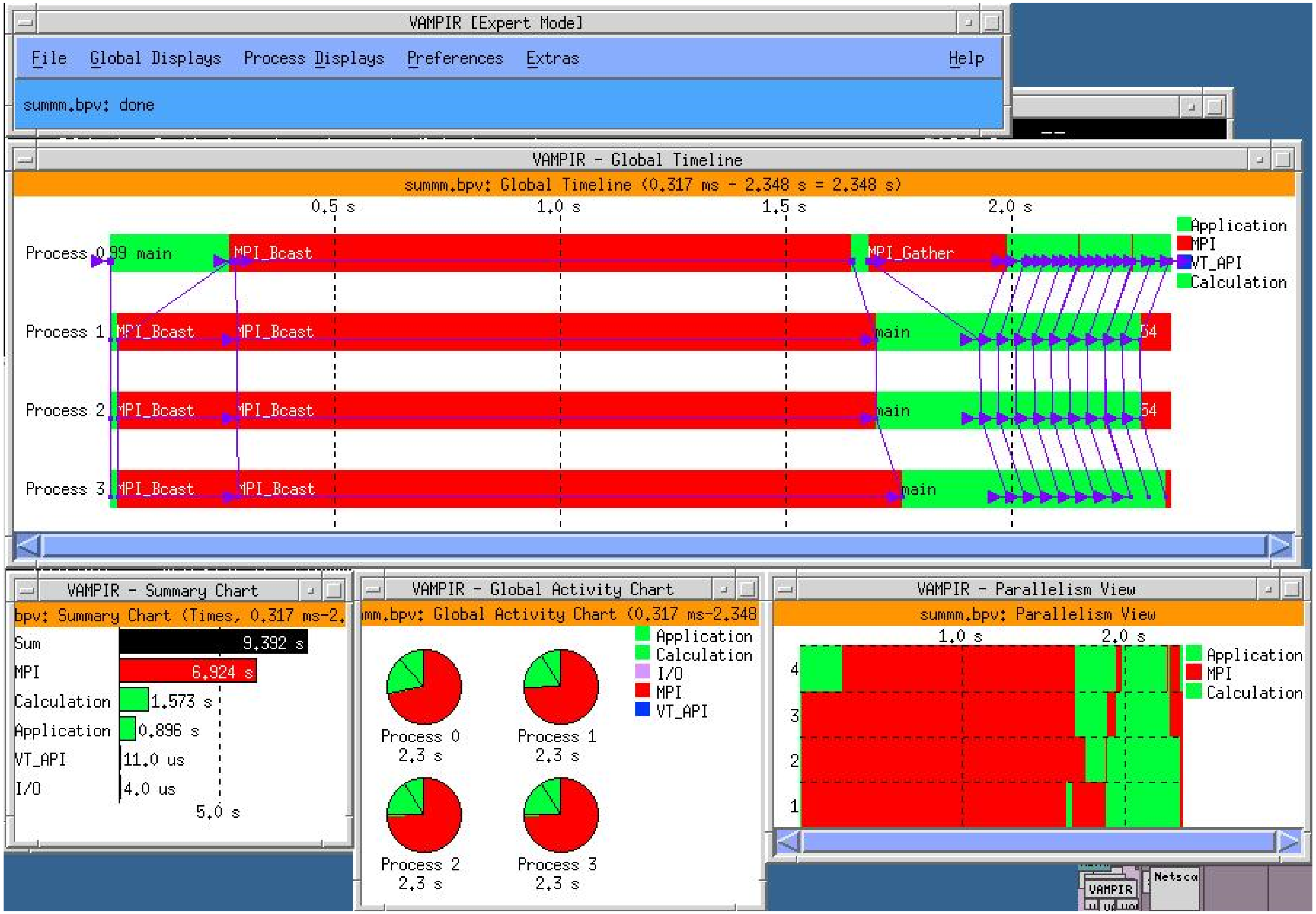
```
STOP  4  line  16 at main+0x24 in "trap
BARR  7  line  27 at main+0x80 in "trap
STOP  8  line  49 at main+0x1e8 in "tra
```

Source code view

Profilers

- Measure performance of the code
- Serial profiling
  - – discover inefficient programming
  - – computer architecture slowdowns
  - – compiler optimizations evaluation
  - – gprof, pgprof, pathopt2, Intel tools
- Parallel profiling
  - – target is inefficient communication
  - – Intel Trace Collector and Analyzer, InspectorXE, VTune

Profilers - parallel



summm.bpv: done

summm.bpv: Global Timeline (0.317 ms - 2.348 s = 2.348 s)

bpv: Summary Chart (Times, 0.317 ms-2.

```
Sum          9.392 s
MPI          6.924 s
Calculation  1.573 s
Application  0.896 s
VT_API       11.0 us
I/O          4.0 us
                     5.0 s
```

Libraries

- Serial
  - BLAS, LAPACK – linear algebra routines
  - MKL, ACML – hardware vendor libraries
- Parallel
  - ScaLAPACK, PETSc, NAG, FFTW
  - MKL – dense and sparse matrices

`http://www.chpc.utah.edu/docs/manuals/software/mat_l.html`

Summary

- Shared vs. Distributed memory
- OpenMP
  - limited on Arches
  - Simple parallelization
- MPI
  - Arches
  - Must use communication

http://www.chpc.utah.edu/docs/presentations/intro_par

References

- OpenMP

  `http://www.openmp.org/`

  Chandra, et. al. - Parallel Programming in OpenMP

  Chapman, Jost, van der Pas – Using OpenMP

- MPI

  `http://www-unix.mcs.anl.gov/mpi/`

  Pacheco - Parallel Programming with MPI

  Gropp, Lusk, Skjellum - Using MPI 1, 2

- MPI and OpenMP

  Pacheco – An Introduction to Parallel Programming

Future Presentations

- Introduction to MPI
- Introduction to OpenMP
- Debugging with Totalview
- Profiling with TAU/Vampir
- Intermediate MPI and MPI-IO
- Mathematical Libraries at the CHPC