Performance experiments on BOUT++

Praveen Narayanan and Alice Koniges
Why Analyze Performance?

• Improving performance on HPC systems has compelling economic and scientific rationales.
  – Dave Bailey: Value of improving performance of a single application, 5% of machine’s cycles by 20% over 10 years: $1,500,000
  – Scientific benefit probably much higher

• Goal: solve problems faster; solve larger problems

• Accurately state computational need

• Only that which can be measured can be improved

• The challenge is mapping the application to an increasingly more complex system architecture
  – or set of architectures
Performance Analysis Issues

- Difficult process for real codes
- Many ways of measuring, reporting
- Very broad space: Not just time on one size
  - for fixed size problem (same memory per processor): Strong Scaling
  - scaled up problem (fixed execution time): Weak Scaling
- A variety of pitfalls abound
  - Must compare parallel performance to best *uniprocessor* algorithm, not just parallel program on 1 processor (unless it’s best)
  - Be careful relying on any single number
- Amdahl’s Law
Performance Questions

• How can we tell if a program is performing well?

• Or isn’t?

• If performance is not “good”, how can we pinpoint why?

• How can we identify the causes?

• What can we do about it?
The multicore era

- Moore’s law still extant
- Traditional sources of performance improvement ending
  - Old trend: double clock frequency every 18 months
  - New trend double #cores every 18 months
  - Implications: flops cheap, communication, network bandwidth expensive in future
Processor-DRAM Gap (latency)

- Memory hierarchies are getting deeper
  - Processors get faster more quickly than memory

"Moore’s Law"

Processor-Memory Performance Gap:
(grows 50% / year)

μProc 60%/yr.

DRAM 7%/yr.

Time

Slide from Katherine Yellick’s ppt
CS-267
Performance analysis tools

- Use of profilers to measure performance
- Approach
  - Build and instrument code with binary to monitor function groups of interest
    - MPI, OpenMP, PGAS, HWC, IO
  - Run instrumented binary
  - Identify performance bottlenecks
  - Suggest (and possibly implement) improvements to optimize code
- Tools used
  - IPM: Low overhead, communication, flops, code regions
  - CrayPAT: Communication, flops, code regions, PGAS, variety of tracing options
- Overhead depends on number of trace groups monitored
- Level of detail in study depends on specifics: time available, difficulties presented by code
Performance checklist

- Scaling
  - Application time, speed (flop/s)
  - Double concurrency, does speed double?
- Communication (vs computation)
- Load imbalance
  - Check cabinet for mammoths and mosquitoes
- Size and composition of communication
  - Bandwidth bound?
  - Latency bound?
  - Collectives (large concurrency)
- Memory bandwidth sensitivity
BOUT++ scaling results (Elm_pb)

Grid used:
- nx = 2052
- ny = 512

Set: nxpe = 256

Does not scale
But ...
Communication
Does not increase

Runtime increases because of other reasons.
Performance decreases because of other reasons.
• Separate out communication portion from walltime and compute speed=Flop/(computation time-core)

• Should be constant for ideal scaling, if comm were reason for performance degradation
MPI pie shows significance of Allreduce calls

- MPI_Allreduce calls form bulk of pie
- MPI average message size 5 kB, 74,000 calls
- Not quite entirely latency bound on hopper (5 kB should be large enough)
- Might become bottleneck after other issues are sorted out
- (communication not yet a bottleneck)
Bout++: break up times in each kernel to check how they scale

- Breakup by time spent: Calc scales somewhat, but inv, solver do not scale
• Up to concurrency 8,192, code scales nearly perfectly.

• Two issues beyond 8,192
  – Performance decreases (flop/time decreases)
  – Wall time increases
  – MPI not the reason for performance degradation
  – Computational performance decreases
Issues with increasing flop count

- Steady increase in flop count (number of operations)
  Conjecture: Extra computations in ghost cells (and more cycles spent in doing these)
  Valid region (excluding ghost region) does same amount of work

Grid points/proc
Decreases with concurrency

Increaseasing concurrency
• Experiment: turbulence in an annulus (LAPD)
• Investigate source of extra computations in code [more work done – leads to greater flop’s (not ‘flop/s’) count]
• Code annotated to give flop count with CrayPAT
• A given portion is annotated and flop count in that code section is compared across concurrency
• Conjecture: increase in flops in this section because of ghost cell computations (arrays consist of valid+ghost regions)
\texttt{PAT\_region\_begin(24, "phys\_run-14");}

\begin{verbatim}
nu = nu\_hat * Nit / (Tet^{1.5});
mu_i = mui\_hat * (Tit^{2.5})/Nit;
\end{verbatim}

\begin{verbatim}
kapa\_Te = 3.2*(1./fmei)*(wci/nueix)*(Tet^{2.5});
kapa\_Ti = 3.9*(wci/nuiix)*(Tit^{2.5});
\end{verbatim}

\begin{verbatim}
// Calculate pressures
pei = (Tet+Tit)*Nit;
pe = Tet*Nit;
\end{verbatim}

\texttt{PAT\_region\_end(24);}

- Quantities Tet, Tit, Nit, etc declared as follows

\begin{verbatim}
// 3D evolving fields
Field3D rho, ni, ajpar, te;

// Derived 3D variables
Field3D phi, Apar, Ve, jpar;

// Non-linear coefficients
Field3D nu, mu_i, kapa\_Te, kapa\_Ti;

// 3D total values
Field3D Nit, Tit, Tet, Vit, phit, VTet, ajp0;

// pressures
Field3D pei, pe;
Field2D pei0, pe0;
\end{verbatim}

Variables defined to comprise valid region + ghost cells

Extra Computation in box can be measured
Computation in guard cells

- Grid: 204x128, ghost cells: MXG, MYG=2

6400 (extreme-2 inner grid pts)

Extreme case 3 times the work as expected!

3200 (extreme but one)

Twice as much work as expected
Observations: validation of ghost cell conjecture

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>FPO in given region from CrayPat</th>
<th>Factor predicted</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>6400</td>
<td>51512160000</td>
<td>1(reference case)</td>
<td>--</td>
</tr>
<tr>
<td>3200</td>
<td>34341360000</td>
<td>2/3</td>
<td>2/3</td>
</tr>
<tr>
<td>1600</td>
<td>25755960000</td>
<td>½</td>
<td>½</td>
</tr>
<tr>
<td>800</td>
<td>21463260000</td>
<td>1.25/3</td>
<td>1.25/3</td>
</tr>
</tbody>
</table>

Predicted values match exactly with computed flops, in terms of ratios
Hypothesis seems to be correct
Kernels: Calc scales, affected by ghost cells

• Breakup by time spent

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>wall</th>
<th>Calc</th>
<th>Inv</th>
<th>Comm</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>364</td>
<td>298</td>
<td>24</td>
<td>23</td>
<td>19</td>
</tr>
<tr>
<td>8192</td>
<td>201</td>
<td>151</td>
<td>19</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>16384</td>
<td>124</td>
<td>74</td>
<td>16</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>16384</td>
<td>197</td>
<td>136</td>
<td>17</td>
<td>30</td>
<td>13</td>
</tr>
</tbody>
</table>

16384 (1st expt) 16384 (2nd expt)

Does not scale

Scales
BOUT++ results: improve INV, PVODE

• **Summary**
  – Scaling degrades beyond 8192 procs
  – Scaling efficiency is very poor at 32768 procs
  – Issues
    • Extra computations in ghost cells
      » Put in place to reduce communication
      » Need to check if extra computations performed are worth it
      » Performance degrades because
        • Laplacian inversion does not scale
        • Pnode solver does not scale
        • MPI collectives
    • Surface to volume ratio tested may not be best but issues remain
  – MPI: Collectives grow with concurrency, but Laplacian inversion and PVODE solver seem to be culpable in equal measure
• Recommendations: need to check if replacing ghost cell computation with communication improves runtime (or not)
• **Need to improve Laplacian inversion and PVODE kernels**
Investigation of collectives in time-stepping algorithms (PETSc)

- Time-stepping algorithms suspected to have collectives
  - First step: check growth of collectives in time-steppers
  - Hook with PETSc and turn on profiling layer
    - \texttt{-log\_summary}
  - Examine \%collectives vis a vis runtime
Overall conclusions and future directions(?)

• BOUT++ scales remarkably well for a strong scaling test
  – Performance degradation ‘not just’ because of increase in surface to volume ratio
• Communication increases at higher concurrency, could constitute the ultimate scaling bottleneck
• Extra ghost cell computations
  – Put in there for a reason, to lessen communication, but manifests as extra time spent in computation
  – Might be good overhead when flops become cheaper
• Bandwidth sensitivity not an issue in BOUT++
• Two dimensional domain decomposition
  – Possibly add OpenMP in third direction?
• Collectives might play dominant role in time-steppers
  – Find ways of minimizing this
  – What is the effect of putting in preconditioners?