Slow Nodes Cost Your Users Valuable Resources. Can You Find Them?

Ricky A. Kendall
2009 Cray User Group,
Compute the Future
Atlanta Georgia
May 4, 2009

U.S. DEPARTMENT OF ENERGY

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Outline

- Motivation
- Bugget Background
- Utilization
- Results
- Future Directions

As Computational Scientists what do we assume as ground zero?

- Hardware gives correct results
  - Often true
- Hardware is performing optimally
  - Hardware is easy. Mostly true.
- Compilers produce correct, runable binaries
  - Most of the time!
- The computer is doing what I think it is doing!
  - Yeah right!

Bugget Background

- Motivation
  - Fat Nodes
    - Manage intra-node and inter-node algorithmic space
    - Hybrid OpenMP/MPI possibilities
  - Hello World for parallel computation is Matrix Multiply
    - Simple enough everyone can wrap their brain around it
    - Complex enough to expose first order problems
    - $O(N^3)$ work and $O(N^2)$ data movement!
  - Straightforward implementations
    - Fox's algorithm for parallel matrix multiply
    - Arbitrary matrix definitions

Why Arbitrary Matrices and not Random numbers?

- The result is predetermined!

$$A = ai + bj + c \qquad B = di + ej + f$$

$$C = \sum_{k=1}^{k_{limit}} A_{i,k} B_{k,j}$$

Analytical

$$= \sum_{k=1}^{k_{limit}} (ai+bk+c)(dk+ej+f)$$

How do you compute a matrix multiply?

A B →

Via Some Algorithm

- **Ddot Loops**
- **Daxpy Loops**
- **Blocked Loops**
- **Strassen's Algorithm**
- **Parallel Daxpy**
- **Cannon's Algorithm**
- **Fox's or BMR Algorithm**
- **Any Algorithm**

= C

An advantage of knowing the answer!

C - C = zero

Computed   Analytical

It worked

fatal error

If your computational algorithm worked then the computed matrix should be identical to the resultant analytical matrix!

= non zero

Process and Data Layout for Fox Algorithm

Process Mappings	Process Column 0	Process Column 1	Process Column 2
Process Row 0	0 (0,0)	1 (0,1)	2 (0,2)
Process Row 1	3 (1,0)	4 (1,1)	5 (1,2)
Process Row 2	6 (2,0)	7 (2,1)	8 (2,2)

Stage 0

	PC 0	PC 1	PC 2
PR 0	Bcast A_{00} C_{00} += $A_{00}B_{00}$ Send B_{00} up	Bcast A_{00} C_{01}+=$A_{00}B_{01}$ Send B_{01} up	Bcast A_{00} C_{02}+=$A_{00}B_{02}$ Send B_{02} up
PR 1	Bcast A_{11} C_{10}+=$A_{11}B_{10}$ Send B_{10} up	Bcast A_{11} C_{11}+=$A_{11}B_{11}$ Send B_{11} up	Bcast A_{11} C_{12}+=$A_{11}B_{12}$ Send B_{12} up
PR 2	Bcast A_{22} C_{20}+=$A_{22}B_{20}$ Send B_{20} up	Bcast A_{22} C_{21}+=$A_{22}B_{21}$ Send B_{21} up	Bcast A_{22} C_{22}+=$A_{22}B_{22}$ Send B_{22} up

Stage 1

	PC 0	PC 1	PC 2
PR 0	Bcast A_{01} C_{00} += $A_{01}B_{10}$ Send B_{10} up	Bcast A_{01} C_{01}+=$A_{01}B_{11}$ Send B_{11} up	Bcast A_{01} C_{02}+=$A_{01}B_{12}$ Send B_{12} up
PR 1	Bcast A_{12} C_{10}+=$A_{12}B_{20}$ Send B_{20} up	Bcast A_{12} C_{11}+=$A_{12}B_{21}$ Send B_{21} up	Bcast A_{12} C_{12}+=$A_{12}B_{22}$ Send B_{22} up
PR 2	Bcast A_{20} C_{20}+=$A_{20}B_{00}$ Send B_{00} up	Bcast A_{20} C_{21}+=$A_{20}B_{01}$ Send B_{01} up	Bcast A_{20} C_{22}+=$A_{20}B_{02}$ Send B_{02} up

Stage 2

	PC 0	PC 1	PC 2
PR 0	Bcast A_{02} C_{00} += $A_{02}B_{20}$ Send B_{20} up	Bcast A_{02} C_{01}+=$A_{02}B_{21}$ Send B_{21} up	Bcast A_{02} C_{02}+=$A_{02}B_{22}$ Send B_{22} up
PR 1	Bcast A_{10} C_{10}+=$A_{10}B_{00}$ Send B_{00} up	Bcast A_{10} C_{11}+=$A_{10}B_{01}$ Send B_{01} up	Bcast A_{10} C_{12}+=$A_{10}B_{02}$ Send B_{02} up
PR 2	Bcast A_{21} C_{20}+=$A_{21}B_{10}$ Send B_{10} up	Bcast A_{21} C_{21}+=$A_{21}B_{11}$ Send B_{11} up	Bcast A_{21} C_{22}+=$A_{21}B_{12}$ Send B_{12} up

Algorithmic Choices for serial matrix multiply in Bugget

- 3 simple loops (DDOT) in C or Fortran
- DAXPY Rearrangement in C or Fortran
- Blocked versions of DDOT/DAXPY in C or Fortran
- OpenMP parallel versions of the above
- A call to dgemm

- 17 algorithms in all!

A run using the parallel mode of Bugget

aprun -n 400 -N 8 bugget q 10 r 59000 a dgemm
Bugget MPI Version 2
rank = 59000
algorithm is 9: DGEMM
group nproc = 100 (q=10)
total number of processors allocated = 400
4 groups of size 100
Number of groups: 4
Per-processes over all groups of any size:
Minimum Memory: 1593.48 MB = 1.56 GB
Average Memory: 1593.48 MB = 1.56 GB
Maximum Memory: 1593.48 MB = 1.56 GB
Minimum Patch Rank: 5900
Maximum Patch Rank: 5900
Rank 0 is on nid00163:c0-1c2s0n3 (core affinity = 0)
Rank 1 is on nid00163:c0-1c2s0n3 (core affinity = 1)
Rank 2 is on nid00163:c0-1c2s0n3 (core affinity = 2)
Rank 3 is on nid00163:c0-1c2s0n3 (core affinity = 3)

A run using the parallel mode of Bugget (2)

For Groups that have more than one process

Group 0 Statistics (group size:100)
Min fox Time: 532.187 Min Norm: 4.489e-12
Ave fox Time: 535.956 Ave Norm: 9.382e-12
Max fox Time: 539.536 Max Norm: 1.391e-11
STD DEV Time: 3.228e-01 STD DEV : 4.893e-13

Group 1 Statistics (group size:100)
Min fox Time: 533.196 Min Norm: 4.489e-12
Ave fox Time: 535.953 Ave Norm: 9.382e-12
Max fox Time: 538.411 Max Norm: 1.391e-11
STD DEV Time: 1.487e-01 STD DEV : 4.893e-13

Group 2 Statistics (group size:100)
Min fox Time: 533.072 Min Norm: 4.489e-12
Ave fox Time: 536.238 Ave Norm: 9.382e-12
Max fox Time: 538.365 Max Norm: 1.391e-11
STD DEV Time: 6.824e-02 STD DEV : 4.893e-13

A run using the parallel mode of Bugget (3)

Stats over all procs in a group of size: 100
(100:400:r=59000:p=9.2GF:DGEMM)
Min fox Time: 532.19 (-2.6) GF(7.7) PP(83.9)
Ave fox Time: 536.05 (0.0) GF(7.7) PP(83.3)
Max fox Time: 539.54 (2.4) GF(7.6) PP(82.8)
Max-Min Time: 7.35 (5.0)
STD DEV Time: 1.460e+00
Min Norm: 4.489e-12 (-2.3)
Ave Norm: 9.382e-12 (0.0)
Max Norm: 1.391e-11 (2.1)
STD DEV : 2.150e-12
out 127: 273 of 400 w/in 1.0-sig (68.2%)
out 20: 380 of 400 w/in 2.0-sig (95.0%)
out 0: 400 of 400 w/in 3.0-sig (100.0%)
Sigma Threshold: 4.2 sec
Time Threshold: 0.333 sec

Total reported bad sigma nodes: 0 of 400 nodes tested.
Global time: 545.683 seconds

Bugget as a system Diagnostic

- Several user codes were seeing poor performance
  - Varied 10%-15% depending on which nodes they ran on
  - Bugget showed the same behavior
- Bugget could tune the run time for the “test”
  - Input parameters are command line
    - Rank,
    - perfect square size,
    - algorithmic choice
  - No large input decks needed.
  - Reliable identification of slow nodes.

Bugget run in SNF mode "Slow Node Finding mode"

```
aprun -n 1408 -S 1 bugget -f
 Bugget MPI Version 2
rank = 3565
algorithm is 2: DAXPY
group nproc = 1 (q=1)
 total number of processors allocated = 1408
1408 groups of size 1
Number of groups: 1408
Per-processes over all groups of any size:
Minimum Memory:     581.78 MB =       0.57 GB
Average Memory:     581.78 MB =       0.57 GB
Maximum Memory:     581.78 MB =       0.57 GB
Minimum Patch Rank:       3565
Maximum Patch Rank:       3565

Rank 0 is on nid00032:c0-0c1s0n0          (core affinity = 0)
Rank 1 is on nid00032:c0-0c1s0n0          (core affinity = 4)
Rank 2 is on nid00033:c0-0c1s0n1          (core affinity = 0)
Rank 3 is on nid00033:c0-0c1s0n1          (core affinity = 4)
```

Bugget run in SNF mode
“Slow Node Finding mode”

Stats over all procs in a group of size: 1
(1:1408:r=3565:p=9.2GF:DAXPY)
Min fox Time: 102.173 (-0.8) GF(0.9) PP(9.6) Min Norm: 1.052e-12 (0.0)
Ave fox Time: 103.424 (0.0) GF(0.9) PP(9.5) Ave Norm: 1.052e-12 (0.0)
Max fox Time: 120.982 (11.6) GF(0.7) PP(8.1) Max Norm: 1.052e-12 (0.0)
Max-Min Time: 18.809 (12.4)
STD DEV Time: 1.516e+00 STD DEV : 0.000e+00
out 20: 1388 of 1408 w/in 1.0-sig (98.6%)
out 20: 1388 of 1408 w/in 2.0-sig (98.6%)
out 10: 1398 of 1408 w/in 3.0-sig (99.3%)
out 10: 1398 of 1408 w/in 4.0-sig (99.3%)
out 10: 1398 of 1408 w/in 5.0-sig (99.3%)
out 10: 1398 of 1408 w/in 6.0-sig (99.3%)
out 10: 1398 of 1408 w/in 7.0-sig (99.3%)
out 10: 1398 of 1408 w/in 8.0-sig (99.3%)
out 10: 1398 of 1408 w/in 9.0-sig (99.3%)
out 10: 1398 of 1408 w/in 10.0-sig (99.3%)
out 8: 1400 of 1408 w/in 11.0-sig (99.4%)
out 0: 1408 of 1408 w/in 12.0-sig (100.0%)

Bugget run in SNF mode
"Slow Node Finding mode"

rank 762 time: 120.597 Sigma: 11.329 Delta Time: 17.173 sec
rank 762 Node: nid00413:c0-4c0s7n1

rank 763 time: 107.718 Sigma: 2.833 Delta Time: 4.294 sec
rank 763 Node: nid00413:c0-4c0s7n1

rank 764 time: 120.982 Sigma: 11.583 Delta Time: 17.557 sec
rank 764 Node: nid00414:c0-4c0s7n2

rank 765 time: 107.609 Sigma: 2.760 Delta Time: 4.184 sec
rank 765 Node: nid00414:c0-4c0s7n2

rank 766 time: 120.963 Sigma: 11.571 Delta Time: 17.539 sec
rank 766 Node: nid00415:c0-4c0s7n3

rank 767 time: 107.577 Sigma: 2.739 Delta Time: 4.152 sec
rank 767 Node: nid00415:c0-4c0s7n3

Total reported bad sigma nodes: 20 of 1408 nodes tested.

What causes the slow nodes?

- Bad Hardware
  - Memory chips with too many single bit errors
  - Memory controller not running at the right frequency
    - Really bad hardware!
- How do you track this?
  - System wide diagnostic runs
    - Over "groups" of processors to avoid scheduling bottleneck
    - Can be tuned to perceived need.
      - Daily, weekly, after a reboot
    - Can't just submit it and forget it!
  - User's can check their own set of nodes
    - % module load bugget
    - % bugget_my_nodes
      - Return nonzero status if there are any bad ones!

Future Directions

- Provide Pthread implementations of the serial matrix multiply kernel
  – Cover the “used” programming model space
- Link to “threaded” optimized libraries
  – Cover the library spaced used by the community
- Develop histogram type data for all timings collected
- Collect both compute and communication timings
- Provide a tool to generate a usable node list based on bugget data.
  – Users can decide to run the job on the reduced set or abort!

Questions?