Parallelized coding
(and profiling)
on genepool

Rob Egan
JGI
Types of parallelism

● Fine grained
  ○ Lots of communication / syncs
    ■ "Game of Life"
    ■ Climate modelling
● Coarse grained
  ○ Moderate communication / few syncs
    ■ 'Map' phase of Map-Reduce
● Embarrassingly parallel
  ○ Zero communication / 0 syncs
  ○ Deterministic partitioning
    ■ Read mapping to a reference
    ■ Mandelbrot
Types of parallelism

- Process multi-tasking, pipelining
  - "for f in *.gz ; do gunzip $f & done; wait"
  - "for j in *.sh ; do qsub $j ; done"
- Array jobs
  - "qsub -t 1-20000 myjob"
  - embarrassingly parallel
- Threaded applications
  - "blast -t 8 ..."
  - "bowtie2 -p 16 ..."
  - UI, some fine-grained & coarse-grained
- MPI applications
  - "mpirun -n 384 myProg"
  - fine-grained, coarse-grained & embarrassingly parallel
Why you should write parallel code!

- **Shorter duration to the solution**
  - More cores -> less walltime
- **Larger problems become achievable**
  - Divide and conquer
- **Use a node more effectively**
  - \#cores / node increasing
  - (network | memory) / core decreasing
- **Use accelerated hardware**
  - Hyper-threading
  - GPUs
  - FPGA (Convey, Timelogic)
- **Use less memory per core**
  - Large, shared global structure (best when read-only)
Why you should NOT!...???

● Harder to get right
  ○ Race conditions, corruption & bugs

● Hard to get fast / efficient
  ○ Amdahl's Law
  ○ Semaphores, mutexes
  ○ System bottlenecks

● Few thread-safe libs
  ○ (But improving)

● Need to learn new libs
  ○ OpenMP, pthreads
  ○ MPI
  ○ nvcc, opencl, openacc
But there are some easy things to do

1. Rely on someone else's parallel code
   a. Pipeline engines: make, ant, galaxy, xargs, GE
   b. TaskFarmer / TFMQ
   c. Use threaded or MPI coded versions

2. Run multiple programs in one job
   a. for i in *.sh ; do $i & done; wait

3. Run job arrays
   a. qsub -t 1-20000 myJobArray.sh

4. Add threads to your code
   a. Java, Perl, Python: Threading classes / modules
   b. C/C++: OpenMP, pthreads, boost::thread
Grid Engine Parallel Environments

● Describe to GE your job's shape
  ○ `qsub -pe <parallel_environment> <num_slots>`

● JGI PE names: `pe_X`
  ○ `pe_slots <num_slots>`
    ■ allocate cores on a **single** node
  ○ `pe_<int> <num_slots>` (i.e. `pe_1`, `pe_8`, `pe_16`)
    ■ allocate `<int>` blocks of cores on **many** nodes
    ■ (`<num_slots>` must be a multiple of `<int>`)  
  ○ `pe_fill <num_slots>`
    ■ allocate cores on **any** available node filling first
  ○ `pe_rrobin <num_slots>`
    ■ allocate cores on **any** available node roundrobin
pe_slots

Finds most loaded single node with available slots
Best for threaded programs & process parallelism

-pe pe_slots 3
-pe pe_slots 2
-pe pe_slots 3
-pe pe_slots 2

Wait Queue
pe_1, pe_2, etc

Finds most loaded nodes with available slots
Best for fine-grained MPI

-pe pe_1 3
-pe pe_2 4
-pe pe_1 3
pe_fill

Finds and fills most loaded nodes with any available slots
Best for coarse-grained MPI; starts quickly

-pe pe_fill 6
pe_rrobin

Finds and spreads across nodes with any available slots
Best for coarse-grained MPI; starts quickly

-pe pe_rrobin 6
Mendel/GP2 nodes & High Mem

- Nodes are scheduled exclusively
- 16 cores * 2 threads, 120G, infiniband
  - Need to run parallel jobs for efficiency!
Running threaded programs

$ qsub -pe pe_slots <num_threads> myJob.sh

OpenMP: setenv OMP_NUM_THREADS=num

Your Application: find the option (-t, -p, -n, etc)

blastx -num_threads
bwa aln -t
bowtie -p
bowtie2 -p
usearch (automatic)
Running array jobs on GE

SGE_TASK_ID changes for each task

$ qsub -t 1-20000 myJobArray.sh
$ cat myJobArray.sh

#!/bin/bash
myfile=input.$SGE_TASK_ID
runSomething $myfile

Or better yet, use TaskFarmerMQ
Reduce GE overhead for variable length jobs
Running MPI jobs

OpenMPI works directly with GE

$ qsub -pe pe_8 64
$ cat myMPIJob.sh

#!/bin/bash
module load openmpi
mpirun myProgram
Use `qacct` to assess usage
```
qsub -t 1-200 -l ram.c=120G ...
```

How efficient is my job?
```
$ qacct -j 4774143

qname        long_excl.q
hostname      mc0202-ib.nersc.gov
jobnumber     4774143
taskid        24
granted_pe    pe_slots
slots         16
ru_wallclock  10886
ru_utime      10657.092
ru_stime      180.425
cpu           10837.516
mem           89670.590
io            221.227
maxvmem       8.348G
```

Memory:
\[
\text{8.34 G / 120G} = 6.95\%
\]

Cores:
\[
\frac{(10657+180)}{(10886 \times 16)} = 6.02\%
\]

Network:
\[
\frac{221 \times 8}{10886 / 4 \text{ Gbit}} = 4\%
\]
Running array jobs on GP2 / Mendel

Exclusive (1 job per node)

$ qsub -t 1-20000 -pe pe_slots 16 job.sh
$ cat job.sh
    #!/bin/bash
    myfile=file.$SGE_TASK_ID
    bowtie2 -t 32 ... $myfile
module load jgibio (also in /jgi/tools/bin)

- **Profiling & performance analysis**
  - `sar / sysstats`
  - `java -jar /jgi/tools/misc_software/kSar/DEFAULT/kSar.jar -input cmd://ssh -C genepool ssh sgi05a38.nersc.gov sar -i 10 -A -f /tmp/sysstats-trials.sh.4764703.undefined.20130208_045047.15860`

- **Easy to stage in/out datasets**
  - `STAGE_CWD=`pwd` (or use localize_file)`

- **Signal trapping**
  - Know when/why GE terminates the job

- **Helps array jobs on exclusive nodes**
  - Automatically runs tasks between stepsize
  - `-t 1-20000:16`

- **Automatic Checkpointing (coming soon)**
Run more in the job...
qsub -t 1-200:10 -l ram.c=120G jgi_job_wrapper.sh ...

Run 10 tasks per job...

How efficient is my job?

$ qacct -j 4774143

Memory:
83.4 G / 120G = 69.5%

Cores:
(106570+1800) / (10886 * 16) = 60.2%

Network:
2210 * 8 / 10886 / 4 Gbit = 40%
Some profiling of PacBio Code

jgi_job_wrapper.sh job.sh
jgi_job_wrapper.sh STAGE_CWD=$(pwd) job.sh

- **GPFS vs stage-in/out on /scratch**
  - 18h vs 23m (AMOS stage)

Profiling - try same job with different shapes

- **GP2 (mendel 16 cores) vs GP1 (sgi 8 cores)**
  - 1080s vs 1875s

- **GP2: 4, 8, 16 & 32 threads**
  - 1940, 1450, 1140, 1080 secs

- **GP1: 1, 2, 4 & 8 threads**
  - 2600s, 1875s
CPU over time
Disk I/O over time
Memory Usage
Network
$ jgi_run_mpi_in_gridengine.sh

USAGE: /jgi/tools/bin/jgi_run_mpi_in_gridengine.sh [VAR=val ...] cmd and arguments

This script will detect and/or construct the proper MPI environment within which your cmd will be executed. The script will execute within an SGE job:

mpirun cmd and arguments

You can override the default behavior by specifying any of the following, where VAR can be in the environment or command line and is any of:

FORCE_LOOSE= # Do not use GE 'tight' integration
FORCE_EtherNET= # Do not use infiniband, if available
LOGFILE= # Separate log all output
JOB_SHAPE= # bycore, bysocket or bynode
MPI_OPTS= # MPI Options to override default

Example:
/jgi/tools/bin/jgi_run_mpi_in_gridengine.sh JOB_SHAPE=bycore myMpiExecutable [args ...]
NUMA considerations

Non-Uniform Memory Access

Like 2 computers in one box, sharing resources

Threads work best when bound to a socket

Stay tuned...