Performance Measurement and Analysis of Real Programs: Case Study of a Large-Scale Atmosphere Simulation

by Peter Strazdins, Computer Systems Group

Overview:

● running time of real programs and parallel computing

● the Met Office Unified Model for weather and climate simulations
  ■ initiatives in Australia: BoM and ACCESS
  ■ overview of the software, including internal profiler module

● preliminary experiences with the UM_N320L70 benchmark
  ■ variability analysis

● profiling methodology

● scalability analysis & validation of methodology

● load imbalance and affinity issues

● conclusions
Running Time of Real Programs

- on a *real* computer, the running time of a simple computation like computing $n = \text{length } xs$ could be modelled by $t(n) = a_0 + a_1n$
  - computers have H/W support for measuring time (≡ no. of clock cycles) and other events; access these in a program via system calls
  - can determine by experiment: $a_0 = t(0)$, $a_1 = (t(1000) - a_0)/1000$
  - in practice, this might not always give an accurate prediction of $t(n)$. Why?

- large-scale simulations (e.g. $10^7$ GFLOPs) on huge data sets (e.g. $10^2$ GB) is an increasingly important use of computers
  - typically, very large and complex software is required
  - what kind of computer can run such a job in reasonable time?
  - writing efficient software is important - but first, how do we (accurately and efficiently) measure and understand its performance characteristics?
Parallel Processing with Clusters of Multicore Processors

- with $10^3$ modern processing units (cores), we can run an application with $10^7$ GFLOPs in around 2 hours (assuming 1 GFLOPs/sec processing speed)
  - $\approx 200$ memory systems of 1 GB could hold a $10^2$ GB data set
  - e.g. Intel Nehalem computing ‘node’ has $4 \times 2$ cores;
    100 such nodes connected by a (high-speed) network would suffice
- accessing large amounts of data presents a particular challenge: hence the memory hierarchy in each node

- the above assumes we can divide up the whole calculation evenly over all cores with no overhead!

- parallel processing however presents 3 main difficulties
  - load imbalance: hard to divide a computation evenly across each core
  - overhead: cores have to spend time communicating data to each other
  - start-up effects: certain parts of the calculation cannot be parallelized

- concept of parallel speedup: if $T(p)$ is execution time using $p$ cores, we would ideally like a speedup of $S(p) = T(1)/T(p) \approx p$ (called ‘good scalability’)
The Unified Model in Aust. Weather and Climate Simulations

- the Met Office Unified Model (MetUM, or just UM) is a (global) atmospheric model developed by the UK Met Office from early ’90s

- for weather, BoM currently uses a N144L50 atmosphere grid

  - wish to scale up to a N320L70 ($640 \times 481 \times 70$) then a N512L70 ($1024 \times 769 \times 70$) grid
  - operational target: 24 hr simulation in 500s on < 1K cores (10-day ‘ensemble’ forecasts)
  - doubling the grid resolution increases ‘skill’ but is $\leq 8 \times$ the work!

- climate simulations currently use a N96L38 ($192 \times 145 \times 3$) grid

  - ACCESS project to run many (long) runs for IPCC 2011
    - common infrastructure: atmosphere: UM (96 cores);
    - ocean: NEMO, sea ice: CICE, coupler: OASIS (25 cores)
  - next-generation medium-term models to use N216L85 then N320L70

- note: (warped) ‘cylindrical’ grids are easier to code but problematic …
The MetOffice Unified Model

- configuration via UMUI tool creates a directory with (conditionally-compiled) source codes + data files (for a particular grid)
- main input file is a ‘dump’ of initial atmospheric state (1.5GB for N320L70)
- ‘namelist’ files for ≈ 1000 run-time settable parameters
- in operational runs, periodically records statistics via the STASH sub-system
- partition evenly the EW & NS dimensions of the atmosphere grid on a $P \times Q$ process grid
Unified Model Code Structure and Internal Profiler

- codes in Fortran-90 (mostly F77; \(\approx 900\) KLOC) with \texttt{cpp} (include common blocks, commonly used parameter sub-lists, etc)
- main routine \texttt{u_model()}, reads dump file & repeatedly calls \texttt{atm_step()}
  - dominated by Helmholtz \(P-T\) solver (GCR on a tridiagonal linear system)
- internal profiling module can be activated via ‘namelist’ parameters
  - has ‘non-inclusive’ + ‘inclusive’ timers (\(\approx 100\) of each)
  - the top-level non-inclusive timer is for \texttt{u_model()}; sum of all non-inclusive timers is time for \texttt{u_model()}
  - reports number of calls and totals across all processes, e.g.

<table>
<thead>
<tr>
<th>ROUTINE</th>
<th>MEAN</th>
<th>MEDIAN</th>
<th>SD</th>
<th>% of mean</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_Helmholtz</td>
<td>206.97</td>
<td>206.98</td>
<td>0.05</td>
<td>0.02%</td>
<td>207.02</td>
</tr>
<tr>
<td>ATM_STEP</td>
<td>36.39</td>
<td>38.53</td>
<td>9.46</td>
<td>25.99%</td>
<td>44.60</td>
</tr>
<tr>
<td>SL_Thermo</td>
<td>25.38</td>
<td>26.60</td>
<td>3.45</td>
<td>13.58%</td>
<td>31.15</td>
</tr>
<tr>
<td>READDUMP</td>
<td>24.18</td>
<td>24.36</td>
<td>1.12</td>
<td>4.62%</td>
<td>24.37</td>
</tr>
</tbody>
</table>

- due to global sync. when a timer starts, can estimate load imbalance
The Vayu Cluster at the NCI National Facility

- 1492 nodes: two quad-core 2.93 GHz X5570 quad-core Nehalems (commissioned Mar 2010)
- memory hierarchy: 32KB (per core) / 256KB (per core) / 8MB (per socket); 24 GB RAM
- single plane QDR Infiniband: latency of $2.0\mu s$
  latency & 2600 MB/s (uni-) bandwidth per node
- jobs (parallel) I/O via Lustre filesystem
- jobs submitted via locally modified PBS; (by default) allocates 8 consecutively numbered MPI processes to each node
  - typical snapshot:
    - 1216 running jobs (465 suspended), 280 queued jobs, 11776 cpus in use
  - estimating time and memory resources accurately is important!
- the UM profiling project was (in 2010) allocated a few thousand CPU hours, max. core count 2048
Preliminary Experiences on the N320L70 Benchmark

- STASH output was disabled (to simplify)
- memory-related difficulties in running on < 8 or > 1536 cores, and on process grids like 60 × 24
- could not run benchmark under performance profiling tools like Sunstudio collect (memory?); had to rely on internal UM profiler
- variability in repeated run times of ≤ 50% (bimodal; ‘fast’ runs ≤ 10%)
- preliminary subroutine profiling revealed ‘inverse’ scaling on:
  - read_dump() (25s plus)
  - q_pos_ctl(): many ‘gather’ and ‘scatter’ communications
  - ⇒ Met Office subsequently supplied the PS24 patches
Preliminary Experiences: Inter-Iteration Variability

- Inter-iteration variability within `atm_step()`
  - 1st step took $2-5 \times$ longer; every 6th step after slightly longer
  - Partially alleviated by ‘flushing’ (entire!) physical memory 1st
  - Much was in `q_pos_ctl()`; mostly due to setting up ‘connections’

![Graph showing atm_step() times and GCR iterations](image)
Profiling Methodology

- based on these experiences, the following principles are desired:
  - infrastructure should have minimal effect on runtime (or memory)
  - consume minimal resources to accurately predict long-term sim.
  - take into account all variabilities (as far as possible)

- resulting in the following methodology:
  - take at least 5 timings for each configuration (use min., or avg.?)
  - when varying parameters, run each config. once before next rep.
  - run for the minimal number of timesteps for accurate projections (chose 3 hours)
    - need to profile the ‘warmed period’ (hours 2–3) separately
  - reduce the overhead of profiling, and measure (or estimate) its extent
    - use 1st hour to determine which timers were ‘important’ and only do global syncs for these (hard!)
  - time each iteration of \texttt{atm\_step()} for later analysis
Scalability Analysis: Which Time to Take? (N512L70 + PS24)

- process grids aspects between 1:1 and 1:2 chosen
- ‘t16’ is time for 16 cores
- essentially linear scaling from 16 to 64 cores (slightly super-)
- surprisingly, average & minimum times show similar curves

- job time @ 1024 cores includes: pre-launch: 2s, launch processes: 4s, read ‘namelist’ files: 6s, read_dump(): 27s, cleanup: 1s
Results: Scalability of the N320/N512 Benchmarks with/out PS24

- ‘t16’ is time for 16 cores
  (N512L70 ≈ 4× more – uses the same timestep)
  - really eats up our Vayu quotas!
- N512L70 scaled better
  (1.6× higher volume:surface)
- bus errors occurred for non-PS24 for > 1024 cores
- PS24 also scaled better
## Individual Subroutine Scalability (N512L70, 1536 cores)

<table>
<thead>
<tr>
<th>function</th>
<th>UM7.5</th>
<th></th>
<th>UM7.5+PS24</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$s$</td>
<td>$% t_w$</td>
<td>$% \text{im.}$</td>
<td>$s$</td>
</tr>
<tr>
<td>PE_Helmholtz</td>
<td>59.4</td>
<td>51</td>
<td>0</td>
<td>61.6</td>
</tr>
<tr>
<td>atm_step</td>
<td>15.1</td>
<td>11</td>
<td>27</td>
<td>11.4</td>
</tr>
<tr>
<td>q_pos_ctl</td>
<td>0.8</td>
<td>12</td>
<td>0</td>
<td>5.4</td>
</tr>
<tr>
<td>SL_Full_wind</td>
<td>50.8</td>
<td>10</td>
<td>47</td>
<td>49.1</td>
</tr>
<tr>
<td>SL_Thermo</td>
<td>52.7</td>
<td>6</td>
<td>20</td>
<td>53.3</td>
</tr>
<tr>
<td>Convect</td>
<td>28.0</td>
<td>9</td>
<td>54</td>
<td>66.2</td>
</tr>
<tr>
<td>SF_EXPL</td>
<td>6.7</td>
<td>6</td>
<td>69</td>
<td>35.4</td>
</tr>
<tr>
<td>Atmos_Physics2</td>
<td>12.9</td>
<td>5</td>
<td>71</td>
<td>52.9</td>
</tr>
<tr>
<td><strong>total:</strong></td>
<td>52.1</td>
<td>100</td>
<td>17</td>
<td>55.8</td>
</tr>
</tbody>
</table>

- $s$ is the speedup (over 16 cores), $% t_w$ is % of warmed period time,
- % im. is fraction of time due to load imbalance
- none scale perfectly; scope for worthwhile optimization in all
- PS24 patches have solved $q\_pos\_ctl()$ problems
- the (underestimated) load imbalance is significant in most
Load Imbalance Across Cores (16 × 24 process grid)

- large value indicates a lighter load
- clear indication of latitudinal and longitudinal variations
- difficult to avoid with fixed, regular data distribution
- over ‘warmed period’; ‘0’ = 0%, ‘9’ = 15% of run-time

from time spent in barriers within UM internal profiler
we can enforce process affinity (each of job’s process is ‘pinned’ to a core on a node) and NUMA affinity (each process only access memory on nearest DRAM module)

note differing values for t16!

on X5570, local:remote memory access is 65:105 cycles

indicates a significant amount of L3$ misses
a performance analysis tool (IPM) gives a breakup of time

without affinity, groups of 4 processes show spikes in computation times

i.e. always on the 1st ‘socket’ of a core!
Effect of Affinity on Variability

- use the normalized error from the average \[ \frac{\sum_{i=1}^{n} |t_i - \bar{t}|}{nt} \]
- noting number of measurements (\( n = 5 \)) only sufficient to observe general trends
  - no clear correlation of error and number of cores
  - process affinity reduces variability by 20%
  - NUMA affinity reduces this further by a factor of 4!

![Normalized Error vs. Number of Cores](image)

(for ‘warmed time’ of PS24/N512L70)
Evaluation of Profiling Methodology

- projected run time for 24hr operational job (960 cores) is
  \[ t' = (t - t_{2:24}) + 11.5t_{2:3} \]

<table>
<thead>
<tr>
<th>run</th>
<th>( t )</th>
<th>( t_{2:24} )</th>
<th>( t_{2:3} )</th>
<th>( t' )</th>
<th>anomalies</th>
</tr>
</thead>
<tbody>
<tr>
<td>N512L70</td>
<td>527</td>
<td>39.12</td>
<td>6.5</td>
<td>524.3</td>
<td>—</td>
</tr>
<tr>
<td>N320L701</td>
<td>224</td>
<td>163.8</td>
<td>13.7</td>
<td>214.8</td>
<td>iter. 59 (7.9s)</td>
</tr>
<tr>
<td>N320L702</td>
<td>237</td>
<td>174.9</td>
<td>14.9</td>
<td>233.6</td>
<td>iter. 134 (4.2s)</td>
</tr>
</tbody>
</table>

- defensive programming check: sum of ‘non-inclusive’ timers matched total to less than 0.1%
- methodology reduced number of barriers within the profiler by factor of 10
- measured profiler overhead (gather data, barriers) of < 1% of ‘warmed period’ times
  - overhead of printing output needs further investigation for large core counts
Conclusions

● working with such codes and systems is hard!
  ■ ‘bleeding edge’ technology, variability effects, pushing memory limits, cumbersome and huge legacy code systems . . .

● an efficient but accurate performance analysis methodology was developed
  ■ does need to be lightweight for accuracy (and also not to crash)
  ■ use of internal profiler gave useful information
  ■ scalability was reasonable (with PS24 patch), provided an appropriate timing was chosen
  ■ plenty of scope for worthwhile optimization, but only by the hard way!
  ■ load imbalance issues are particularly significant for the UM (may need a major overhaul!)

● significant effect of process and NUMA affinity on performance and its variability

● from this year on, a UM will consume a huge amount of compute cycles in Australia & elsewhere
  ■ and by people who don’t (want to) know about performance!