prepared for plenary lecture at
17th International Colloquium on the
Dynamics of Explosions and Reactive Systems
July 25-30, 1999, Heidelburg, Germany

CFD – Bridging The Communication Gap

James J. Quirk [1]

Center for Simulation of Advanced Rockets
Digital Computer Laboratory MC-258
University of Illinois at Urbana-Champaign
Urbana IL 61801, USA

jjq@amrita-cfd.org

Abstract

Modern computational fluid dynamics (CFD) is a complex amalgam of numerical analysis, flow physics, computer science, not to mention common sense. As such, it is becoming increasingly difficult for individual workers to remain wholly self-sufficient. This talk will explore some of the inter-disciplinary, communication problems that must be overcome, before this relatively new research field can reach full scientific maturity. To avoid undue subjectivity, the talk will present a series of numerical case studies which serve to illustrate the fundamental changes of perspective in moving from algorithm development, through software implementation, to end application. These case studies have been automated using a purpose-built, computational operating-system and can be reproduced – in their entirety – by any interested party, for no cost other than raw CPU-cycles. The talk will explore how such automation opens up the possibility of improving simulation capability through massed scrutiny: thus doing away with the present cottage-industry mentality, and its concomitant vagaries, where "codes" are crafted on a one-off basis. The present extended-abstract provides some pre-requisites for understanding this computational philosophy so that the talk itself can concentrate on specific applications. Full details of the general approach can be found at http://www.amrita-cfd.org/FAQ .

Introduction

There can be little argument that CFD has matured to the point where it earns its keep across a broad range of applications (witness http://www.cfd-online.com). Consequently, the provisional title for this talk – *What CFD Can Do For You* – is clearly outdated. Interestingly, this title was chosen by a colleague (an applied mathematician) who graciously stepped into the breach, following my unavailability, when the conference programme was being drawn up. Upon reflection, possibly a more appropriate title for the times would have been – *What You Can Do For CFD*. The very same words, but a talk with a completely different message to convey.

Taken together, the two prospective titles illustrate the gulf in thinking between a CFD software-developer and a CFD end-user. Whereas the typical user begs-borrows-or-steals "a code" for a one-off simulation, focusing firmly on the flow physics of the application, the author of "a code" looks to spread development costs across a wide range of projects with different sets of users, and by necessity is often forced to concentrate more on software issues than on physical issues. Thus to a user: "a code" is purely a means to an end and is often viewed as a disposable entity whose intellectual content may not even be acknowledged. But to a developer: "a code" represents a considerable investment of effort, much of it emotional, and each new user (although welcomed) adds to the programming burden. In this extended abstract it seems appropriate to identify concrete mechanisms for bridging such communication gaps: because, ultimately, miscommunications between related disciplines act to limit CFD's potential. Hence the actual title of the talk.

The first step in finding a communication-solution is to recognize that there is indeed a communication-problem to solve. For instance, in response to:

> *considerable concern with the quality of published numerical results* [2]

the American Institute of Aeronautics and Astronautics (AIAA) have issued the following editorial policy:

> *The AIAA journals will not accept for publication any paper reporting (1) numerical solutions of an engineering problem that fails adequately to address accuracy of computed results or (2) experimental results unless the accuracy of the data is adequately presented.*

[1] Formerly of Graduate Aeronautical Laboratories, California Institute of Technology, Pasadena CA 91125, USA. Now in transit to Los Alamos National Laboratory, New Mexico, USA.

[2] This quote, along with the next three quotes are taken, without permission, from AIAA information sheet: "Numerical Accuracy and Experimental Uncertainty *AIAA Journals*."

The effectiveness of the AIAA policy hinges on their chosen definition for numerical accuracy:

> *The accuracy of the computed results is concerned with how well the specified governing equations in the paper have been solved numerically. The appropriateness of the governing equations for modelling the physical phenomena and comparison with experimental data is not part of this evaluation. Accuracy of the numerical results can be judged from grid refinement studies, variation of numerical parameters that influence the results, comparison with exact solutions, and any other technique the author selects.*

and the extent to which they enforce the diktat:

> *For computational papers, the author must provide an adequate description of the numerical solution procedure, if not documented elsewhere. In addition, the complete governing equations must be specified with sufficient detail along with the input parameters to the code so that a reader could reproduce the results of the paper.*

If CFD were a completely mathematically rigorous discipline then there would be no need for the above policy statement, as any investigation worthy of publication would automatically be reproducible via an unbroken sequence of logical steps. Unfortunately, the typical CFD investigation contains numerous software mundanities which act to limit its integrity. This is especially true of modern algorithms, such as adaptive mesh refinement schemes, which do not have a concise mathematical recipe but achieve their ends through convoluted programming. With the best will in the world, the more a CFD investigation relies on custom-written software, the harder it is to satisfy the diktat that the work should be reproducible by the reader. Even supposing the necessary information were available, the software-engineering needed to redo the work would prove prohibitively expensive for all but the most persistent. On this score, matters are compounded by the apparent unwillingness of the scientific community to embrace the need to train its members to program efficiently: a criticism which deserves some explanation.

Casual programmers (i.e. individuals with no formal training in software construction) who switch from an old-fashioned language, such as *Fortran*, to a modern one, such as *C++*, often badger their peers to follow suit: flushed by the successes of an improved programming style. But before doing so they should ponder this assertion, made by the architect of *C++*[4] (my underlining):

> *... as programs get larger, the problems associated with their development and maintenance shift from being language problems to more global problems of tools and management.*

In other words, there is much more to constructing software than selecting a programming language and learning to use its syntax. This perhaps explains, but not pardons, why many "a research code" is unavailable for inspection: the author is embarrassed by its construction.

A similar assertion can be made about CFD simulations:

> *... as simulations get larger, the problems associated with their development and maintenance shift from being algorithmic problems to more global problems of tools and management.*

The upshot being that if the right *tools and management* were put in place then many of the concerns which prompted the AIAA computational policy could be tackled head on. For instance: grid refinement studies, numerical sensitivity studies, and comparisons with exact solutions, are all time consuming activities, and so it should come as no surprise that they are not performed as often as they should be. But if such activities could be fully automated then comprehensive validation tests could be run routinely, and even standardized. Of course, recognizing the need for the types of *tools and management* to do this is a far cry from actually producing them, and an even farther cry from successfully convincing a conservative community of the need to change its work practices.

Back in 1995, I started converting an adaptive mesh refinement algorithm into an interactive teaching aid so as to allow students to explore the practical aspects of compressible, computational fluid dynamics. The aim was not to turn experimentalists into heavy-duty programmers, but to produce a hierarchical, open-ended system in which individuals, reluctant programmers and CFD experts alike, could engage in unambiguous, scientific dialogue. Over time, *Amrita* has grown into a full-blown operating-system for automating numerical investigations[3]. This system, although far from perfect, demonstrates how the AIAA diktat could be realized in a methodical, watertight fashion.

The remainder of this extended-abstract will attempt to add substance to this claim by presenting a concrete numerical example. But before doing so, it is worth noting that the current *Amrita* installation-kit runs to over 520,000 lines of source-code, and so it provides a *tool* to enable any interested party to navigate the source in a painless fashion: see `http://amrita-cfd.org/source-code`. An examination of the source reveals that *Amrita* could legitimately be called any of following: an educational aid; a research tool; a communication protocol; a document preparation system; a quality assurance system; an archive system; a device-independent compilation system; a literate-programming system; a run-time shell for "legacy code"; a computational operating system; a labour saving device; a computational philosophy. In fact, almost anything *except* "a CFD code."

AmritaMailit::run_T5

Experience shows that reluctant programmers are easily put-off *Amrita* by the sheer size of the system. But the system is large precisely because it is striving to take care of all the tasks a reluctant programmer should not be expected to cope with. It also comes as a shock to many that the system has its own programming language (also called *Amrita*, but note the typographic distinction). The system needs its own language to be able to articulate the wide range of tasks needed to automate a numerical investigation. For instance, this six line *Amrita* fragment:

```
fold::file $abstract -> listing
fold::latex { typeset listing
   \begin{verbatim}
      *listing
   \end{verbatim }
}
```

produces this listing of the abstract you are currently reading:

```
...  ICDERS top matter
Latex2eHead {
   dir  = 4ICDERS
   file = jjq.tex
}
...  title
...  abstract
...  introduction
...  AmritaMailit::run_T5
...  closing comments
LatexTail
Latex
```

Several observations are needed to preempt the inevitable question – *Why bother?*

First, re-read the AIAA diktat; it effectively stipulates that precise and complete documentation is an integral part of a numerical investigation. Here, for preciseness the *Amrita* fragment is typeset from the actual working script, and in turn the script-fragment typesets the abstract from the original [3]. In so doing, the abstract is *folded* so that its structure is immediately apparent to the reader [4]. A computer scientist would call *Amrita* "a literate programming language," such languages are designed to produce self-documenting programs. The rationale being that manually prepared documentation is both onerous to produce and rarely a true reflection of a program's innards, but if the process is automated then it becomes straightforward to produce error free documentation which evolves as the program evolves. Through its use of *program folds*, *Amrita* could also be called "a cooperative programming language." If it needs to utilise a specialist language to accomplish some task, the new language can be folded into *Amrita* as LaTeX is folded in here. In essence an *Amrita* program can be viewed as a tree in which each branch can, if needs be, employ a different parser. The main implications of this are: *Amrita* cannot become outdated and superseded by another language, because with the right plumbing it can always fold in the new language (e.g. `fold::python`); reluctant programmers need only unfold an *Amrita* script (fed to them by an expert) to the level of complexity they feel comfortable with; all the activities needed to *validate* a numerical investigation: setting up a problem, running a simulation, plotting results, documenting the method etc., can all be freely intermixed within a single program.

Amrita is not a random collection of tools and much of its utility stems from the seamless manner in which it operates. For instance, consider the *AmritaMailit* shown in Figure 1 (a), it contains all "the user-supplied code" (albeit in an encoded form) needed to cajole the system into producing Figure 2. The packing and unpacking of an *AmritaMailit* takes place automatically in such a fashion that anyone with access to an *Amrita* installation [5] can now springboard directly off the efforts of the original author. In this case: Prof. Hans Hornung, Director of Graduate Aeronautical Laboratories California Institute of Technology.

[3] The following is given for completeness, but need not be fully appreciated to follow the main argument. `fold::file` is an *Amrita* keyword which takes the contents of the file identified by `$abstract` and *folds* it following a specific set of rules, depending on the filetype, and stores the resultant text in a *string token* called `listing`.`fold::latex` is a keyword which embeds a block of LaTeXinto the *Amrita* script. The `*` is an operator which expands the contents of `listing` much as `*` dereferences a pointer in `C`. Full details are given in [3].

[4] `Latex2eHead` is an *Amrita* standard-library procedure which constructs the head of a LaTeX document – here *jjq.tex*. `LatexTail` writes the tail of the document, and `Latex` runs *latex*, *dvips* and *bibtex* – having first requested *bibitems* from a series of local and remote servers – to produce a *PostScript* file *jjq.ps*.

[5] An *Amrita* kit can be downloaded by anyone with a *genuine commitment* to see the system succeed. Interested parties can obtain a 90 day licence from `licence-mgr@amrita-cfd.org` which automatically rolls over to a life-time licence upon receipt of a suitably polished *AmritaMailit*. Before requesting a licence, please make sure the system will run on your machine: for a list of platforms, see `http://www.amrita-cfd.org/FAQ` .

(a)

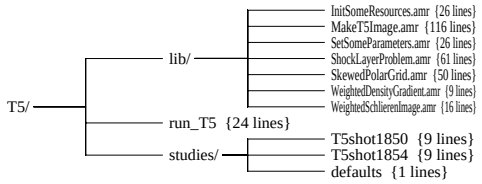
```
-----BEGIN PGP SIGNED MESSAGE-----

AmritaMailit::run_T5 {
   origin {
      Amrita v1.44 R22-03-99
      user   James Quirk (aka jquirk)
      date   Fri Apr  9 11:04:36 CDT 1999
   }
   operation {
      cd T5
      amrita -mailit run_T5
      amrps images/T5_shot1850.ps
      amrps images/T5_shot1854.ps
   }
   script {
      M'XL("!4E#C<''U0U+G1A<@#M'/USVL8R/_NON+HT'<R')!"D3$G'L5V'/-OQ
      M')(XDW@\LCA'L=!17133-/_[V[W3)TC@OCJ\]E4[8R-TNWMWNW=[NWMW#-7Z
      MDZ\-I"FUVRIY0MJ**C7ADY"F*CY]D'AIM:5&4U&4MDR(+$E*XPE1OWK+'#S'
      MU6Q"GGSZQ3/LNVP\W='L7;1GQS!4ZX[KC0SJ?+UQ\'#]2RV5ZU]692G7_RX@
      MIO^AZDR9*S]7I4>N'Z9SJ]7,UK_2"/6O-F&@$+FAR/(3\MCM2(5_N/['S!QU
      M.H8U8N3S'NA8[\N>,Z>Z,$Z2T\/S\T-".ITNDGM2>\^A+IEJEM/IG',^X,)[
      M^796F\U-_C;^TM1FMR-M%;4O%-\EC?C;^51SJ"5P_]=R^:=’VOQO/G(=6^>_
      MT@KGORHWQ?QOYO-_%Y"8_UF37VZD3TZUIN:S_^\.L?D_HF/-,UWGT>O8-O])
      M0UGQ_QI*6\GG_RX@FG\.,W^E-LP_F]&;L9G/P7\$P/PWC=NOFP-X>/PG2VJ#
      MQ_]J2\WCOUV'K_^>9;@#-J-]ZC#/UJE3TV:/UMTM]E]NM.10_[(,OJ"L2I*-
      MV_]=P-QF.EG3/D:"FN>RN>9.R4'A,PP&P]4Z'7TQYM'!\B+EPG_4)>0SP85D
      M1F?,7A*;HBSIC%HNYP1P:-O:<F#\1AUR<=J[?]U5)'0L_9+"[-8SS!&!VO4I
      M,2R73FS-9;;/*V7)NF6CY0VL6SZ&,2Y^\Q28.:RXK[,1K0?E^Z42<:?4V@LM
      MCZAX;D,M4.\$6-O@',\I%-@A6X),(AJ?[(7+[JC5Y6V]"5'K0.[9CL$LPL;C
      M-9I8'\>&1:N8@T@>CWTI?ZK3Y!Y]@@/'PGQU["]TOBVXEG4OOO%T]S
      MH0%.HFAN>A/#0J8W(+9$T4O-,79CZ.(IM2B7M,@%)!HD1$W6A1VAZ%/0.4<9
      MF]Y]U31FADM'S]A<,*TZ<]-P5ZDFMC'B#X)Q=6RX0)35Q]CC7+,=2K@.2%('
      MY*F/1<V1,19C#!X)CO7<K4F";_\'4Z;?G6E+:E_:[-:DLQW:_T9#"?W_9@-S
      M'7*SK>;V?R?'[?^:]H/YOV*R8,II,^I2.S(M$VTVTX(O:*&:89&?*'B*E%ID
      MN?R\0$@5E?AIA%2BF78?+PH+[%[,6,0+^B11T)!J>Y$5^4)^J/KK2,H*!,$P
      MV&=<+,@QFVE&L&3'^^0+;HEPG?7!KLB51 4NJ3%O2BY3"@Y3"P1U=T-$E,S7[
      M%.U@OUOH1[8+*\M:(6/M&S#30WO_DT'-P'#B^G$;]%NJP5(KRVI34F/%H'-?
      M7E(@%_[>\_#IVRXI%E"#9><7VRT6N)Y+)1_/LX,>8('HX'GEHERKTOMYL8I?
      MW'_7]0+P#_'%<C8/^**2ZV'5:F7'+L4DF]85@'?/%K>C9;S*'_JOWG3ERMNN
      M9W^XKOS<Y3S+CF5S?)EKXQOPU;'2"N5/ZZN.W$HW);?'I%9+?,N2+4&?B^5
      M*I==^46\&6.;4L>U*8HNK1G2.@7V9+7AH!'DDGP'.!HC'F$)*I_Z#P^(E)RS
      M1IJ]/&*P_L4]'11Q9B''!4IYQ$=W!_R@L4GUV'H]V%AZLK'T'99J)K,FI->5
      MR=AF,_*ZVY1A_29MB71@;::.;ANW-"'E;0S:G(&L2!U"[UU;F\-,<NDV*H@V
      M@*JY6BV.L8?)&L@,]'10V-I(FZ,8R4S3IZ'!.^"1KI]SZDP/?9*HG:B8]!(2
      MMWL%7$R41::O8/>BDEC[2>GTI;>HSS_063^"'FH;6K)=F\I1Q-283,%C.Z:6
      MB[C+4UL;&8"[K2TS[8[&1_>(S1-FTDH/N?"CKZMSGH7+';P.'8'_EUR('M/[
      MV^[_R6TI\O]:#?3_FHJ4^W^[',_K;@AOO?'W2?PSL0WC(’A6Q0"\\BM2QQ.
      M?(.FU]X<SR-!2H1>X"%Z'4O7X_.YYTSUQ8@44D/P*'(7B8OU0-U'?#$V3-KE
      M==0<6U]%^=:P=-.#_NT?GO=O!F_.ZO#9&Q[NIWL-@[<O^V_>#GL7)V3P[W?]
      M2ZEX<5KIG5=>GU>.KRK'[RM7E?>5WKO*ZT$IG8'@?C-\?]F[6(O](Y2+(0D8
      M9R(=OSPCO-*-&%=%$:/,W!<MBH9>G):R&X<D[_\8"3:6=5D=>X^CP['S1.IW3
      MD^'-\,V_3BZ*ST1-G4[_6:6?P;DOPQ#KEX^OTHNOCJ*XV))J!_U2]M)E#Y":
      MY</AX451I%'R*CM^@VY"E<O\X.+T0,YT&@$3'0E$?;T%%6#XZF1XB%/II[,W
      MA\/BZX/7@ZI2'D>U#JY+;1-E7^$??>CH0?'JJ-I7E%)=J7%W^^C-H,@Y9VG%
      M9T%\%GWEH-B7@46I+-K1-^B]@W;46YO;<%7L5SZ7>!NJ_7)4ZR::]P%-OSSH
      M7GPC,+DX!HF8%\JZY\,W_8O,NDV),J2Q@/]"H)^X'PM/;5J/&9WJQ@;<FYD
      M8]J-"/_J-+*""4*<>)T_;%^2WE8(1VPVATYPZX[]$+W9("""17<..DZ#C85H-
      MB]D<K/)ZF@V_W<^9[1(>Q_(E(F;?<R_MKP:^_Y?AQ3^.'[AM_[_9;(?^7[.!
      MYW]:<J.1^W^[’.[_9G@_\'//G7#>A]DUEX'5TRQPN7B"28ZYA''FYU/?I"PX
      M>P."</0FZ_U7;SY<A^8"LQVF-G'(!\;8[^S^_G?XO"9%[=8I(N:!<SWE%*52
      MN5@(697(BV(A;$2)A/8LQN\>^;%5?I_^'+__?W.U,O\'^M0TJ$VMWDR;T$<*
      M'[?M_RKM5C3_,1:4U7:CE<__74!B_B>U'TQ_7-'=SZ;D@]21K_W)+=6>0]A$
      M1II]9U''(6Q,#"02))C@-<:&SGTBI/OQNHMJ11+'LJ(RH%M0[8Z,J>9"'<Z:
      M'8DL#]@3I'<HIA9CP63,O$3(,GY\&RL;>Y8>I-:XX1D,Y!'?,[-%GN@N7G&+
      M<_6A:ER7RF5PF=_S%^^#%\+9'@R4#.)/G/A31/R)$W^*$1_;4W9\'R..VSFL
      MI@YM$R:2HRY47T3ADP!5i4H3J'\'Y6WZ%ZNNWE0/#BSTG3A]0SP\(T8\CD
      MPS6IOL"W%>)GY!8N6>ERA%LM'&:I7BP'KG@6WA]=.E-82I#\?)\@&$DBSSY(
      MC)+RP@6;C&1SD[EB,'DK3&8_Q#A]N'[Q#S#+.P/?_I]#!#14')/HQV"S_9<E
      M50G._RB2(N'^;Z.9^W^['6[_8]I_^,XO7A8(K>US-=JG73#[;F2(LR)\&D<T
      M\\"3W.^2H7J#+'KXKS:/<-BOU#:U)<?QGJ=P8"'D\H17WYV_-&;N[8V=LD,
      MPH(7%#D/TE2]_=.$N@Y&AK&4G<RPRZ6.N2.L="C83#W'6/'FS#F&QA6^3@
      M=Z+.&U^(=V&UO1-Q+H<2RYM1&PRF&4@F%I&#U13A^EK1.D-^6$N;&#K1F043
      MP%K+>S@+'_>M887FPD])4.C07J[X3GH"9FN5:[F2A(C%;10<0Q/-U'VWCB75
      M_?MI,!1JD]\VDKM3'\<:C"&I(E442:K(DK218@I+F:''Y=RR>T[8E#CQ1D)F
      MN6P>)U2>U]IOG8+_OV]5FMJO)+]GXKE+OM<SS)$W(R'T9OGX$Y[P;]O499K
      MV?E')^B:3<5IC.9#I(4G'QT8A!2'QW:"+QL&3W/G@Z?Y%Q\\K>]K.A\J3::]
      M7E&5?/"L@G]1)6/H4-MF-GBNCH-^Y[.58?.,'^P<JF+YXK;MF[0<JK!_6S/(
      M\25EP0\PKC([Y!@O36]]7\HP33HBNF'K:+;O6:6PA#]?'PX$.=!2B1]P>+[I
      MT">$'K?@PL0CB0,_DCCPPX6#,%QH)D@W!WYQD00Q((=P3S'&R9''3Q6MH20#
      M-3\5!$T7\4M:[X)%;2]9LB)[GK[7V0R::,'B"6VE>#[#<C6S&JV88V,"'8AX
      MKS"!U9AB5"G-IX;N@,,S2ZIL0-W+5-DXQF\T>_Z?MP.3NN'GK6(QaS-J@7'P
      M0VYKZX=AS\ZCYS01K><4#Q/EK;7REUOHYW,CFS[]&&YB/A?9<J;Y)/.5P\B:
      M'XJ:NNZ\4Z\O%HN:4$-5'X]JS)[4N=M8+_AFO(#6-84^\'3@':WHQNG)B*G=
      M4M.$T9>VQ1-#O3.]T61ML@+,N,N'!XSN;[!7:TO0FJT2G#D%,'Z,CC&"D0!3
      M97V0,3!<D[7Z0@AI"!^O*43P_JL'S=JI]E)@"#.C]^2<N"P^\PX"U>\K)$S
      M.C-HsT\^/B6O:)<U\HK9EN>G"%:K.S.957D#)&',/%-;.^NT00B;"'Q1C$W#P
      MN$TZ=C.DD6R%8<T]-RVR22>+#NXS>Z:Y@CR3AHCE=!P<@H4%U>';C>/B_G<U
      M>;Q?*437:3=NI7(VP3';)!LE8B/.9V[E$QROS>2#"-O9!.=W,]D(A'K@0VSG
      M&-PO'(Y/Q11SM#$M[A?B-SWVL_ED:)R$8\/5;F'\V%\^F[J9C0Q0.+_Y"#IQ
      MP:R0I]VGI%#VE5C(G#D^X<=`2R&A>+&=T!=7/R04+[82AB>O0T+4W\?OMA&&
      MHSD'D)"4?05D2Q%G>R#HA[MJG"Q[6JZ].$/5GZ&E]BE&L"*6[EE7S2H+UI7;
      M2F7"G8KN^EJV@55H2BK'JBUQ5BH)6#75!*LM[HB_$_YW34D&Y_\HOP!V&1K#
      MG=[_4X/]'UEMJ:JX_]?.\W^['''^;U7[@9?KNPSQ!!_^6L!2Y._"])W(N%5?
      M9&?<\&)=QGIKC(M52O:#7R$(##Y6L[]V&C'ZE9*&'V3A3"?FOXEXT&]V@CNQ
      MH'?B%/0H"'^3&;R57IA,QRR<P0AX@CP^,T(Q,+J5&%^KE[,B_R4-?VLC@>?G
      M_U^\02HPL=K5,REJ71YNQ9;LC;C^8I)8F3?AG\=<DXV(D^=Q!R1+<'\ZX2KJ
      MS&$P2T67LMT;X,7DHOY%?_GL(@/VW/>MF^#7OVF^S_Y(4VG^ES<]_RXJ2__[7
      M3B!Y>E'8?6>NZ3RET27N@E5'F$O!&\T:/[\(-@WLM[W$L^!+B*='/+VR]V4/
      M_(A.9_TJ^<H)2)--N/V&3_S)J;TY<UQQ])):'>Q%=7T:9#'+>*Z4\&LK=AU
      M7UNG$"M;B$+DM9N,O,7"VTW<!$\Q4C/-AIH5"=8^&N8KDB_)PG"G1!^;?:GV
      M/$A&F6S!/#7[ER9V=:(+[B-&+=S?$O^MHT0S#>F_K'/'^4178(V\\6ZWB4Q
      M00FCZ(L[MXLYY)!##CGDD$,,.>200PXYY)!##CGDD$,,.>200PXYY)!##CGD
      2D$,..>3PMX3_'%$47=$'>'''
      '
   }
}

-----BEGIN PGP SIGNATURE-----
Version: 2.6.2

iQCVAwUBNx5Ga7mgfLsjP/xJAQGyBgP+IShLMBKF6llx45xqa2k2PcwtLE1SKpZ6
vhDXoWVOsAIvhFw0itAcWL795gTt9GC+5GSDCsOhgxbuvaNGX7DnzAI5GUTOHQAE
t404V5496CyuwFyEBEhg3PO91DkEeB88n9+IiPNn41+5RuXHNIzwus36UfbiJOG3
hZoFGRWHJUU=
=C/qZ
-----END PGP SIGNATURE-----
```

(b)



(c)

```
EulerEquations {
   space    = two-dimensional
   symmetry = cylindrical
}
lib::InitSomeResources
plugin amr_sol
logfile logs/T5
postscript on
foreach T5shot (1850,1854)
   SetSomeParameters shot=$T5shot
   ShockLayerProblem
   solver code/body_roe
   fold::amrita { check all is well
      march 20 steps with cfl=0.8
      flowout $hans::results/phase0
   }
   do n=1,$hans::phasen
      march 300 steps with cfl=0.8
      flowout $hans::results/phase$n
   end do
   MakeT5Image {
      shot = $T5shot
   }
end foreach
```

(d)

```
proc ShockLayerProblem {
     ...  parameters
   } <- hans::
   ...  define the Domain
   ...  define the SolutionField
   ...  define the BoundaryConditions
   ...  prime  the adaption machinery
end proc
```

(e)

```
fold::amrita’composite { experimental-numerical figure
   ...  reset graphics page
   autoscale on $this
   plotfile $ps
      paste http://www.amrita-cfd.org/image/$T5shot $here
      paste $overlay $ontop
      ...  do labelling
   plotfile
}
```

Figure 1:

(a) “The code” *AmritaMailit::run_T5* is sufficient for any *Amrita* user to reproduce Figure 2:

unix-prompt-originator> ***amrita -mailit run_T5***
unix-prompt-recipient> ***amrita run_T5.mailit***

(b) The mailit unpacks itself into this directory tree. Other files are produced as the mailit runs, for instance: *code/body_roe.src* contains the source for a *patch-integrator* and is compiled using the script *code/body_roe.mk*. The compilation process adds *bindings* so that the *patch-integrator* can communicate with *Amr_sol*[3].

(c) The *run_T5* driver script; the expanded *Amrita* fold, when collapsed would appear as:

```
... check all is well
```

(d) *Amrita* procedure to set up the computational problem. Interestingly, it has fewer lines than the procedure `MakeT5Image`. The reason for this lies with the *choice of syntax* for defining flow problems[3].

(e) The *Amrita* fold within *MakeT5Image.amr* which pastes-up the experimental and numerical images into a single figure: `paste` is happy processing both local files and *urls*; ditto for most other *Amrita* commands (e.g. `solver`). Where necessary, digital-signatures are used to safeguard against “Trojan-horse code.”

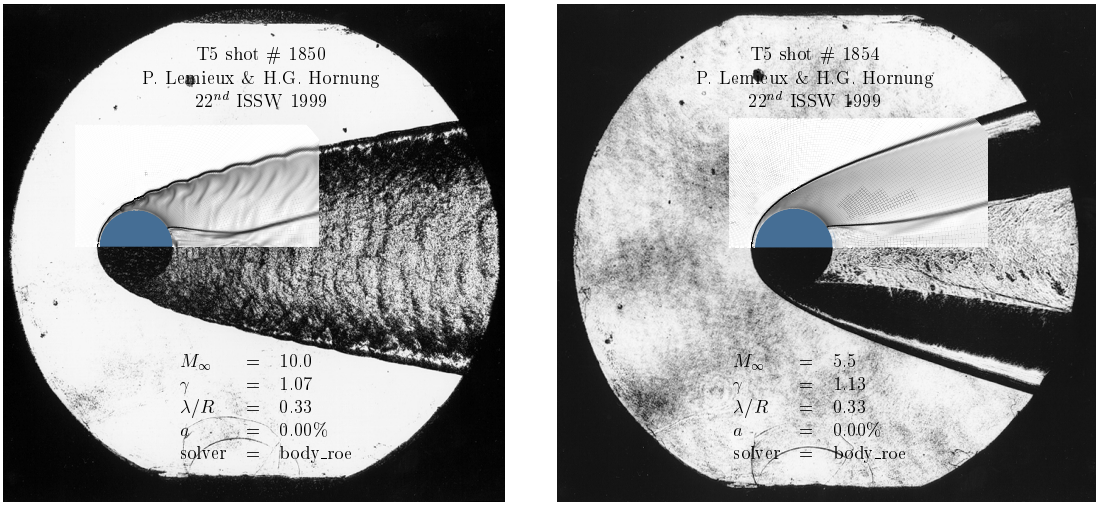


Figure 2: Output of *AmritaMailit::run_T5*. The images show experimental shadowgraphs of a one-inch diameter nylon sphere fired into propane, with results from numerical simulations superimposed (see [2] for details). The AIAA definition on numerical accuracy is careful to distinguish the *evaluation* of a simulation by comparison against experiment (as done here) from a true *validation* of a simulation which necessitates comparison against an exact solution for the model employed.

The present *AmritaMailit* unpacks itself to the source tree shown in Figure 1 (b), and the main driver routine is shown in Figure 1 (c). All told, there are just over 300 lines of *Amrita*, which for the competent script writer represents about one day's effort. *Amrita* scripts are short because they merely schedule work to be done by a `plugin` computational engine, which is then trusted to carry out the work as it sees fit. But this does not mean that an engine is necessarily a black-box. In the case of *Amr_sol*[3], which is built around a block-structured mesh refinement algorithm, the instruction:

```
solver code/body_roe
```

causes a *patch-integrator* to be linked dynamically with the engine (*unix-prompt*> ***man dlopen***). Dynamic-linking is a powerful programming technique for constructing programs which can bolster their capabilities on-the-fly. Here *body_roe* is a specialist component for integrating the two-dimensional Euler equations on a body-fitted grid. It amounts to 388 lines of *Fortran* and is the one component within *Amrita* which is close to the classical notion of "a CFD code," but with one crucial distinction. All the work needed to schedule the *T5* simulation and process its results live external to *body_roe*. Thus it can be thrown away and replaced by another *patch-integrator* of the user's choice without affecting the leg-work needed to produce Figure 2.

Amr_sol could also be termed "a code," and is also a replaceable item within the software hierarchy, but at over 40,000 lines, much of it dense logic, the label starts to become derogatory. On the other hand, to call *Amrita* "a code" would be as nonsensical as calling *UNIX* "a code." But some observations are in order lest you think this is just playing with semantics. *Amrita* is a general-purpose, user-programmable system for orchestrating simulations and it is not tied to a single computational methodology (AMR or otherwise). Nor is it tied to solving a single set of equations. Here `EulerEquations` is a standard-library procedure which teaches the `plugin` a set of symbolic mappings for writing to, and reading from, the computational domain using `notation`: `RHO`, `U`, `V`, `P` etc. The procedure could have been written by the user, but in the interests of standardization the system provides a definitive version. Thus to satisfy the part of the AIAA diktat which deals with *governing equations*, the clued-in scriptwriter could get away with:

```
grab::info BCG:Latex::[Document] from file EulerEquations.amr
parse token Document
```

which grabs a block of folded-information from the file containing the library procedure and then executes it to typeset up the equations. Inside the script, the named information was introduced using:

```
fold::info'Document {
    ...  body of fold deleted
```

where the identifier `Document` is a moniker chosen by the programmer. Thus the `Document` fold lives inside a `Latex` fold which in turn lives inside a `BCG` fold.

The primary customer of the BCG information is a standard-library routine `BasicCodeGenerator` which constructs CFD components upon demand [6]. Here the routine `InitSomeResources` runs:

```
EulerEquations
plugin amr_sol
BasicCodeGenerator {
   solver   = body_roe
   scheme   = flux-limited'operator-split
   grid     = body-fitted
   document = yes
}
```

to produce – on my office machine – the shared-object *code/AMRSO/serial/Solaris/sparc/body_roe.so*. In addition, a 30 page document is constructed which explains the `solver`'s construction. This document can be used either by a student to learn the basic construction of "a CFD code," or it can be used by an expert to see how to plumb in "third-party code." In principle, any explicit integration scheme, developed for a topologically rectangular patch, is a candidate for linking with *Amr_sol*. The motivation being that "a serial, single patch code" once linked with the engine becomes – on a parallel machine – "a message-passing, adaptive mesh refinement code." This is possible because all the AMR machinations and inter-node data transfers are orchestrated external to a patch; a strategy which proves efficient for the target applications for which *Amr_sol* was *designed*[3]. Of course, many details take place behind the scenes to allow this to happen, and therein lies one reason for the seemingly bizarre emphasis of this abstract for a colloquium on the dynamics of explosions and reactive systems.

Even with my extra page allotment, over the standard length abstract, there is not enough room to fulfill the AIAA diktat on a meaningful problem. This statement hinges on the interpretation of *reproduce*; the one point where I part company with the AIAA and most other individuals in the CFD community. The thrust of this paper has been to demonstrate that with the right *tools and management* a CFD investigation – as distinct from an isolated computation – can approach the reproducibility of a theoretical investigation. There is no need to be resigned, as the AIAA appears to be, and bracket numerical investigations with experiments in the reproducibility stakes. Simulations are inferior to experiments in many ways; their superior strengths, however, are portability and repeatability. The fact that CFD developers and end-users are not exploiting these strengths to the full will not result in an apocalyptic spate of planes dropping out of the sky. The problem is more insidious than this and essentially pertains to the division of labour needed to advance CFD to scientific maturity.

Judged by the teraflop yardstick – beloved of computational grand-challenges – Hornung's computations are small beer, because they are so cheap to run. But his intellectual contribution lies not in the CPU power needed to run the simulations, nor with the construction of 300 lines of *Amrita* script, but in the physical observations drawn from an extensive numerical parameter study[2], of which just two results are shown here. Although the experiments shown in Figure 2 are essentially inert, they were inspired by an earlier set in which the projectile initiated a detonation wave[1]. Therefore it is pertinent to ask what script changes would be needed to be able to simulate these other experiments: the answer depends on the choice of reaction model.

`EulerEquations` could be changed to `ReactiveEulerEquations` and given the reaction parameter `model=1step-Arrhenius`, for which *BCG* could be coaxed into generating one of 100+ different *patch-integrators*. Then, after the addition of a heat release and an activation energy the reactive simulation could be run as readily as a homework problem given in an introductory course on CFD – *How?* *Amrita* is designed to allow simulations to be scheduled independently of the mathematical complexity of the physical model, and independently of the cost of the simulation [7]. The clued-in scriptwriter could incorporate the necessary changes in a literal 5 minutes. But if this reaction model proved inadequate then life becomes more difficult. A change could be made to `model=3step-ChainBranching` which – for the current installation kit – would reduce *BCG* down to just one `solver`, which may or may not be up to the job. Then if further improvements to the reaction model where needed to simulate the experiment, the 5 minute job has turned into a four year PhD thesis: unless outside expertise can be brought in to bridge the gap. Hence the need for "a cooperative computing system."

In many instances, CFD progress is now limited not by a lack of ideas but by the sheer software effort needed to implement new ideas. To combat this malaise, *Amrita* seeks to provide incentives for third-parties to want to work in situ, by offering the *tools and management* that are needed to push CFD forward, but which have now outgrown a researcher's capacity to craft them singlehandedly; at least in a reasonable time frame. This is done in the hope that specialist components then filter back to improve the system. So that over time, without compromising numerical accuracy, more and more applications become "5 minute jobs." Thus *Amrita* acts as a conduit between developers and end-users, neither of whom can survive without the other.

[6] "*BCG* codes" are constructed routine by routine, much as this document is constructed paragraph by paragraph. In fact the same "literate-programming techniques" are used in both cases. Interestingly, a reluctant programmer will often utilize the convenience of *BCG* while at the same time condemning the low-level *Amrita* constructs employed as system-overkill.

[7] Serial or parallel, 2D or 3D, `LinearAdvectionEquation` or `RelativisticEulerEquations`, makes no difference to the way the work is scheduled in an *Amrita* script and so has no bearing on either the length or complexity of "user-code." On the other hand, "the system-code" needed to support this utopia can be extremely onerous to produce.

Closing comments

This abstract is too short to cover all the subtleties of the communication gaps within CFD. For example, a computer scientist recognizing [8]: folds from *Occam*; literate programming from *Web*, or *Icon*; dynamic linking from *Perl*, or *Python*; binding generation from *Swig*; source archiving and navigation from *Cvs*; and so on, might conclude that *Amrita* has nothing, intrinsically, new to offer. But [3] carefully articulates a similar conclusion, together with reasons why the system is not proffered as a panacea. However, a case can be made for an Amrita-style system, not because there is a lack of existing *tools*, but because the existing *tools* are too arcane for casual programmers whose intellectual interests lie outside computer science. For instance, when a user issues a `solver` command, *Amr_sol* does not limit itself to linking in a shared-object as would a typical *Perl*, or *Python*, script. It goes on to perform a comprehensive series of checks to ensure that the *patch-integrator* is consistent with the current state of the `plugin` engine. The need for such *management*, over and above raw language features, lies at the root of Stroustrup's assertion[4]. Moreover, the very individuals who benefit most from such safety features, are the reluctant programmers who would have most difficulty articulating, by themselves, the reams of *Perl*, or *Python*, needed for said checks. Thus, at one level, *Amrita* is striving to be a *Mathematica*-cum-*Matlab* for CFD, albeit with an altruistic mission along the lines of *Linux* [9].

At another level, *Amrita* is striving to provide researchers with a software communication-protocol for propagating CFD-expertise out to community, whether this involves the introduction of new algorithms, or devising standardized tests which reveal weaknesses of current approaches. On this score, it is important to dispel two popular misconceptions: (i) standardization limits intellectual freedom; (ii) simulations which takes days to run, if not weeks, do not need automating. One, CFD is already subject to standardization, whether it be the AIAA diktat on numerical accuracy, or the IEEE-754 rules for floating-point arithmetic; a society cannot survive without some elements of standardization. Two, the longer an end-application takes to run, the more important it is that "the code" is subjected to automated *validation* tests between runs so that "code rot" is detected before it wastes large amounts of CPU time and results in false scientific conclusions. Moreover, if computing power continues to improve at present rates: today's cutting-edge, research problems will eventually become tomorrow's homework problems and so the sooner they are automated, the better.

To come full circle, the answer to – *What You Can do For CFD* – depends on your disposition. A first-step, perhaps, would be to recognize that CFD has become a victim of its own success: there are far too many algorithmic variations on a theme to know precisely when to use one method in preference to another – witness the 100+ schemes that `BasicCodeGenerator` can produce for the `ReactiveEulerEquations`; the all important "code" – which is getting more complex by the year – is rarely open to scrutiny, and so too many claims have to be taken on trust for a discipline with pretensions of mathematical respectability. A second-step, at least for reluctant programmers, would be to recognize that CFD's problems can only be solved through software *tools and management* that transcend specific physical applications, but since the *tools* are essentially labour saving devices, it pays to learn to use them productively: there can be no CFD gain, without at least some software pain, whatever route is chosen. In the case of *Amrita*, the talk will present a collection of numerical investigations: pulsating detonations, detonation diffraction and reflection, which can be picked up by interested individuals to springboard off, adding intellectual content on top of what the present author can achieve singlehandedly.

If you are unswayed by any of the preceding lines of argument in this abstract, and you are unwilling to substantiate counter arguments using *AmritaMailits* [10], then it is probably fitting to end on a light note with an old engineering joke – *It takes a crank to start a revolution!*

Acknowledgement

I am indebted to Hans Hornung whose enthusiasm for *Amrita* has been a constant source of inspiration. I would also like to thank John Buckmaster and Mark Short for giving me the opportunity to get on my CFD soap-box, although I accept full responsibility for any criticisms which come as a result.

References

[1] J. Bélanger, M Kaneshige and J. E. Shepherd. Detonation initiation by hypervelocity projectiles. In *Shock Waves*, (eds. B.Sturtevant, J.E.Shepherd & H.G.Hornung), Proc. 20th ISSW, World Scientific Press, 1995.

[2] H. G. Hornung. Shock layer instability near the Newtonian limit of hypervelocity flows. Proc. 13th Australasian Fluid Mech. Conf., eds. M. C. Thompson & K. Hourigan, Monash University, Dec. 1998, 111-118.

[3] J. J. Quirk. (i) *Amrita* : A Computational Facility (for CFD Modelling). 72 pages. (ii) *Amr_sol*: Design Principles and Practice. 81 pages. In "29^{th} Computational Fluid Dynamics," von Karman Institute Lecture Series, edited by H. Deconinck. ISSN0377-8312 1998. Available at: `http://www.amrita-cfd.org/doc`.

[4] B. Stroustrup. The C++ Programming Language (2nd ed. p.7). Addison-Wesley, 1991.

[8] If any of these *tools* are unfamiliar to you, a web-search will throw up as much information as you can handle.

[9] See: `http://www.mathematica.com`, `http://www.mathworks.com` and `http://www.linux.org` .

[10] This paper is itself generated by an *AmritaMailit*, with the specific aim of allowing flaws – found by critical third-parties – to be corrected.