mpipipe

A Parallelized Unix Pipe for Serial Filter Programs

Nathan Weeks <weeks@iastate.edu>
Pipes

- Task: align DNA sequences in two files to the soybean genome. Keep alignments >100 in length.

- Example without pipes:

```bash
$ ls
DNA1.fasta.gz DNA2.fasta.gz
$ gunzip DNA?.fasta.gz
$ blastn -infile DNA1.fasta -db Glycine_max > temp1.txt
$ blastn -infile DNA2.fasta -db Glycine_max > temp2.txt
$ awk '$5 - $4 > 100' temp1.txt temp2.txt > output.txt
```

- Note the use of temporary files
Pipes

- Pipe: a one-way communication channel between two processes on the same host

- Previous example with pipes

  $ gzcat DNA?.fasta.gz |

    blastn -db Glycine_max |

    awk '$5 - $4 > 100' > finaloutput.txt

- Note the lack of temporary intermediate files
Problem

• Task: parallelize the previous example to run multiple simultaneous instances of blastn on 32 nodes

• Data distribution
  – Typical solution: split the two files into 32 files (and hope you don't exceed your quota or available disk space) and submit 32 blastn jobs

• Load Balancing
  – What if the time to process one of the 32 input files is much greater than the others (e.g., slow cluster node, longer sequences, or repetitive sequence?)
gzcat DNA?.fasta.gz |
mpirun -np 33 \
mpipipe -n 2000 'blastn -db Glycine_max' |
awk '$5 - $4 > 100' > output.txt
Solution

gzcat DNA?.fasta.gz |

mpirun -np 33 \

mpipipe -n 2000 'blastn -db Glycine_max' |

awk '$5 - $4 > 100' > output.txt

Uncompress & concatenate input files
Solution

gzcat DNA?.fasta.gz |

mpirun -np 33 \

mpipipe -n 2000 'blastn -db Glycine_max' |

awk '$5 - $4 > 100' > output.txt

(Parallelization) Run 32 worker processes, each of which runs the user-specified command, plus + 1 manager process that sends input to the workers & collects output from the workers
Solution

gzcat DNA?.fasta.gz |

mpirun -np 33 \

mpipipe -n 2000 'blastn -db Glycine_max' |

awk '$5 - $4 > 100' > output.txt

(Data distribution & load balancing) Send 2000 “records” at a time to each worker process, collecting output from the worker process before sending it another 2000.
Solution

gzcat DNA?.fasta.gz |
mpirun -np 33 \
mpipipe -n 2000 'blastn -db Glycine_max' |
awk '$5 - $4 > 100' > output.txt

Output is printed (non-interleaved) to standard output; in this case, .
fork()

- Creates a new ("child") process
  - clone of the parent (same program, same state)
  - has its own copy of all memory/variables in program
  - begins execution at the call to fork()

```c
if (fork() == 0) {
    // child process executes this
} else {
    // parent process executes this
}
```
Manager

MPI_Sendrecv()

Worker

MPI_Sendrecv()

INPUT1

STDOUT

INPUT2
Manager
MPI_Sendrecv()

STDIN

Worker
INPUT1
pipe[READ]
pipe[WRITE]
fork()

Worker(child)
INPUT1
pipe[READ]
pipe[WRITE]

STDOUT

STDOUT

INPUT1
pipe[READ]
pipe[WRITE]
Manager

MPI_Sendrecv()

Worker

\texttt{free(INPUT1)}

Worker(child)

\texttt{INPUT1}

\texttt{pipe[READ]}

\texttt{pipe[WRITE]}

STDIN

INPUT2

STDOUT

STDOUT

STDIN

INPUT1

STDOUT

STDOUT

STDOUT
Manager

MPI_Sendrecv()

Worker

OUTBUF
pipe[READ]
malloc(OUTBUF)

Worker(child)

INPUT1
pipe[WRITE]
dup2(pipe[WRITE], STDOUT)

STDIN

STDOUT

STDOUT

INPUT2
Manager

STDIN

MPI_Sendrecv()

INPUT2

Worker

OUTBUF

pipe[READ]

Worker(child)

INPUT1

close(pipe[WRITE])

STDOUT

STDOUT

STDIN

OUTBUF

pipe[READ]
Manager

MPI_Sendrecv()

Worker

OUTBUF
read(pipe[READ])

Worker(child)

INPUT1
popen(cmdpipe)

STDIN

STDOUT

COMMAND

STDIN

STDOUT

OUTBUF

STDIN

INPUT1

STDOUT
Manager

MPI_Sendrecv()

Worker

OUTBUF

read(pipe[READ])

Worker(child)

INPUT1

write(cmdpipe,INPUT1)

STDIN

STDOUT

WRITE

INPUT2

read(pipe[READ])

STDIN

STDOUT

OUTBUF

INPUT1

READ

STDOUT

STDOUT

STDOUT
Manager
MPI_Sendrecv()

Worker
READ
read(pipe[READ])

Worker(child)
INPUT1
fwrite(cmdpipe,INPUT1)

OUTBUF

COMMAND

STDIN

STDOUT

STDIN

STDOUT

STDOUT

STDOUT

STDOUT

STDOUT

STDOUT
Manager
MPI_Sendrecv()

Worker
OUTBUF
read(pipe[READ])

Worker(child)
INPUT1
fwrite(cmdpipe,INPUT1)

STDIN

STDOUT

INPUT2

STDOUT

OUTBUF

INPUT1

STDOUT

COMAND

INPUT1

OUTPUT1

STDOUT

STDOUT
Manager

MPI_Sendrecv()

Worker

OUTBUF
read(pipe[READ])

Worker(child)

INPUT1
fwrite(cmdpipe,INPUT1)

STDIN
INPUT1

STDOUT
INPUT1

OUTBUF

STDPipe
INPUT1

STDOUT

OUTPUT1

COMMAND

INPUT1

STDOUT

OUTPUT1

STDOUT

INPUT1

STDOUT

INPUT1

STDOUT

INPUT1

STDOUT

INPUT1

STDOUT
Manager
MPI_Sendrecv()

Worker
read(pipe[READ])
OUTPUT1

Worker(child)
write(cmdpipe, INPUT1)

STDIN
INPUT1
STDOUT

INPUT1
STDOUT

INPUT2
STDOUT

OUTPUT1
STDOUT

CommandLine
INPUT1
STDOUT

OUTPUT1
STDOUT

Manager
MPI_Sendrecv()

Worker
OUTPUT1
read(pipe[READ])

Worker(child)
INBUF
pclose(cmdpipe,INPUT1)

STDIN

STDOUT

INPUT2

STDOUT

INPUT1

OUTPUT1

STDOUT

INPUT1

COMMAND

STDIN

STDOUT

INPUT1

STDOUT

OUTPUT1

STDOUT

OUTPUT1

STDOUT

OUTPUT1

STDOUT
Manager

MPI_Sendrecv()

Worker

OUTPUT1
read(pipe[READ])

Worker(child)

INBUF
pclose(cmdpipe,INPUT1)

STDIN

STDOUT

INPUT2

STDIN

STDOUT

COMMAND

INPUT1

OUTPUT1

STDOUT

OUTPUT1

STDOUT

OUTPUT1

STDOUT

pclose(cmdpipe,INPUT1)
Manager
MPI_Sendrecv()

Worker
read(pipe[READ])

Worker(child)
pclose(cmdpipe,INPUT1)

STDIN
STDOUT
INPUT2
STDIN
STDOUT
INBUF
STDOUT
COMMAND
STDOUT
STDIN
STDOUT
OUTPUT1
STDOUT
STDIN
STDOUT
OUTPUT1
STDOUT
Manager

MPI_Sendrecv()

Worker

OUTPUT1

close(pipe[READ])

Worker(child)

INBUF

pclose(cmdpipe,INPUT1)

STDIN

STDOUT

COMMAND

STDIN

STDOUT

STDOUT

STDOUT

STDIN

STDOUT

STDOUT
Manager

STDIN

INPUT2

MPI_Sendrecv()

STDOUT

Worker

OUTPUT1

MPI_Sendrecv()

Worker(child)

INBUF

_exit()

STDOUT
Manager

STDIN

STDOUT

MPI_Sendrecv()

Worker

BUF

MPI_Sendrecv()

INPUT2

OUTPUT1
Benchmark

- System: hpc-class
- Program: blastn (NCBI blast-2.2.19+)
- Input Data
  - Query
    - 70,229 “454” EST sequences (average length approx. 233 nucleotides)
  - Target
    - Glycine max (soybean) genome
      - 955,054,837 (known) nucleotides
Benchmark

blastn Timings

70,229 454 sequences vs. Soybean genome

Seconds vs. Processes

1 Thread
2 Threads
Problems

- Open MPI uses polling for blocking MPI calls
  - Excessive CPU usage while waiting for a message
  - Possible workaround: replace “MPI_Recv(...)” with

```c
received = false;
MPI_Irecv(..., request,...);
do {
    MPI_Test(request, received,...);
    if (received == true) break;
    else sleep(seconds);
} while (true)
```
Tested Configurations

- Mac OS X 10.5.x (Intel iMac)
  - Open MPI 1.2.3 (included in default installation)
- Solaris 10 (Sun Blade 1000 workstation)
  - Sun HPC Cluster Tools 8.1 (Open MPI 1.3)
- Linux (hpc-class cluster)
  - Open MPI 1.3.1
MPIPIPE

Questions?