

Herding millions of chickens: Programming models for Blue Gene/Q and beyond

Jeff Hammond

Leadership Computing Facility
Argonne National Laboratory

24 January 2011



*If you were plowing a field, which
would you rather use: 2 strong oxen or
1024 chickens?*

– Seymour Cray

If you were plowing a field, which would you rather use: 12,160 bipolar oxen¹ or 786,432 chickens²?

– Not Seymour Cray

¹ A bipolar oxen is one AMD Bulldozer and one NVIDIA Fermi.

² A chicken is a Blue Gene/Q core.

Heterogeneous wagon-pulling?



Unless your wagon is designed to be pulled by two different animals, pulling it with two different animals doesn't make a lot of sense. . .

Millions of chickens

Mira

- 48 racks
- 1024 nodes/rack
- 16 cores/node (16 GB/node)
- 4 hardware threads per core

$$P = 48 \times 1024 \times 16 \times 4 = 3145728.$$

Unlike a CPU+GPU node, you can chop BGQ nodes at nearly arbitrary granularity and evolve between them painlessly.

Herding chickens

There are many ways to fill the machine:

- `mpiexec -n 49152 --env OMP_NUM_THREADS=64`
- `mpiexec -n 98304 --env OMP_NUM_THREADS=32`
- `mpiexec -n 196608 --env OMP_NUM_THREADS=16`
- `mpiexec -n 393216 --env OMP_NUM_THREADS=8`
- `mpiexec -n 786432 --env OMP_NUM_THREADS=4`
- `mpiexec -n 1572864 --env OMP_NUM_THREADS=2`
- `mpiexec -n 3145728 --env OMP_NUM_THREADS=1`

This does not even scratch the surface in terms of parallel modes...

Types of parallelism

Categorizing parallel models

- **MPI-1** Private address space, explicit communication, independent execution.
- **MPI-RMA** Private-ish address space, explicit communication, independent execution.
- **OpenMP (loops)** Shared address space, implicit communication, collective execution.
- **OpenMP (sections)** Shared address space, implicit communication, semi-independent execution.
- **Pthreads** Shared address space, independent execution.
- **PGAS** Shared address space, implicit/explicit communication, independent execution.

OpenMP is always implemented on top of Pthreads.

PGAS can use MPI and/or Pthreads.

Categorizing application needs

- What does your data look like? Is it naturally private or naturally shared? Big dense linear algebra is naturally shared. Domain decomposition is naturally private.
- What is your execution model? SPMD? Task parallel?
- Are you synchronizing tasks or data? Are tasks working on independent data or shared data? Can one map trivially between independent (i.e. distributed) data representation and shared representation?

Decision making process

- MPI exists in a constructive cycle of ubiquity and optimization.
- OpenMP is reaching ubiquity; optimization is debatable.
- Pthreads is ubiquitous but invisible (system programmers only).
- PGAS portability, especially performance portability, is still emerging.

The right answer for the right reason.

– Ernest Davidson

Science first, then algorithms, then programming models.

– Some guy

Pthreads example

Function prototype:

```
int pthread_create(pthread_t *thread,  
                  const pthread_attr_t *attr,  
                  void *(*start_routine) (void *),  
                  void *arg);
```

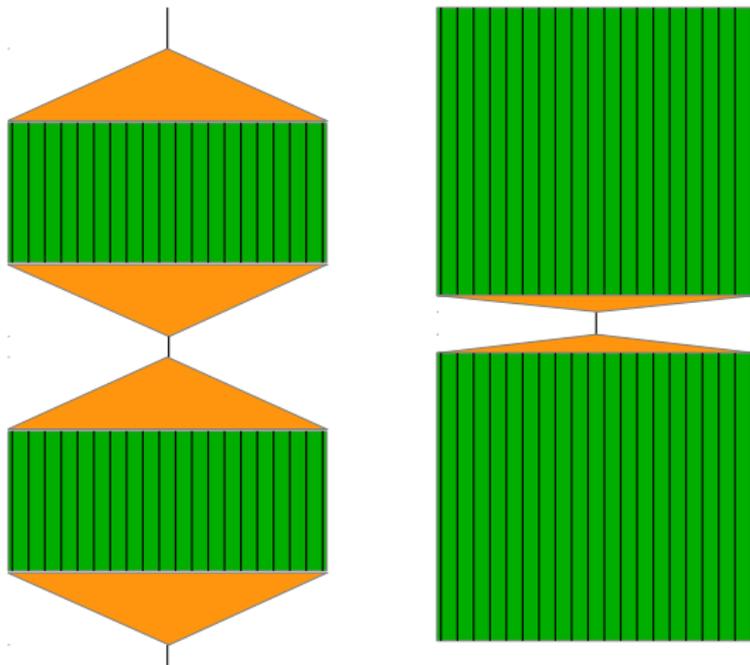
Usage:

```
void* foo(void* dummy){ /* do something */ }  
pthread_create(&my_thread, NULL, foo, NULL);  
pthread_join(my_thread, NULL);
```

Pthreads: no compiler, no problem.

Back to reality

OpenMP: from 1 to N or N to 1?



OpenMP: from 1 to N or N to 1?

```
#pragma omp parallel
{ /* thread-safe */ }

#pragma omp single
{ /* thread-unsafe */ }

#pragma omp parallel for
{ /* threaded loops */ }

#pragma omp sections
{ /* threaded work */ }
```

```
{ /* thread-unsafe */ }

#pragma omp parallel for
{ /* threaded loops */ }
{ /* thread-unsafe */ }

#pragma omp parallel for
{ /* threaded loops */ }
/* thread-unsafe work */
```

Final thoughts

MPI ranks on a node are parallel by default.
OpenMP threads on a node are serial by default.
Most MPI vs. OpenMP comparisons are bogus.

Amdahl's law is not a myth and the future is hybrid.

Making your code thread-safe is never a bad thing.