Diagnosing Network Bottlenecks: One-sided Message Contention

Nathan Tallent, Abhinav Vishnu, Huub Van Dam, Jeff Daily
Darren Kerbyson, Adolfy Hoisie

Pacific Northwest National Lab

- Network contention is poised to affect app developers
  - Growing ratio of cores to network bandwidth
  - Growing interest in one-sided programming models
  - Contention can severely increase blocking time
- Difficult to reason about one-sided message contention
  - Asynchrony can decrease or increase contention

Contributions

- Characterize contention in production application
  - Platforms: InfiniBand, Cray Aries, BlueGene/Q
    - See graphs on right
  - Bottleneck (blocking time) changes with task size and platform
  - Contention changes substantially with task size, interconnect, and data layout
- Pinpoint and quantify network contention
  - Distinguish between contention at message initiator and target
  - Low overhead (typically < 5%)
- Correct network contention
  - Change distribution of a few global arrays
  - Reduce contention by 50% (PIC)
  - Improve *total* runtime by up to 20% (PIC)

QDR InfiniBand (PIC) oversubscribed fat tree; static routing

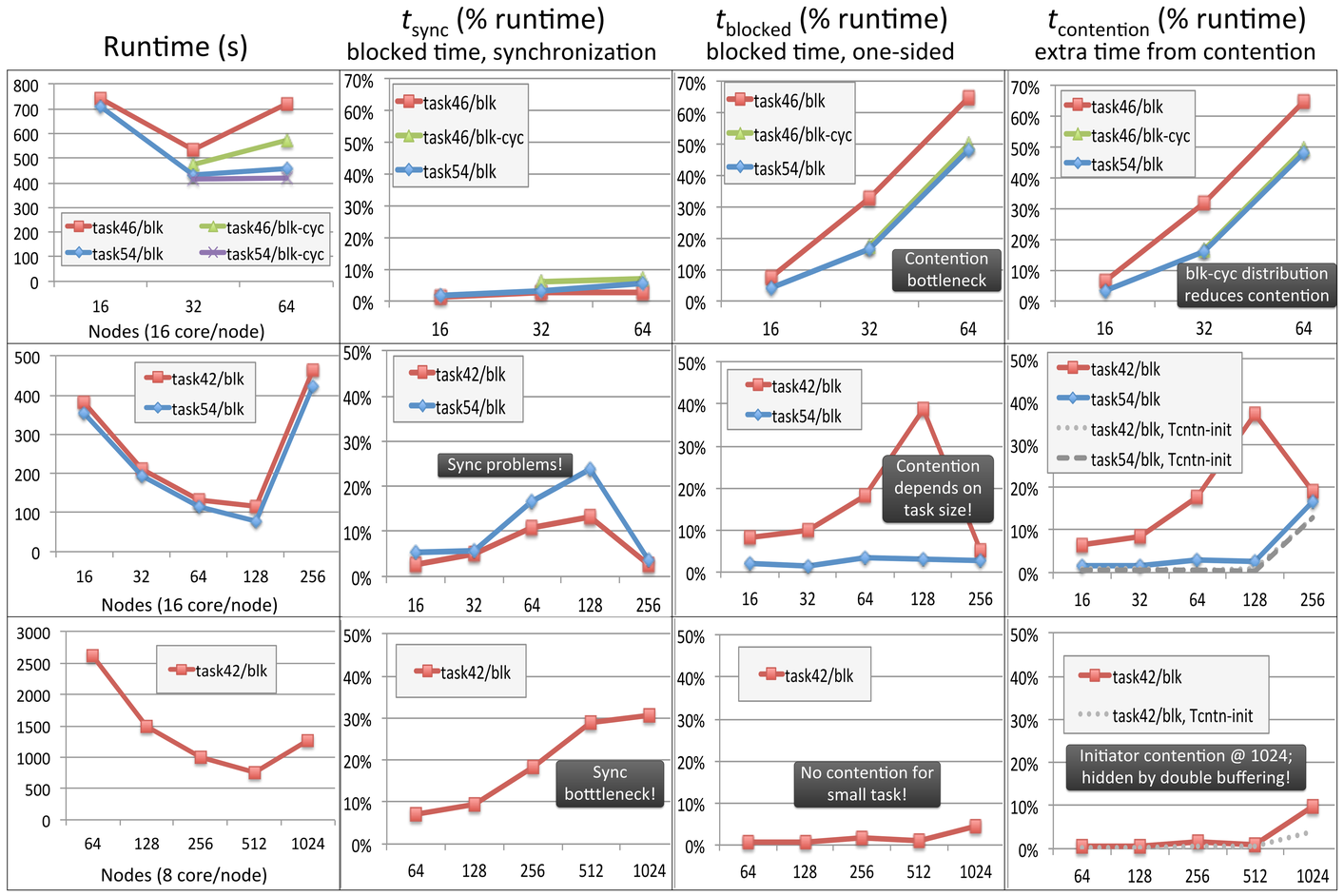
Cray Aries (Eos) Dragonfly, adaptive routing

IBM BG/Q (Mira) 5D torus; zone/ adaptive routing

Characterizing One-sided Contention in NWChem

NWChem/Carbon240:
- designed to be asynchronous
- tasks compute on distributed arrays
- avoids collectives

Severity of contention changes substantially with task size, platform, and data layout



Total measurement overhead: 1–5%

Diagnosing Contention in NWChem/Carbon240

Plugin to HPCToolkit

PIC: QDR InfiniBand (oversubscribed fat tree, static routing); task46; 1024 cores

source code

```
fock_2e_cdf.F
      call fock_get_blk(nfock, vg_dens,
     $     llo, lhi, klo, khi, dlk, tmp)
      call fock_get_blk(nfock, vg_dens,
     $     llo, lhi, ilo, ihi, dli, tmp)
      call fock_get_blk(nfock, vg_dens,
     $     llo, lhi, jlo, jhi, dlj, tmp)
```

100% of contention – 65% of execution time – due to `get`

Contention mostly at `get`'s *target* (location of data)
- 4.7e11 vs. 2.2e9 (µs)

Calling Context View | Callers View | Flat View

← call chain

	$t_{blocked}$		$t_{contention}$ (Sum/E)		$t_{contention\text{-}initiator}$		$t_{contention\text{-}target}$	
1167: ga_get_	4.78e+11	99.9%	4.76e+11	99.6%	2.32e+09	84.5%	4.74e+11	99.7%
56: fock_get_blk	4.78e+11	99.8%	4.76e+11	99.6%	2.22e+09	80.9%	4.74e+11	99.7%
89: fock_2e_cache_dens_fock	3.70e+11	77.3%	3.69e+11	77.2%	1.32e+09	48.2%	3.68e+11	77.4%
91: fock_2e_cache_dens_fock	3.61e+10	7.5%	3.58e+10	7.5%	2.81e+08	10.2%	3.55e+10	7.5%
62: fock_2e_cache_dens_fock	2.63e+10	5.5%	2.61e+10	5.5%	2.37e+08	8.6%	2.59e+10	5.4%
87: fock_2e_cache_dens_fock	1.68e+10	3.5%	1.67e+10	3.5%	1.30e+08	4.7%	1.66e+10	3.5%
71: fock_2e_cache_dens_fock	1.49e+10	3.1%	1.48e+10	3.1%	1.30e+08	4.7%	1.47e+10	3.1%
80: fock_2e_cache_dens_fock	1.36e+10	2.8%	1.35e+10	2.8%	1.22e+08	4.4%	1.34e+10	2.8%

Moving hot-spot reading data, i.e., target sending to initiator
- 2 `gets` (read) for each `acc` (write)
- `gets` go over different links than `accs`
- `gets` contend for out links when sending to initiator

Challenges

- Must characterize *demand* for network resources
  - Throughput is easy to measure; *demand* is harder
    - throughput: actual messages
    - demand: actual + pending messages
  - Helpful hardware (backpressure) often inaccessible
  - Unhelpful hardware (throughput) often accessible
- Avoid perturbing network
- Detect contention at both boundary & interior of network
  - indirect nw: difficult to detect interior traffic at switch
  - direct nw: difficult to distinguish interior and boundary traffic at NIC-router
- Ensure portability across several interconnects
- Manage asynchrony between cores and NIC's DMA engine
  - NIC controls delivery but cores control measurement

Change array distribution to improve runtime by 20% at 1024 cores.

Title

- Bullet 1
  - Bullet 2
    - Bullet 3
      - Bullet 4
        - Bullet 5

Subtitle: asdfasdf