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.
- Please give us feedback

https://www.surveymonkey.com/r/KHVDC5H

Parallel architectures

Single processor:

- SISD – single instruction single data.

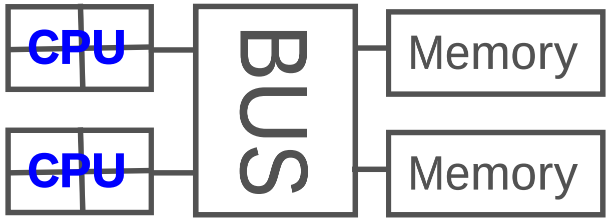
Multiple processors:

- 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

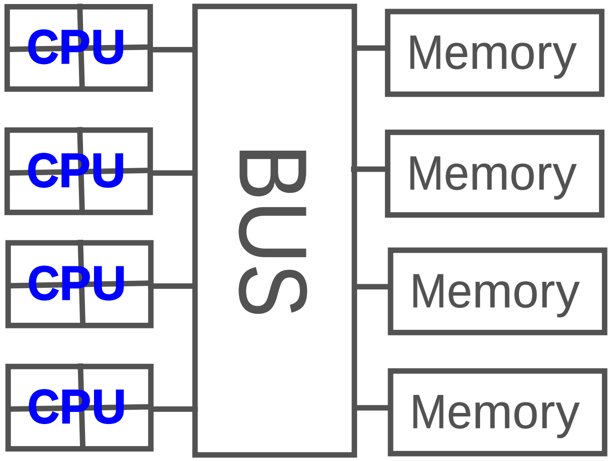
Shared memory

- All processors have access to local memory
- Simpler programming
- Concurrent memory access
- More specialized hardware
- CHPC :
  Linux clusters 8, 12, 16, 20, 24, 28, 32 core nodes
  GPU nodes

Dual quad-core node

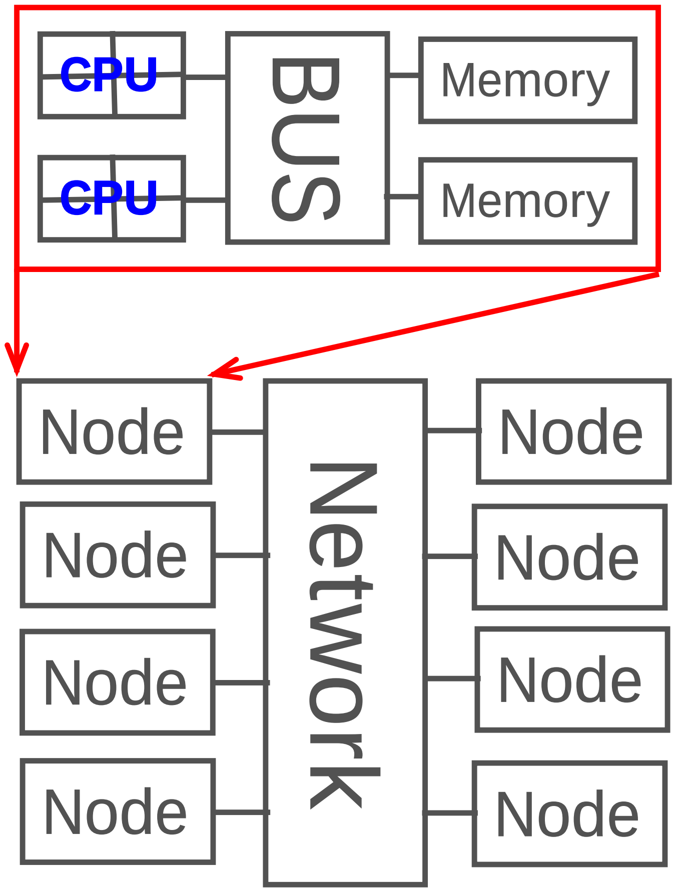


Many-CPU node (e.g. SGI)



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

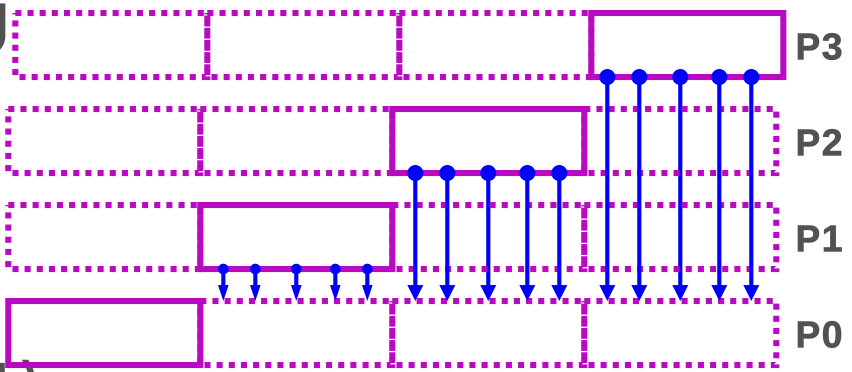
```
subroutine saxpy_parallel_mpi(z, a, x, y, n)
integer i, n, ierr, my_rank, nodes, i_st, i_end
real z(n), a, x(n), y(n)

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 (tasks)

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

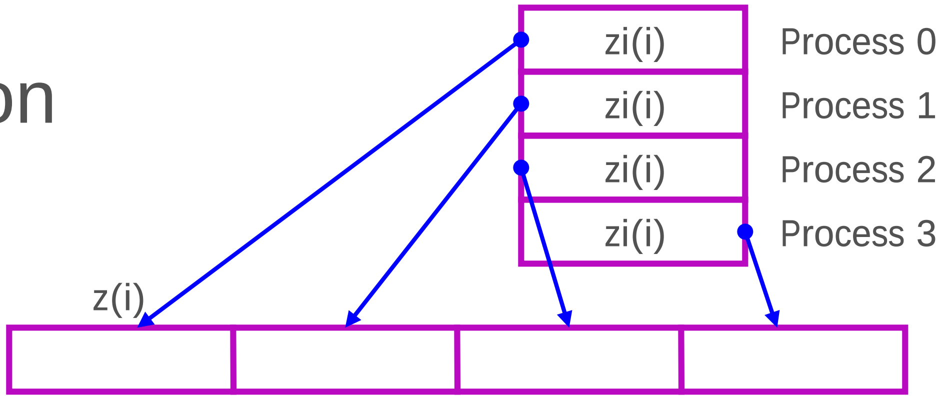
- Collective communication

```
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

But wait, my program is not in C or Fortran

Interpreted languages are popular

- Matlab, Python, R

Each has some sort of parallel support, but most likely it will not perform as well as using OpenMP or MPI with C/Fortran.

Try to parallelize (and optimize) your Matlab/Python/R code and if it’s still not enough consider rewriting in C++ or Fortran.

Matlab

Threads

- Built in Matlab functions. Vector/matrix operations threaded (and vectorized) through Intel MKL library, many other functions also threaded

Tasks (processes)

- *Parallel Computing Toolbox* allows for task based parallelism
- *Distributed Computing Server* can distribute tasks to multiple nodes
- Great for independent calculations, when communication is needed uses MPI under the hood

https://www.chpc.utah.edu/documentation/software/matlab.php

Python

Threads

- No threads in Python code because of GIL (Global Intepreter Lock)
- C/Fortran functions can be threaded (e.g. *NumPy*)

Tasks (processes)

- Several libraries that use MPI under the hood, most popular is *mpi4py*
- More-less MPI function compatibility, but slower communication because of the extra overhead
- Also many other data-parallel libraries, e.g. *Dask*

https://www.chpc.utah.edu/documentation/software/python.php

R

Threads

- Under the hood threading with CHPC built (or Microsoft) R for vector/matrix operations using MKL
- *parallel* R library

Tasks (processes)

- *parallel* R library (uses *multicore* for shared and *snow* for distributed parallelism)
- Parallelized **apply* functions, e.g. *mclapply*
- *Rmpi* library provides MPI like functionality
- Many people run multiple independent R instances in parallel

https://www.chpc.utah.edu/documentation/software/r-language.php

Clusters - login

- First log into one of the clusters

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

  `ssh ember.chpc.utah.edu; ssh kingspeak.chpc.utah.edu; ssh notchpeak.chpc.utah.edu` – Ethernet, InfiniBand

- May debug and do short test runs (< 15 min, <= 4 processes/threads) on interactive nodes

- Then submit a job to get compute nodes

```
srun –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

Compilation - OpenMP

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

  ```
  module load intel
  module load pgi
  module load gcc
  ```

  e.g. `pgf77 –mp source.f –o program.exe`
- Nodes with up to 32 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 – set by `module load`
  `module load mpich` MPICH Ethernet, InfiniBand
  `module load openmpi` OpenMPI Ethernet, InfiniBand
  `module load mvapich2` MVAPICH2 InfiniBand
  `module load impi` Intel MPI Ethernet, InfiniBand

  = after this simply use `mpixx`
- Ensure that when running (using `mpirun`), the same module is loaded.

Running a parallel job – Clusters

- MPICH Interactive batch

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

- MPICH Batch

```
sbatch –N 2 –n 24 –p ember –A chpc –t 1:00:00
script.slr
```

- OpenMP Batch

```
srun –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

- MPICH running

```
mpirun –np 4 ./program.exe
```

- OpenMP running

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

- See more details/combinations at `https://www.chpc.utah.edu/documentation/software/mpilibraries.php`

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 mpich
mpicc –axCORE-AVX512,CORE-AVX2,AVX 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
  - **Intel Inspector XE** – memory and threading error checker
- How to use:
- `http://www.chpc.utah.edu/docs/manuals/software/par_devel.html`

Debuggers - parallel

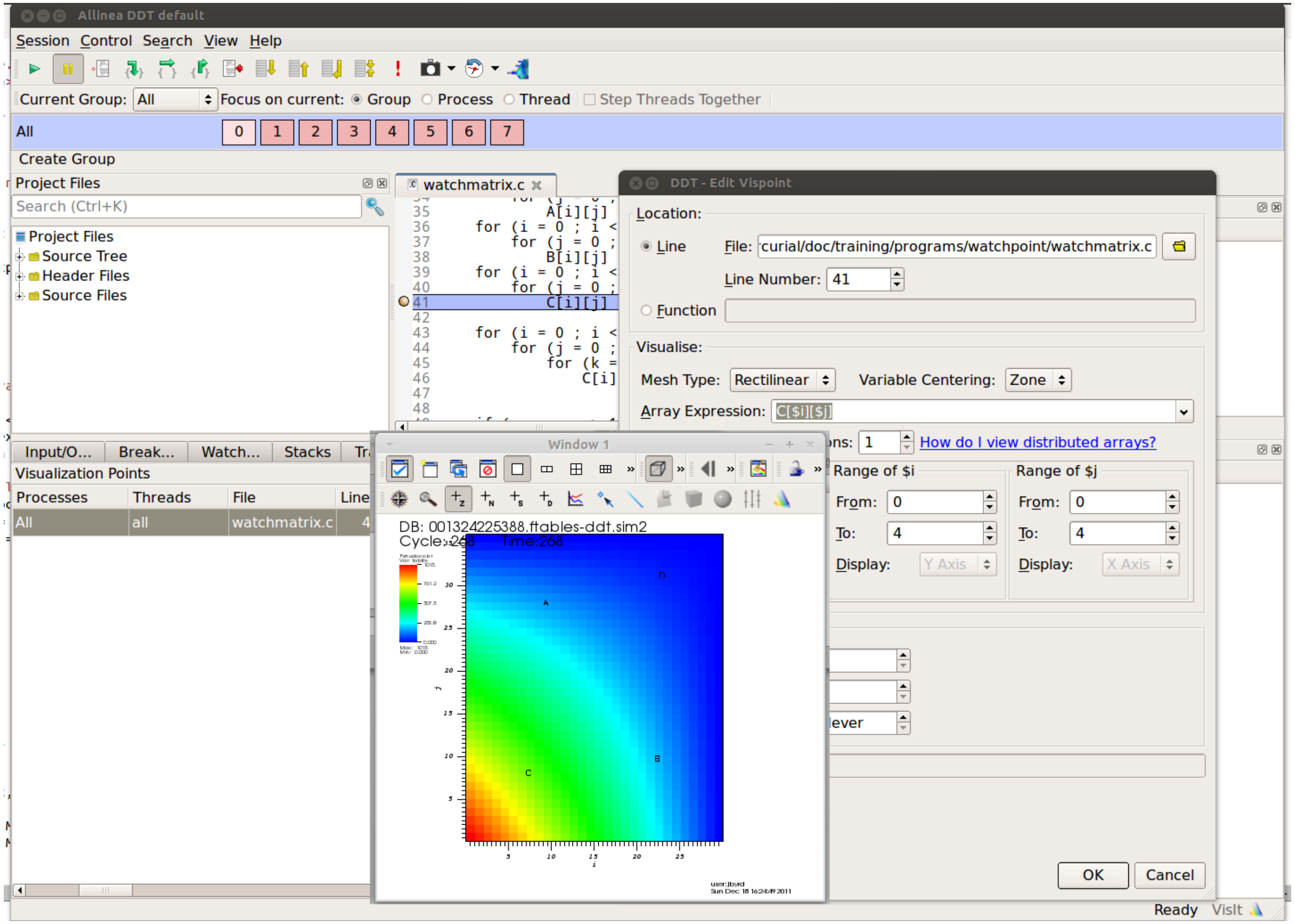
- Parallel debugging more complex due to interaction between processes
- DDT is the debugger of choice at CHPC
- Expensive but academia get discount
- How to run it:
  - compile with –g flag
  - run `ddt` command
  - fill in information about executable, parallelism, …
- Details:

`https://www.chpc.utah.edu/documentation/software/debugging.php`

- Further information

`https://www.allinea.com/products/ddt`

Debuggers – parallel



Allinea DDT default

Current Group: All Focus on current: [✓ Group] Process Thread Step Threads Together

All	0	1	2	3	4	5	6	7

```
35            A[i][j]
36      for (i = 0 ; i <
37          for (j = 0 ;
38              B[i][j]
39      for (i = 0 ; i <
40          for (j = 0 ;
41              C[i][j]
42
43      for (i = 0 ; i <
44          for (j = 0 ;
45              for (k =
46                  C[i]
47
48
```

Visualization Points

Processes	Threads	File	Line
All	all	watchmatrix.c	4

DDT - Edit Vispoint

Location:

[✓ Line] File: curial/doc/training/programs/watchpoint/watchmatrix.c

Line Number: 41

Function

Visualise:

Mesh Type: Rectilinear Variable Centering: Zone

Array Expression: C[$i][$j]

ns: 1 How do I view distributed arrays?

Range of $i		Range of $j	
From:	0	From:	0
To:	4	To:	4
Display:	Y Axis	Display:	X Axis

ever

OK Cancel

Window 1

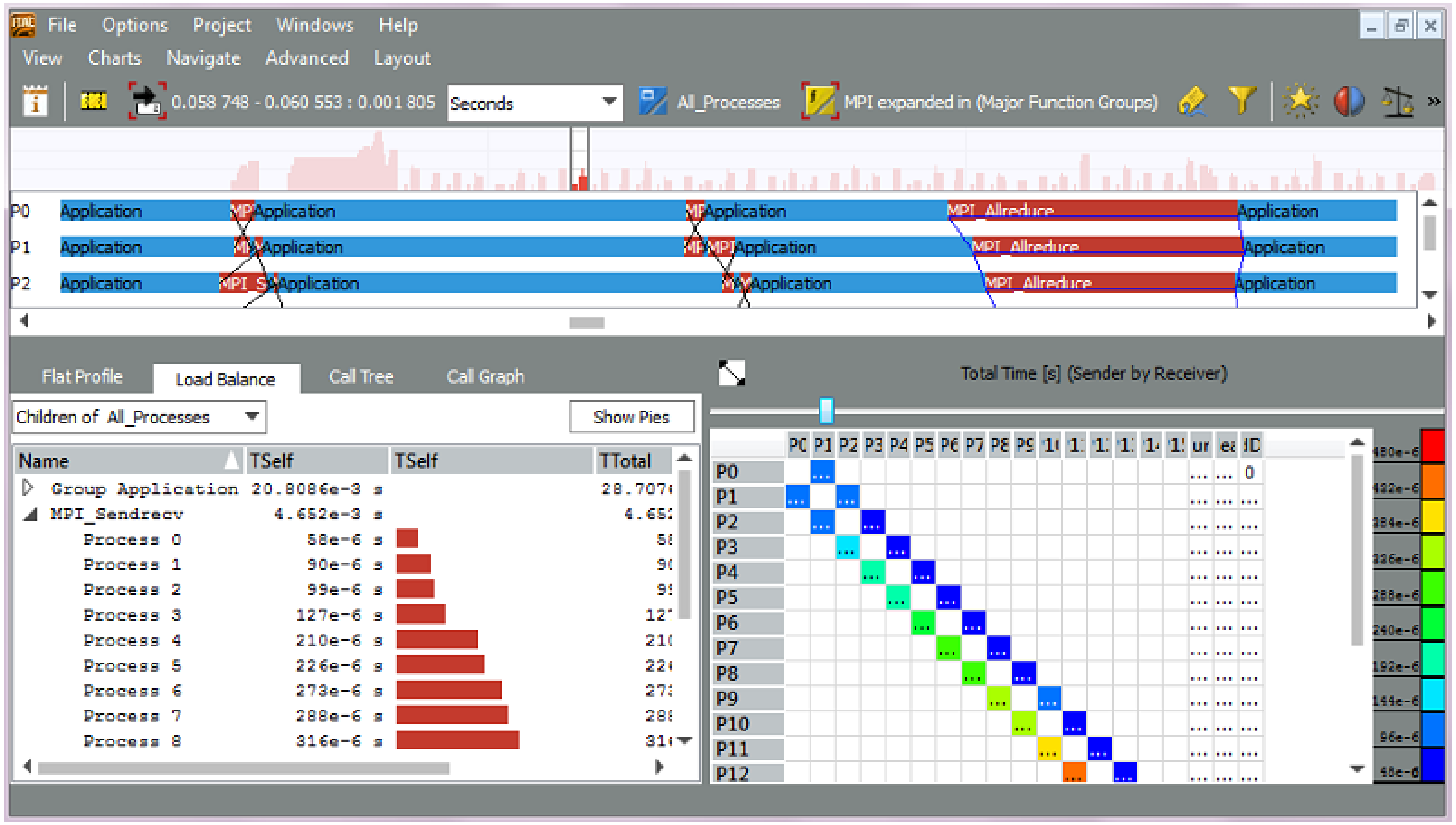
DB: 001324225388.ftables-ddt.sim2
Cycle: 268 Time:268

Ready VisIt

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, AdvisorXE, VTune**

Profilers - parallel



0.058 748 - 0.060 553 : 0.001 805 Seconds

Children of All_Processes

Name	TSelf	TSelf	TTotal
▷ Group Application	20.8086e-3 s		28.707
◢ MPI_Sendrecv	4.652e-3 s		4.65
Process 0	58e-6 s	🟥	5
Process 1	90e-6 s	🟥	9
Process 2	99e-6 s	🟥	9
Process 3	127e-6 s	🟥	12
Process 4	210e-6 s	🟥	21
Process 5	226e-6 s	🟥	22
Process 6	273e-6 s	🟥	27
Process 7	288e-6 s	🟥	28
Process 8	316e-6 s	🟥	31

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 to 1 cluster node
  - Simple parallelization
- MPI
  - Clusters
  - 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
- Introduction to Debugging
- Introduction to Profiling
- Hybrid MPI-OpenMP programming
- Introduction to I/O at CHPC

Feedback

https://www.surveymonkey.com/r/KHVDC5H