RevoR scaling analysis

on NumaConnect

Daniel J Blueman

Numascale Asia

August 21, 2014

Introduction

- Revolution R exhibits poor scaling with increasing core count in the 1Billion benchmark
- Large-SMP systems have much higher scheduling overhead
  - efficiency loss increases significantly with non-compute bound workloads
- System under test is a four Supermicro quad-socket servers (dual-populated)
  - 128 cores of Opteron 6380 processors
  - 512GB DDR3-1600 ECC memory
  - interconnected with revision C NumaConnect adapters in 2x2 geometry
- A warm-up run was performed, ensuring data was accessed directly from pagecache

Approach

- need coarse-grain system profiler
- perf record profiles instuction-level throughput, so too narrow
- vmstat is more system-level, so ideal:

```
$ vmstat 3
r  b   swpd      free   buff     cache   si   so    bi    bo   in   cs us sy  id wa st
1  0      0 403729408  67056 120149440    0    0     0     0   21   14  0  0 100  0  0
...
1  0      0 403729408  67056 120149440    0    0     0     0   19   13  0  0 100  0  0
```

- choose interval greater than iteration/block time to average load variation
  - eg vmstat 10

Useful fields

- r is the average number of runnable processes
  - should be equal to the number of allocated cores
- b is the average number of processes waiting for I/O
  - should be low, ideally 0 (dataset in pagecache)
- in is interrupts per second
  - almost all scheduling interrupts; higher is less efficient
  - lengthen maximum scheduling quantum by running processes in batch class
- cs is context switches per second
  - they occur when making a system call or blocking
  - various overheads: TLB flush, state store/restore

Useful fields (continued)

- us is percentage of time executing application code
  - critically indicates how much compute resource the application is receiving; should be close to the percentage of cores allocated
- sy is percentage of time executing kernel code
  - key indicator of over-use of system calls, context switches or communication; expected to be $<3\%$
- id is percentage of time not consumed otherwise
- wa is percentage of time waiting on disk or network I/O
  - expected to be less than $<5\%$

Observed data

idle

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
1	0	0	403729408	67056	120149440	0	0	0	0	21	14	0	0	100	0	0

2 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
3	0	0	403376832	39152	120176928	0	0	0	0	372	58	2	0	98	0	0

4 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
5	0	0	403409760	40028	120166608	0	0	0	4	725	129	3	0	97	0	0

8 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
5	0	0	403442080	40252	120166736	0	0	0	0	1064	247	4	0	95	0	0

16 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
8	0	0	403457152	40484	120166736	0	0	0	1	2872	4054	3	1	96	0	0

32 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
10	0	0	403442144	41108	120164928	0	0	0	4	5933	7898	4	2	94	0	0

64 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
7	0	0	403390784	41704	120164968	0	0	0	0	6868	7833	3	3	94	0	0

128 cores

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa	st
6	0	0	403432384	44028	120165136	0	0	0	0	8477	11562	1	3	96	0	0

Analysis

- runnable processes (thread pool size) less than allocated cores
  - particularly as allocated cores increases
  - possibly constrained by data-dependencies or domain decomposition
  - inefficiency perhaps due to unneeded communication
  - locking scheme possibly a poor fit
  - code potentially not designed for scaling
- high rate of context switches per second
  - potential producer/consumer design issue
  - time lost waiting for locking or events
  - generates a lot of scheduling pressure
  - thus interrupts per second and system time
- instructions per clock low (measured using perf stat)
  - due to core back-end stalls (ie data fetch)
  - instruction scheduling and set not optimised for family 10h+ processors
  - loose prefetch model used due to code not being optimised for family 10h+ processors

Recommendations

- decompose problem optimally
  - break into as near compute-time uniform chunks as possible
- elide data-copy overhead
  - map data files into address space via mmap
  - each cacheline (64 bytes) copied needs a two-cycle coherency transaction
  - latency of accessing the NumaChip's coherency cache (300ns) reduces this throughput; copying is not 'free'
  - problem would be observable on a dual or quad-socket system
- elide communication
  - if inter-process, use mmap() to shared buffers
  - use dual or triple buffer of equal size to the number of cores allocated to avoid waiting

Recommendations (continued)

- mitigate cache-line bouncing
  - align hot variables (eg locks, counters) on cacheline (64 byte) boundaries
  - pack read-mostly variables together for optimal fetch and occupancy
- facilitate platform efficiency
  - use large/huge-pages (see mmap flags) to avoid page faulting
  - emit prefetch instructions
  - emit non-temporal instructions to access data not reused
  - page-faults will allocate on faulting NUMA node, which may be sub-optimal
- build optimised code optimised for family 10h Opterons
  - switch between Xeon/Opteron code at runtime depending on cpuid instruction