Performance Measures for RELAP5-3D Version 4.0.3

Dr. George L Mesina

INL/CON-12-27489

RELAP5 International Users Seminar

Oct 23-24, 2012

www.inl.gov

Idaho National Laboratory

Outline

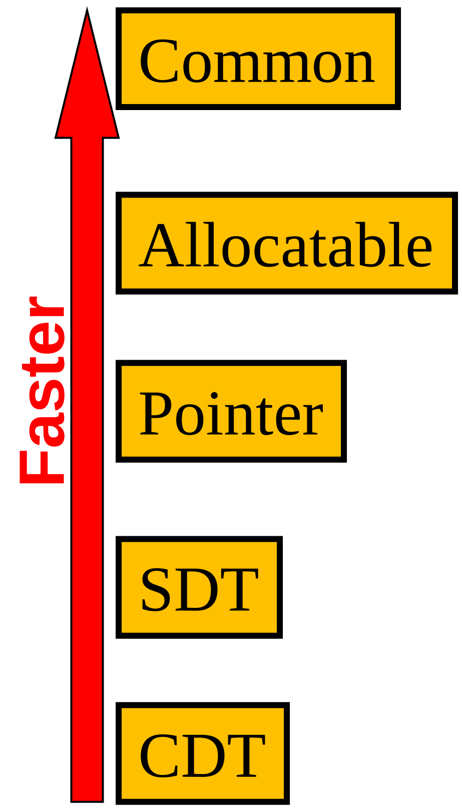
- Architectural change impacts timings
- Timing comparisons 4.0.3 vs. 2.4.3
- Detailed study of timings
- Runtime Improvements going forward

Architectural change impacts timings

- Version 2.4 database: mostly a single array (FA) in common storage
- Version 4.0.3 database: memory in many modules with allocatable and pointer arrays and derived types
- Speed of access
  - Common blocks have the fastest memory access
    - Location is fixed at beginning of run
  - Allocatable memory is slower
    - Location and length unfixed until allocated
    - Extra overhead to access

Speed of Accessing Data

- Pointer arrays are slower
  - Location and length unfixed until allocated
  - Pointer overhead (pointing, nullity issues)
  - Access to allocatable/pointer array adds overhead
- Simple Derived Types (SDT) = mix of fixed length basic data types
  - DT access slightly slower than a basic data array
    - Fix-length SDT vs fix-length basic close in access time
    - Same for Allocatable SDT and pointer SDT
- Complex Derived Types (CDT) has sub-derived-types
  - Overhead involved to access each sub-level(s)



Coding change impacts timings

- Coding changes give and take speed in places
- Direct access out of module VS. through subroutine call sequence
  - Overhead involved in subroutine argument access
  - Essentially no overhead in module access
- Typically

<u>Module Access Faster</u>
Scalars, fix length array, Simple DT

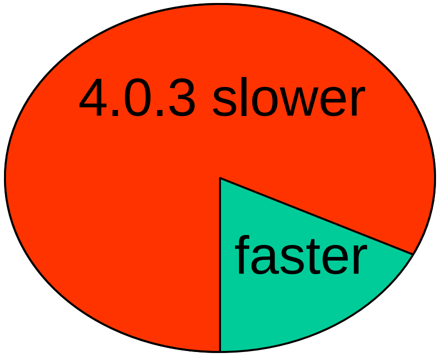
<u>Subr. argument faster</u>
Sub-derived type

<u>Toss-up</u>
Array section, pointer

- Data attributes affect

Timings

- Most of the database changes introduced slower memory access devices into 4.0.3.
- Code slowdown is expected for all problems.
  - Five out of six test cases run slower
- For some problems, 4.0.3 is faster than 2.4.3
  - Proper advantage taken of pointers and subroutine calls



Timing Study of 4.0.3

- It was reported that 2.4 runs slower than very old versions like rlpdoebf08
- Sparked a comparison of those two and of 4.0.3 against 2.4.3
- A Fortran program that extracts start and end time from RELAP5-3D runs for any version was written to perform comparisons efficiently
- **NOTE**
- **The changes reported here will be made in future code releases, not in 4.0.3**

Detailed Study of Timings

- Statistical profiling methods provide insight into code bottlenecks
  - Sample where program counter sits in code every so-many clock cycles (often every 100 – 1000 cycles or so)
  - Varies from run to run of the same problem based on computer workload
  - Affected by compiler options such as optimization & inlining
- GPROF is a built-in timer available with Intel Fortran
- It was applied to study Typical PWR 1200 second run
  - with default installation options
  - Semi- and nearly-implicit

Detailed Study of TYP1200

- With default installation options plus activation for gprof capability
  - PHANTV is largest time-consumer
  - MOVER and VEXPLT are next
- MOVER copies memory from old to new on a time-step backup or from new to old on a successful advancement
  - Much larger percentage since full back-up replaced partial
- Solver routines should be largest, but are surprisingly efficient
  - LU factorization < 1.5% of run time
  - Back substitution < 1%

Detailed Study of TYP1200 Nearly

- It was applied to study Typical PWR 1200 second run
  - with default installation options
  - Semi- and nearly-implicit
  - PHANTV is largest time-consumer
  - MOVER and VIMPLT are next
- MOVER copies memory from old to new on a time-step backup or from new to old on a successful advancement
  - Much larger percentage since full back-up replaced partial
- Solver routines are again surprisingly efficient
  - LU factorization < 3%

Detailed Study of Timings

- Open Speed Shop uses statistical sampling for closer view
  - Can show timings by function
  - Can show timings within routine – reveals slow lines and loops
- All-function analysis for typical PWR 1200 second
  - Power (raising an number to a power) is most time-consuming
    - Should be investigated
  - PHANTV is second largest
  - Heavily impacted by inlining

Runtime Improvements Going Forward

- Analysis of time-consuming lines shows
  - Some if-tests are among most time-consuming
  - Also some else-clauses (one BLANK else in particular)
  - A few do-loop statements
  - Some calculation statements ranked high
  - Some static quantities were recalculated every time-step
- Mitigation Methods devised thus far:
  - For same if-clause(s) repeated with no change to quantities in a subroutine
    - Store comparison in logical variable
    - Replace if-clause(s) with variable throughout routine
  - Similar strategy can be effective in a long much-used section of code

Runtime Improvements Going Forward

- For time-consuming else clauses
  - Change test order to reduce # things checked
    - If things A & B are checked, but mostly B occurs, check B first
  - Reverse the if-test (apply <u>.not.</u> to the if-condition)
- Turn off unneeded if-statements
  - Diagnostics that are never used except for debugging runs were “live” in all the BPLU routines.
  - Applying an if-def reduced run time
- Do loops run faster :
  - With unit (or fixed) stride
  - When the start and end values are variables, not calculations

Runtime Improvements Going Forward

- Blocks of calculation statements can be speeded up
  - By replacing a repeated array-reference with a scalar copy
- Single calculation statements can sometimes be algebraically simplified
- Some FORTRAN 95 intrinsic routines are faster than loops
  - Introduce judiciously
  - Done in solver
- Some static quantities that were calculated in a double loop in subroutine LEVEL
  - This was reduced from 10 inefficient statements to four
  - It was moved to input processing

Runtime Improvements Going Forward

- Improvements from mitigation efforts based on Open Speed Shop information reduced runtime about 0.5%
- Improvements from compiler options can provide 0.5%
- Further improvements possible judicious use of:
  - Subroutine call arguments
  - Pointers to sub-types
  - Intrinsic functions
  - Interface blocks

Conclusions

- 4.0.3 runs slower than 2.4.3 on most problems
- 4.0.3 runs faster than 2.4.3 on some problems
- Numerous runtime reductions have already been made in 4.1.0
- Many more techniques remain to be employed.