Implementing the Kernel of the Australian Region Weather Prediction Model in SISAL

G.K. Egan
Laboratory for Concurrent Computing Systems
Swinburne University of Technology
John Street, Hawthorn 3122, Australia

Abstract

The SISAL implicit parallel programming language has been implemented on a number of platforms ranging from scientific workstations through medium cost multiprocessors to high end parallel super computers and recently massively parallel processors. No changes to source code are required to obtain good performance across these platforms and it has been claimed that SISAL exhibits similar uniprocessor performance to FORTRAN while providing significant speedup compared to FORTRAN on multiprocessors.

The Australian Region Weather Prediction Model is an experimental FORTRAN code which uses a variable resolution nesting scheme to provide higher resolution predictions over important areas of the Australian continent such as cities and coastal fisheries. In this preliminary study we explore the performance of the SISAL implicit parallel programming language on a significant scientific application by recoding the kernel subroutine of the Model in SISAL. Results are presented for a low end SPARC workstation, an entry level Cray Y-MP EL and a high end Cray C90.

1 Introduction

The Australian Region Weather Prediction Model (ARPE) was developed by the Australian Bureau of Meteorology Research Centre [1] for short-term weather forecasting up to 36 hours. ARPE draws upon the work of Arakawa, Lamb and Miyakoda [2][3] for its formulation and is intended to be a production code for the prediction of weather over the Australian region. This paper will concentrate on the implementation of the core subroutine of the ARPE in the SISAL language and readers are directed to reference [1] for a detailed description of the model. The work is part of a continuing long term international study of SISAL being conducted in collaboration with the Lawrence Livermore National Laboratory.

2 The SISAL language

SISAL is a functional language for numerical computation [4]. The developers of SISAL have been able to demonstrate performance comparable with FORTRAN on a number of computing platforms including the Cray Research multiprocessors [5].

SISAL prohibits by design the ability to express constructions which lead to the side effects that make compilation for parallel computer systems extremely difficult. Examples of side effects include those which occur through the COMMON and EQUIVALENCE statements in FORTRAN and SISAL has neither of these constructs. SISAL is block structured and superficially resembles a number of modern languages. The single assignment nature of SISAL means variables have values assigned to them once. This requires some departure from a common style of programming where variables are re-used in programs sometimes for unrelated computations. Translation of FORTRAN programs into SISAL is not necessarily a simple process and can be complicated significantly if the program being re-expressed has been the subject of undisciplined maintenance or construction. This may be compounded if there is no original formulation of the mathematical model available. Direct transliteration of well written FORTRAN code can yield satisfactory results.

Most comparative studies to date have involved the complete recoding of an application in SISAL. In this study the mixed language facility of the current (V12.9.1) Optimising SISAL Compiler is used with an initial core subroutine being recoded.

3 The weather prediction model

The Weather Prediction Model code (ARPE) consists of some 10,000 lines of FORTRAN source code. Its preprocessors and ancillary code constitute perhaps another 5,000 lines of code. The code is generally well written with disciplined use of COMMON and EQUIVALENCE

statements. The kernel routines make almost no use of subroutines although the structure of the code suggests they should be used. ARPE then is a reasonable example of a code where inlining has occurred from the outset in an attempt to obtain improved performance. It predates modern FORTRAN pre-processors which automatically inline selected subroutines.

4 FORTRAN

The Cray Research FORTRAN tool suite used [6] runs under X Windows and is a marked advance on those generally available only a few years ago. The tool set comprises: a profiler (flowview) which identifies key subroutines and subroutines which are candidates for inlining; a pre-processor which performs inlining and attempts to identify and annotate parallel regions; an assistant for explicit parallel annotation (atscope); and a parallelism estimator (atexpert).

Routine Name	Tot Time	Calls	Avg Time	Percentage	Accum%	
INNER2	2.52E+01	9	2.80E+00	42.24	42.24	**********
LIE	1.09E+01	24	4.53E-01	18.23	60.47	****
PHYS	6.15E+00	5	1.23E+00	10.31	70.79	**
LIEBIG	5.61E+00	12	4.68E-01	9.42	80.21	**
LIEH	5.50E+00	12	4.58E-01	9.23	89.44	**
LIEBH	1.69E+00	9	1.88E-01	2.84	92.28	
SEMIMP	1.48E+00	9	1.64E-01	2.48	94.76	
VMODES	1.08E+00	4	2.71E-01	1.82	96.57	
INNER	9.56E-01	9	1.06E-01	1.60	98.18	
DADADJ	4.40E-01	11470	3.84E-05	0.74	98.91	
LAMLL	1.43E-01	2600	5.50E-05	0.24	99.15	

Table 1: Execution Profile (5 iterations Y-MP EL)

The original program was profiled using flowtrace to identify the core subroutines. For reasons already stated flowtrace did not identify any subroutines eligible for inlining.

The INNER2 subroutine was chosen as the starting point for this study but as it represents only 42% of the run time contribution no significant speedup is to be expected. The LIE and PHYS subroutines will be translated in due course. Our interest here is to confirm that the run time is not adversely affected and that underlying concurrency is uncovered by the OSC compiler.

4.1 Results for FORTRAN

The automatic parallel annotator was used to annotate the INNER2 subroutine. No attempt was made to resolve data dependencies in the original FORTRAN in this part of the study although this is intended later. The atexpert measurement tool was used to examine individual DO loops for predicted speedup. Atexpert is claimed to accurately predict performance for dedicated systems. The

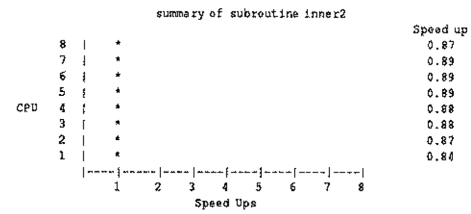

Figure 1: speedup of INNER2 predicted by atexpert

tool provides parallelism profiles and allows routines associated with parallel or sequential regions to be examined and analysed interactively.

It can be seen in Figure 1 that fpp failed to discover significant parallel regions in INNER2.

5 SISAL

5.1 Mixed language compilation

The osc compiler compiles and links modules written in FORTRAN and SISAL. In this FORTRAN is invoking a SISAL function. To do this the original INNER2 subroutine was replaced by a FORTRAN shell. The shell initialises the array descriptors required by SISAL and calls the replacement INNER2 written in SISAL [7].

Fortunately the array descriptors may be re-used for other arrays which have an identical shape. The ability to specify an offset for returned data structures could be used to avoid the often clumsy process of dealing with boundary values. The current descriptor mechanism unfortunately sets to zero the elements not written to.

5.2 The transliteration process

Although the mathematical formulation was available it did not provide significant assistance in the transliteration process. The INNER2 subroutine was directly transliterated into SISAL with no restructuring being attempted. A number of unintentional out of bound accesses were discovered in the FORTRAN program during this transliteration.

The transliteration process was significantly complicated by the size of the INNER2 subroutine. While the SISAL debugger (sdbx) gave some assistance there were many cases where sdbx was not able to determine the original source line causing the error. Other minor difficulties which would case irritation for programmers used to imperative styles also arose. In this case even though the author has a reasonable understanding of SISAL the passage

of time since writing his previous SISAL program still led him to be caught by the following:

```
for initial
  ....
  k:=0;
  while k < kz repeat
       k:= old k +1;
       ......u[k].....
```

Most programmers will expect k to be 1 when the variable u is accessed on the first loop iteration rather than zero as stated by the for initial clause.

Transliteration and debugging took approximately 35 hours.

5.3 Results for SISAL

The results for one call of INNER2 in FORTRAN and SISAL are shown in Table 2. In their current form both versions are several hundred lines long and the interleaving of initialisation, the calculation of primary meteorological variables and common working variables makes their inner workings difficult to comprehend (Appendices).

Language	Sparc	EL (1-cpu)	C90 (1-cpu)	C90 (4-cpu)
f77 -O	6.6+0.7			
cf77 -Zp		3.01+0.48	0.39+0.01	
osc -O	7.2+1.0	6.57+0.25	1.08+0.01	0.29+0.01

Table 2: Run Times for FORTRAN and SISAL

It may be noted that although the run times on the SPARC workstation for FORTRAN and SISAL are comparable performance on the Cray systems is not as good. It is believed that the transliteration resulted in a SISAL style which caused difficulty for the SISAL optimisers; this is currently being resolved.

6 Conclusions

A modest amount of difficulty was encountered in the transliteration of the kernel INNER2 subroutine into SISAL. The run time for this first SISAL implementation relative to FORTRAN is acceptable. Good speedup has been achieved with the SISAL version's runtime falling below that for FORTRAN at four processors. Given this promising start the study will now refine the version of INNER2 and move to the other dominant kernel subroutines LIE and PHYS. The PHYS subroutine is dominated by conditionally executed code as are many other weather codes. It is anticipated that this will produce a more demanding test for SISAL.

Acknowledgements

The author thanks the Australian Bureau of Meteorology Research Centre for access to the ARPE code. The author also thanks the members of the Laboratory for Concurrent Computing Systems for their contributions to the work presented in this paper.

Appendices

INNER2.F

The original code of INNER2 has been stripped out and replaced with descriptor initialisation and call to sinner2.

```
      SUBROUTINE INNER2
C
C  INNER2 CALCULATES THE RH SIDES OF THE MAIN SEMI-IMPLICIT EQUATIONS
C
      include 'arpe.inc'
      PARAMETER
     + (
     +  I2=I1+1, I3=I1+2, I4=I1+3, ILM=IL-1, ILN=IL-2
     +  ,J2=J1+1, J3=J1+2, J4=J1+3, JLM=JL-1, JLN=JL-2
     +  ,KZM1=KZ-1, KZP1=KZ+1
     +  ,CP=1.00464E7, G=980.6, HL=2.501E10, PBAR=1.E6, R=2.87E6
     +  ,RV=4.61E+6
     + )
C
      COMMON
     +        /DTDS/   DT,DS,DTI,DSI,DSI2,DSSQ,TDSI,HDTDS,BET65,DTMAX
     +       ,/INTGRL/ PRECP, PRECTA, CKS, EKE, PE, PSBAR, TRHAT, VROMG
     +       ,/KTAU/   KTAU
C
      COMMON
     +        /CDIFF/  CDIFF(IL,JL)
     +       ,/CORP/   CORP(IL,JL)
     +       ,/DNORM/  DNORM(KZ)
     +       ,/DQ/     DQ(KZ)
     +       ,/DTODQ/  DTODQ(KZ)
     +       ,/EM/     EM(IL,JL)
     +       ,/EMSQ/   EMSQ(IL,JL)
     +       ,/EMSQI/  EMSQI(IL,JL)
     +       ,/GAMA/   GAMA(KZ)
     +       ,/OMEGA/  OMEGA(KZ,IL,JL)
     +       ,/PHI/    PHI(KZ,IL,JL)
     +       ,/PS/     PSM(IL,JL), PS(IL,JL), PSP(IL,JL)
     +       ,/Q/      Q(KZ)
     +       ,/QPH/    QPH(KZ)
      COMMON
     +        /RM/     RMM(KZ,IL,JL), RM(KZ,IL,JL), RMP(KZ,IL,JL)
     +       ,/RTBAR/  RTBAR(KZ)
     +       ,/SIGDOT/ SIGDOT(KZ,IL,JL)
     +       ,/T/      TM(KZ,IL,JL), T(KZ,IL,JL), TP(KZ,IL,JL)
     +       ,/TBAR/   TBAR(KZ)
     +       ,/U/      UM(KZ,IL,JL), U(KZ,IL,JL), UP(KZ,IL,JL)
     +       ,/V/      VM(KZ,IL,JL), V(KZ,IL,JL), VP(KZ,IL,JL)
     +       ,/ZS/     ZS(IL,JL)
C
      REAL Q
      integer ik(100),iij(100),ikij(100)
      DIMENSION RMPR(KZ,IL,JL)
      DIMENSION TFLEV(KZ),DTFDQ(KZ),WVEL(KZ)
      DIMENSION VADVU(KZP1),VADVV(KZP1),VADVRM(KZP1)
      DATA VADVU/KZP1*0./,VADVV/KZP1*0./,VADVRM/KZP1*0./
      DATA OMG / 0.0 /
C
c SISAL array descriptors
c one dimension
      ik(1)=0
      ik(2)=0
      ik(3)=0
c
      ik(4)=1
      ik(5)=kz
      ik(6)=1
      ik(7)=kz
      ik(8)=1
c
c two dimensions
```

```
      iij(1)=0
      iij(2)=0
      iij(3)=0
c
      iij(4)=i1
      iij(5)=i1
      iij(6)=i1
      iij(7)=i1
      iij(8)=1
c
      iij(9)=j1
      iij(10)=j1
      iij(11)=j1
      iij(12)=j1
      iij(13)=1
c
c three dimensions
      ikij(1)=0
      ikij(2)=0
      ikij(3)=0
c
      ikij(4)=1
      ikij(5)=kz
      ikij(6)=1
      ikij(7)=kz
      ikij(8)=1
c
      ikij(9)=i1
      ikij(10)=i1
      ikij(11)=i1
      ikij(12)=i1
      ikij(13)=1
c
      ikij(14)=j1
      ikij(15)=j1
      ikij(16)=j1
      ikij(17)=j1
      ikij(19)=1
c
       call sinner2(
     +dt,ds,dsi,dsi2,tdsi,dtmax,cks, eke, pe, psbar, trhat, vromg,
     +ktau,
     +cdiff,iij,
     +corp,iij,
     +dnorm,ik,
     +dq,ik,
     +dtodq,ik,
     +em,iij,
     +emsq,iij,
     +emsqi,iij,
     +gama,ik,
     +omega,ik,
     +phi,ikij,
     +psm, iij,ps,iij,
     +q,ik,
     +qph,ik,
     +rmm, ikij,rm, ikij,rmp,ikij,
     +rtbar,ik,
     +sigdot,ikij,
     +tm,ikij, t,ikij, tp,ikij,
     +tbar,ik,
     +um, ikij,u, ikij,up,ikij,
     +vm, ikij,v, ikij,vp,ikij,
     +zs ,iij,
c      returns
     +rm,ikij,
     +sigdot,ikij,
     +up,ikij,
     +new_tp,ikij,
     +new_rmp,ikij,
     +vp,ikij,
     +eke,
     +cks,
     +trhat,
     +pe,
     +psbar,
     +vmrong)
c
      RETURN
      END
```

inner2.sis

```
define sinner2

% G.K. Egan 1993

type OneDReal = array[real];
type TwoDReal = array[OneDReal];
type ThreeDReal = array[TwoDReal];

global log(a:real returns real)

global sqrt(a:real returns real)

function boundary_cell(i,i1,il,j,j1,jl:integer returns boolean)
  ((i = i1)|(i = il)|(j = j1)|(j = jl))
end function

function divergence_sums(
          i,j,kz,i1,il,j1,jl:integer; dsi:real;
          dq:OneDReal;u,v,t:ThreeDReal;emsq:TwoDReal
        returns
          real, real, real, OneDreal, OneDReal,
          OneDReal, OneDReal, OneDReal)
  for initial
    sumu:=0.0;
    sumv:=0.0;
    sumx:=0.0;
    k:=1;
  while (k < kz) repeat
    k:=old k +1;
    sumu, sumv, sumx := (
    if boundary_cell(i,i1,il,j,j1,jl) then
      old sumu, old sumv, old sumx
    else
      old sumu+dq[k]*(u[k,i+1,j]-u[k,i-1,j]
        +v[k,i,j]+v[k,i+1,j]-v[k,i,j-1]-v[k,i+1,j-1]),
      old sumv+dq[k]*(u[k,i,j]+u[k,i,j+1]
        -u[k,i-1,j]-u[k,i-1,j+1]+v[k,i,j+1]-v[k,i,j-1]),
      old sumx+dq[k]*(u[k,i,j]-u[k,i-1,j]+v[k,i,j]-v[k,i,j-1])
    end if)
  returns
    value of sumu
    value of sumv
    value of sumx
    array of sumu
    array of sumv
    array of sumx
    array of (-emsq[i,j]*sumx*dsi)
    array of t[k,i,j]
  end for
end function

function sinner2(
          dt,ds,dsi,dsi2,tdsi,dtmax,cks, eke, pe, psbar, trhat, vromg:real;
          ktau:integer;
          cdiff:TwoDReal;
          corp:TwoDReal;
          dnorm:OneDReal;
          dq:OneDReal;
          dtodq:OneDReal;
          em:TwoDReal;
          emsq:TwoDReal;
          emsqi:TwoDReal;
          gama:OneDReal;
          omega:OneDReal;
          phi:ThreeDReal;
          psm, ps:TwoDReal;
          q:OneDReal;
          qph:OneDReal;
          rmm, rm, rmp:ThreeDReal;
          rtbar:OneDReal;
          tm, t, tp:ThreeDReal;
          tbar:OneDReal;
          um, u, up:ThreeDReal;
          vm, v, vp:ThreeDReal;
          zs:TwoDReal
        returns
          ThreeDReal,%new_rm
          ThreeDReal,%new_sigdot
          ThreeDReal,%new_up
          ThreeDReal,%new_tp
          ThreeDReal,%new_rmp
          ThreeDReal,%new_vp
          real,%new_eke
          real,%new_cks
          real,%new_trhat
          real,%new_pe
          real,%new_psbar
          real%new_vmrong
        )

let
  kz:=15;
  il:=65;
  jl:=40;
  i1:=1;
  j1:=1;
  i2:=i1+1;
  i3:=i1+2;
  i4:=i1+3;
  ilm:=il-1;
  j2:=j1+1;
  j3:=j1+2;
  j4:=j1+3;
  jlm:=jl-1;
  kzm1:=kz-1;
  kzp1:=kz+1;
  cp:=1.00464e7;
  g:=980.6;
  hl:=2.501e10;
  pbar:=1.e6;
  r:=2.87e6;
  rv:=4.61e+6;

  dtt:=
    if (ktau = 1) then
      dt
    else
      2.0 *dt
    end if;

  new_rm, rmpr:=
```

```
for k in 1,kz cross i in i1,il cross j in j1,jl
  t_rm,
  t_rmpr:=
    if (rm[k,i,j] > 0.0) then
      rm[k,i,j],
      rm[k,i,j] / (ps[i,j]+pbar)
    else
      0.0,
      0.0
    end if
returns
  array of t_rm
  array of t_rmpr
end for;

dmonp:=0.0;
emonp:=0.0;

new_tp, new_up, new_vp, new_rmp, new_sigdot,
new_eke, new_cks, new_trhat,
new_pe, new_psbar, new_romg:=
for i in i1,il cross j in j1,jl

  psijc:=ps[i,j]+pbar;
  psijci:=1.0/psijc;
  corf1,corf2:=

  if boundary_cell(i,i1,il,j,j1,jl) then
    0.0,0.0
  else
    0.125*(corp[i,j]+corp[i+1,j]),
    0.125*(corp[i,j]+corp[i,j+1])
  end if;

  emthad := emsq[i,j]*psijci*tdsi;
  em2tps := emthad*psijci*r / cp;

  emhad1,emhad2:=
  if boundary_cell(i,i1,il,j,j1,jl) then
    0.0,0.0
  else
    0.25*tdsi*(em[i,j]+em[i+1,j]),
    0.25*tdsi*(em[i,j]+em[i,j+1])
  end if;

  amonp:=emonp; % 0.0 then cycle_emonp
  bmonp:=dmonp; % 0.0 then cycle_dmonp

  cycle_dmonp, fmonp:=
  if boundary_cell(i,i1,il,j,j1,jl) then
    0.0,0.0
  else
    em[i+1,j]/(ps[i+1,j]+pbar),
    em[i,j+1]/(ps[i,j+1]+pbar)
  end if;

  new_bmonp:=
  if (i = i2) & (~((j = j1)|(j = jl))) then
    em[i,j]*psijci
  else
    cycle_dmonp
  end if;

  new_amonp:=
  if (i = i2) & (~((j = j1)|(j = jl))) then
    0.25*(new_bmonp+fmonp+em[i-1,j] / (ps[i-1,j]+pbar)
     +em[i-1,j+1] / (ps[i-1,j+1]+pbar))
  else
    emonp
  end if;

  cmonp:=new_bmonp+cycle_dmonp;
  cycle_emonp, new_cmonp:=
  if boundary_cell(i,i1,il,j,j1,jl) then
    0.0, cmonp
  else
    0.25*(cmonp+em[i+1,j+1]/(ps[i+1,j+1]+pbar)+fmonp),
    0.25* (cmonp+em[i+1,j-1] / (ps[i+1,j-1]+pbar)
      +em[i,j-1] / (ps[i,j-1]+pbar))
  end if;

  pse, psn:=
  if boundary_cell(i,i1,il,j,j1,jl) then
    0.0, 0.0
  else
    0.5*(ps[i,j]+ps[i+1,j])+pbar,
    0.5*(ps[i,j]+ps[i,j+1])+pbar
  end if;
%
%   extra variables for evaluating p.grad terms logarithmically
%
  psrmi, psrmj, psldi, psldj, zspdi, zspdj, emudsi, emvdsi, emrdsi:=
  if boundary_cell(i,i1,il,j,j1,jl) then
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
  else
    ps[i+1,j]-ps[i,j]-psm[i+1,j]+psm[i,j],
    ps[i,j+1]-ps[i,j]-psm[i,j+1]+psm[i,j],
    log(ps[i+1,j]+pbar)-log(psijc),
    log(ps[i,j+1]+pbar)-log(psijc),
    pbar*(zs[i+1,j]-zs[i,j]),
    pbar*(zs[i,j+1]-zs[i,j]),
    emsq[i,j]/(4.0*ds*pse),
    emsq[i,j]/(4.0*ds*psn),
    emsq[i,j]/(4.0*ds*psijc)
  end if;
%
%   compute total divergence
%
  sumu, sumv, sumx, vadvu, vadvv, vadvrm, wvel, tflev:=
    divergence_sums(i,j,kz,i1,il,j1,jl,dsi,dq,u,v,t,emsq);

  sigdot_k:=
  for k in 1,kz
  returns array of (
    if (k = 1) then
      0.0
    else
      wvel[k-1]-qph[k-1]*wvel[kz]
    end if)
  end for;

  vadvrm_k, vadvu_k, vadvv_k:=
  for l in 1,kz
    l1:=l+1;
    t_vadvrm, t_vadvu, t_vadvv :=
    if (l = kz) then
      0.0, 0.0, 0.0
    else
      emrdsi*(qph[l1]*sumx-vadvrm[l1])
       *(new_rm[l1,i,j]+new_rm[l,i,j]
       +2.0*sqrt (new_rm[l1,i,j]*new_rm[l,i,j])),
      emudsi*(qph[l1]*sumu-vadvu[l1])
       *(u[l1,i,j]+u[l,i,j]),
      emvdsi*(qph[l1]*sumv-vadvv[l1])
       *(v[l1,i,j]+v[l,i,j])
    end if
  returns
    array of t_vadvrm
    array of t_vadvu
    array of t_vadvv
  end for;
%
%   set up temperature difference terms
%
  dtfdq:=
  for k in 1, kz
  returns array of (
    if ((k = 1)|boundary_cell(i,i1,il,j,j1,jl)) then
      0.0
    else
      if (k = kz) then
        dtmax*(tflev[kz]-tflev[kzm1])+dtodq[kz]
      else
        0.5*(tflev[k+1]-tflev[k-1]) / dq[k]+dtodq[k]
      end if
    end if)
  end for;
%
  psmue, psmuw, psmun, psmus, psmu, psmvn, psmvs, psmve, psmvw, psmv:=
  if ((j = j2)|boundary_cell(i,i1,il,j,j1,jl)) then
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
  else
    if (i = ilm) then
      1.5*psm[il,j]-0.5*psm[ilm,j]+pbar
    else
      0.5*(psm[i+1,j]+psm[i+2,j])+pbar
    end if,

    0.5*(psm[i-1,j]+psm[i,j])+pbar,
    0.5*(psm[i,j+1]+psm[i+1,j+1])+pbar,
    0.5*(psm[i,j-1]+psm[i+1,j-1])+pbar,
    0.5*(psm[i,j]+psm[i+1,j])+pbar,

    if (j = jlm) then
      1.5*psm[i,jl]-0.5*psm[i,jlm]+pbar
    else
      0.5*(psm[i,j+1]+psm[i,j+2])+pbar
    end if,

    0.5*(psm[i,j-1]+psm[i,j])+pbar,
    0.5*(psm[i+1,j]+psm[i+1,j+1])+pbar,
    0.5*(psm[i-1,j]+psm[i-1,j+1])+pbar,
    0.5*(psm[i,j]+psm[i,j+1])+pbar
  end if;
%
%   commence vertical level loop
%
  tp_k, up_k, vp_k, rmp_k, omega_k,
  new_eke, new_ppe, new_pvromg, new_ptrhat:=
  for k in 1,kz
%
%     compute vertical advection contribs. in rhs of mtm. = ns.
%
%     compute horizontal advection terms associated with rhs of mtm. = ns.
%
    up_k:=
    if ((j = j2)|boundary_cell(i,i1,il,j,j1,jl)) then
      up[k,i,j]
    else
      let
        vat1:=
        if (k = kz) then
          0.0 %gke
        else
          -(vadvu[k+1]-vadvu[k])/dq[k]
        end if;
        ub:=u[k,i,j]+u[k,i-1,j];
        uc:=u[k,i,j]+u[k,i,j-1];
```

```
          ud:=u[k,i,j]+u[k,i+1,j];
          ue:=u[k,i,j]+u[k,i,j+1];
          vb:=v[k,i,j]+v[k,i,j-1];
          vc:=v[k,i,j-1]+v[k,i+1,j-1];
          ve:=v[k,i,j]+v[k,i+1,j];
          hadv1:=emhad1* ( ud*ud*cycle_dmonp-ub*ub*new_bmonp
           +ue*ve*cycle_emonp-uc*vc*new_cmonp);
%
%         compute pressure gradient terms on rhs of mtm. = ns.
%         logarithmically
%
          pg1:= ((pse-pbar)*(phi[k,i+1,j]-phi[k,i,j])
           +zspdi-rtbar[k]*psrmi
           +pse*psldi*( r*0.5* (t[k,i+1,j]+t[k,i,j])
           +rtbar[k]))*dsi;
          ct1:= corf1*(vc+ve);
          ubdiff:=
          if (dnorm[k]  =  0.0) then
            0.0
          else
            cdiff[i,j]*dnorm[k]*dsi2
             *( um[k,i+1,j]/psmue+um[k,i-1,j]/psmuw
             +um[k,i,j+1]/psmun+um[k,i,j-1]/psmus
             -4.0*um[k,i,j]/psmu )*psmu
          end if
        in
          (ct1+vat1-hadv1-pg1+ubdiff)*dt+um[k,i,j]
        end let
      end if;
%
      t_rmp_k, tp_k, omega_k:=
      if ((i = i2)|boundary_cell(i,i1,i1,j,j1,j1)) then
        rmp[k,i,j],
        tp[k,i,j],
        omega[k]
      else
%
%       calculate horizontal advection term in the temp. = ation
%
        let
          wvelav:=
          if (k > 1) then
            0.5*(wvel[k-1]+wvel[k])
          else
            0.5*wvel[1]
          end if;
          thadv1:=  u[k,i,j]*(t[k,i+1,j]-t[k,i,j])
           +u[k,i-1,j]*(t[k,i,j]-t[k,i-1,j]);
          thadv2:=  v[k,i,j]*(t[k,i,j+1]-t[k,i,j])
           +v[k,i,j-1]*(t[k,i,j]-t[k,i,j-1]);
          thadv:= emthad*(thadv1+thadv2);
          tfull:= tflev[k]+tbar[k];
          phadv1:=  u[k,i,j]*(ps[i+1,j]-ps[i,j]]
           +u[k,i-1,j]*(ps[i,j]-ps[i-1,j]);
          phadv2:=  v[k,i,j]*(ps[i,j+1]-ps[i,j])
           +v[k,i,j-1]*(ps[i,j]-ps[i,j-1]);
          t_omg:= em2tps*(phadv1+phadv2);
          tpstar:= t_omg*tfull;
          omg:= wvelav+q[k]*psijc*t_omg*cp / r;
          gamapr:= (r / cp)*tfull / q[k]-dtfdq[k];
          thdiff:=
          if (dnorm[k]  =  0.0) then
            0.0
          else
            cdiff[i,j]*dnorm[k]*dsi2
             *( tm[k,i+1,j]+tm[k,i-1,j]
             +tm[k,i,j+1]+tm[k,i,j-1]-4.0*tm[k,i,j])
          end if;
%
          tp_k:= (tpstar-thadv+thdiff
           +(wvelav*gamapr+q[k]*dtfdq[k]*wvel[kz])*psijci
           -(wvelav*gama[k]+q[k]*dtodq[k]*wvel[kz]) / pbar
            )*dt+tm[k,i,j];
%
%         moisture
%
          rme:=rmpr[k,i,j]+rmpr[k,i+1,j];
          rmw:=rmpr[k,i,j]+rmpr[k,i-1,j];
          rmn:=rmpr[k,i,j]+rmpr[k,i,j+1];
          rms:=rmpr[k,i,j]+rmpr[k,i,j-1];
          rmvad:=
          if (k = kz) then
            0.0 %gke
          else
            -(vadvrm[k+1]-vadvrm[k]) / dq[k]
          end if;
          rmhad:=-emsq[i,j]*tdsi* ( u[k,i,j]*rme-u[k,i-1,j]*rmw
           +v[k,i,j]*rmn-v[k,i,j-1]*rms);
          rmme := rmm[k,i+1,j]/(psm[i+1,j]+pbar);
          rmmw := rmm[k,i-1,j]/(psm[i-1,j]+pbar);
          rmmn := rmm[k,i,j+1]/(psm[i,j+1]+pbar);
          rmms := rmm[k,i,j-1]/(psm[i,j-1]+pbar);
          rmmij := rmm[k,i,j]/(psm[i,j]+pbar);
          rmhdif:=
          if (dnorm[k]  =  0.0) then
            0.0
          else
            cdiff[i,j]*dnorm[k]*dsi2
             *(rmme+rmmw+rmmn+rmms-4.0*rmmij)
             *(psm[i,j]+pbar)
          end if;
          t_rmp_k:= rmm[k,i,j]+dtt*(rmhad+rmvad+rmhdif);

%         suppress negative mixing ratios
%
          rmp_k:=
          if (t_rmp_k < 1.0E-20) then
            0.0
          else
            t_rmp_k
          end if
        in
          rmp_k,
          tp_k,
          omg
        end let
      end if;
%
      vp_k,
      new_eke_k,
      new_ppe_k,
      new_pvromg_k:=
      if ((i = i2)|boundary_cell(i,i1,i1,j,j1,j1)) then
        vp[k,i,j],
        0.0, %eke
        0.0, %ppe
        0.0 %pvromg
      else
%
        let
          ua:=u[k,i-1,j]+u[k,i-1,j+1];
          ue:=u[k,i,j]+u[k,i,j+1];
          va:=v[k,i,j]+v[k,i-1,j];
          vb:=v[k,i,j]+v[k,i,j-1];
          ve:=v[k,i,j]+v[k,i+1,j];
          vf:=v[k,i,j]+v[k,i,j+1];
%
%         calculate v velocity component logarithmically
%
          ct2:=-corf2*(ua+ue);
          hadv2:= emhad2*( ue*ve*cycle_emonp-ua*va*new_amonp
           +vf*vf*fmonp-vb*vb*new_bmonp);
          pg2:= ((psn-pbar)*(phi[k,i,j+1]-phi[k,i,j])
           +zspdj-rtbar[k]*psrmj
           +psn*psldj* ( r*0.5* (t[k,i,j+1]+t[k,i,j])
           +rtbar[k]))*dsi;
          vbdiff:=
          if (dnorm[k]  =  0.0) then
            0.0
          else
            cdiff[i,j]*dnorm[k]*dsi2
             *( vm[k,i+1,j]/psmve+vm[k,i-1,j]/psmvw
             +vm[k,i,j+1]/psmvn+vm[k,i,j-1]/psmvs
             -4.0*vm[k,i,j]/psmv)*psmv
          end if;
          vat2:=
          if (k = kz) then
            0.0 %gke
          else
            -(vadvv[k+1]-vadvv[k])/dq[k]
          end if;
          t_vp:= (ct2+vat2-hadv2-pg2+vbdiff)*dt+vm[k,i,j]
%
%         calculate contributions to integrals
%
        in
          t_vp,
          dq[k]*psijci* (u[k,i,j]*u[k,i,j]+ v[k,i,j]*v[k,i,j]),
          tflev[k]*dq[k],
          (omega_k*omega_k)*dq[k]
        end let
      end if;
      returns
        array of tp_k
        array of up_k
        array of vp_k
        array of t_rmp_k
        array of omega_k
        value of sum (dq[k]*psijci*(up_k*up_k+vp_k*vp_k)) %new_eke_k
        value of sum (tflev[k]*dq[k]) %new_ppe_k
        value of sum (omega_k*omega_k*dq[k]) %new_pvromg_k
        value of sum (t_rmp_k*dq[k])%ptrhat
      end for; % k
      t_new_ptrhat := new_ptrhat+emsqi[i,j];
      t_new_epe := new_ppe*psijc*emsqi[i,j];
      t_new_vromg := new_pvromg*emsqi[i,j];
    returns
      array of tp_k
      array of up_k
      array of vp_k
      array of rmp_k
      array of sigdot_k
      value of sum new_eke
      value of sum (emsqi[i,j]) %new_cks
      value of sum (t_new_ptrhat*emsqi[i,j])
      value of sum t_new_epe
      value of sum ((psijc-0.988e6)*emsqi[i,j]) % psbar
      value of sum t_new_vromg
    end for % i,j
  in
    new_rm,
    new_sigdot,
    new_up,
    new_tp,
    new_rmp,
    new_vp,
    new_eke,
    new_cks,
```

```
      new_trhat,
      new_pe,
      new_psbar,
      new_romg
  end let
end function
```

References

[1] Leslie, L.M. et al., "A High Resolution Primitive Equations NWP Model for Operations and Research", pp 11-35, Australian Meteorological Magazine, No. 33, Mar 1985.

[2] Arakawa, A. and Lamb, V.R., "Computatioal Design of the Basic Dynamical Processes of the UCLA General Circulation Model," pp 174-256, 337, Methods of Computational Physics, Vol. 17, Academic Press, 1977.

[3] Miyakoda, K., "Cumulative Results of Testing a Meteorological-Mathematical Model," pp 99-130, Royal Irish Academy Proceedings, July 1973.

[4] McGraw et al., "SISAL: Streams and Iteration in a Single Assignment Language," Language Reference Manual Version 1.2, Lawrence Livermore National Laboratory, March 1, 1985.

[5] Feo, J.T., and Cann, D.C., "A Report on the SISAL Language Project," *Journal of Parallel and Distributed Computing*, Vol. 10, pp 349-366, 1990.

[6] Cray Research, "CF77 Compiling System Volume 4: Parallel Processing Guide," Cray Research, Incorporated, SG-3074 5.0, 1991.

[7] Cann, D.C., "The Optimising SISAL Compiler: Version 12.0," Computing Research Group, L-306, Lawrence Livermore National Laboratory, Livermore, 1992.