Implementació optimitzada del paquet de química computacional **Mopac 93** sobre arquitectures **RISC**.

F. Alted*

22 de Novembre de 1994

Resum

L'objectiu d'aquest treball ha estat de fer un estudi del paquet de química computacional Mopac 93, per tal d'obtenir una versió optimitzada que puga córrer de manera eficient sobre màquines **RISC**, com ara les HP 9000 sèrie 700, IBM RS6000, DEC ALPHA, etc... Les tècniques usades en aquesta implementació són anomenades de blocatge i desenrrotllament, així com d'altres que impliquen fer un ús eficient de la memòria *cache* present als processadors RISC. Els guanys en velocitat s'ilustren comparant amb la versió original del **Mopac 93**.

Mots clau : Química computacional, Arquitectures RISC, BLAS, FORTRAN

*Centre de Processament de Dades. Universitat Jaume I de Castelló

Índex

Agraïments

M'agradaria donar les gràcies en primer lloc a la Conselleria d'Educació i Ciència de la Generalitat Valenciana per haver-me donat l'oportunitat de dedicar-me a l'elaboració d'aquest treball. També a tots aquells qui han col·laborat en la complicada tasca burocràtica què representa enviar a un membre del Personal d'Administració i Serveis d'aquesta Universitat durant dos mesos al CERFACS amb finalitats de formació professional. En aquest sentit, agraïsc molt especialment el suport i recolzament que m'ha oferit Josep Miquel Castellet, director del Centre de Processament de Dades, per tal de véncer els entrebancs administratius per accedir a l'estància. També volguera fer extensiu aquest agraïment a Rogelio Montañana, analista del Servei d'Informàtica de la Universitat de València pel seu desinteressat suport en moltes de les facetes relacionades amb aquesta estada.

També, aquesta estància no haguera estat tan productiva de no ser pel magnífic ambient de companyerisme creat entre els diferents becaris que hi érem al CERFACS. El meu sincer agraïment per a Joao Lopes, amb el qual vaig treballar conjuntament en l'el·laboració d'aquest treball, i que va tenir la amabilitat i la paciència d'introduïr-me en el camp de la química computacional, i, en particular, en la interpretació física dels resultats obtinguts amb el Mopac 93. Gràcies a Wladimiro Diaz, pels seus extraordinaris consells tècnics relacionats amb gairebé qualsevol branca de la informàtica, així com també a Juan Zúñiga, sempre disposat a invitar-me al seu apartament per sopar i permitir-me poder apreciar així la seua excel·lent valia culinària, a més d'humana. Naturalment, molta més gent contribuí a que l'estància fóra molt més plàcida i tranquila, com ara Vicent Mas, Jose Antonio Font, Ricardo Quirós, el *colombiano*, etc...

Per últim, agrair a la meua Mercè la paciència amb la que m'ha hagut d'aguantar durant l'estància, i el seu constant suport. Espere al menys que li haja agradat l'experiència d'haver pogut poder conéixer paisatges tan bells com els de l'Occitània francesa, i haver deprés moltes coses sobre la interessant *heretgia càtara*, sobre tot, gràcies a les explicacions constants, però sempre encertades, d'un guia tan versat i amerat de cultura *càtara* com ara el nostre amic Juan Zúñiga.

1 Introducció

1.1 Objectiu de l'estada

La finalitat d'aquesta estada de dos mesos al **CERFACS** (Tolosa), ha estat desenvolupar una versió optimitzada del programa de química computacional Mopac 93 per a arquitectures **RISC** monoprocessador.

Mopac 93 és un paquet basat en tècniques de càlcul semi-empíric d'orbitals moleculars per a l'estudi d'estructures i reaccions químiques. El paquet fa ús dels hamiltonians semi-empírics **MNDO**, **MINDO/3**, **AM1** i **PM3** en la part d'interacció electrònica del càlcul per obtenir-ne, entre d'altres propietats, les energies dels orbitals moleculars, el calor de formació molecular i la seua derivada respecte de la geometria molecular (veure referència [1]).

Mopac 93 és un software que conté parts amb copyright de **Fujitsu Limited**, encara que es diu explícitament que no existeixen restriccions sobre la part del codi que forma part de prèvies versions del Mopac les quals són de domini públic. Val a dir que totes les modificacions que s'han efectuat sobre el codi original del **Mopac 93** *han estat fetes sobre la part de codi de domini públic*, i, qualsevol persona o grup que dispose del Mopac 93, podria usar aquest codi optimitzat sense violar cap copyright.

1.2 Importància en la portabilitat del codi entre diferents arquitectures

És molt important per a la longevitat d'un codi qualsevol que siga portable, çò és, que siga capaç de passar d'una màquina a una altra sense haver de patir una modificació gran per tal de reproduir els mateixos resultats en les dues plataformes. L'ús de llenguates standards permet açò, al menys en teoria. La realitat és, però, que les capacitats específiques de cada màquina s'oposa, en general, a la portabilitat dels codis.

Una de les possibilitats per tal de soluconar aquest problema, consisteix en fer ús de llibreries standards. Dintre del domini del càlcul numèric, la llibreria **BLAS** (Basic Linear Algebra Subroutines) constitueix la rajola de base per construir codis portables i potents alhora. En aquest sentit, s'ha intentat fer ús d'aquesta llibreria durant el procés d'optimització del Mopac 93.

1.3 Característiques generals dels processadors RISC

El concepte d'arquitectura RISC (Reduced Instruction Set Computer) neix de la preocupació per alcançar altes prestacions per a codis escrits en llenguatges d'alt nivell. Els processadors RISC moderns presenten les següents característiques :

- Instruccions executades en un cicle de rellotge
- Joc d'instruccions i modes d'adressament simplificats
- Memòria jerarquitzada d'altes prestacions
- Explotació del principi de "pipeline"
- Gran nombre de registres

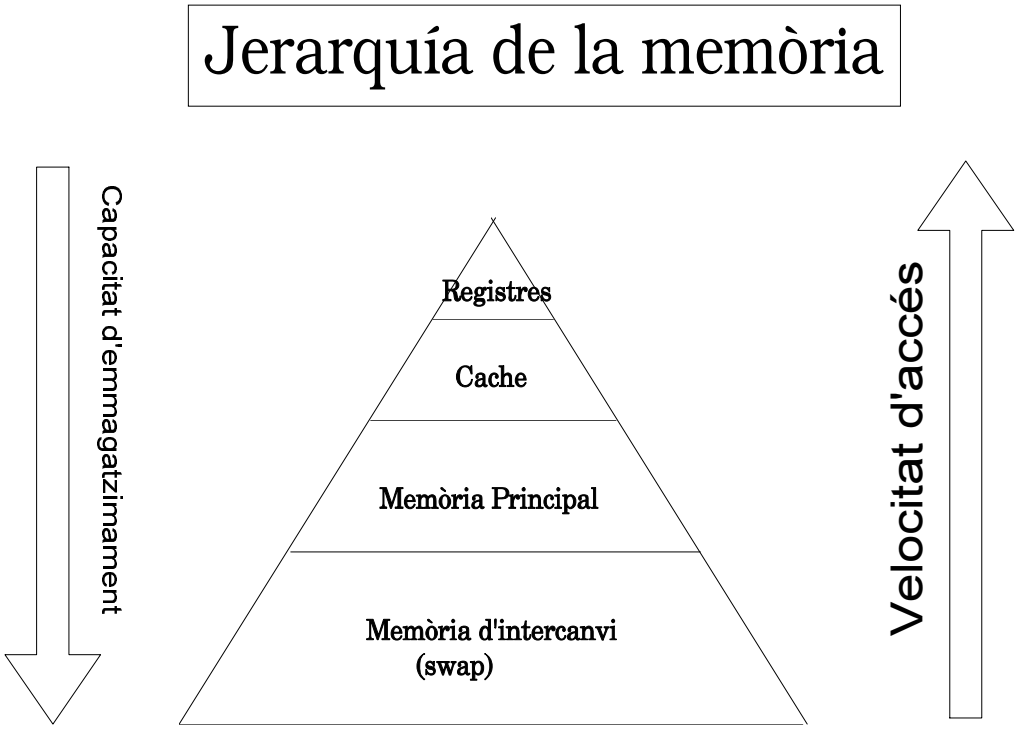

Figura 1: Jerarquia de la memòria

- Interface Load/Store amb la memòria (aquestes són les úniques operacions capaces de referenciar la memòria)
- Possibilitat d'execució de varies operacions en un mateix cicle de rellotge (arquitectura *superescalar*).

Fem notar que tenir en compte la jerarquia de la memòria de les arquitectures RISC és fonamental a l'hora d'usar-la eficientment. En efecte, les unitats funcionals del processadors RISC usent molt el principi de *"pipeline"* i, per tant, deuen ser alimentades molt ràpidament (per a que el *pipeline* no es quede bloquejat). Per tant, un bon algorisme de càlcul sobre màquines RISC ha de tenir molt en compte el fet de que és més ràpid accedir als registres del processador que a la *cache*, i més ràpid l'accés a aquesta última que no a la memòria central.

2 Motivacions generals

El temps d'accés a memòria és el factor que limita les prestacions de tots el processadors actuals, i en particular els RISC. En efecte, com acabem de recordar, aquests últims poden executar una instrucció (o dos o quatre, si l'arquitectura és superescalar) per cicle de rellotge. Això però, no és possible a menys que els operands siguen accesibles immediatament; aquest és el cas a sols si aquests estan ja als registres, o bé si se suministren al processador amb un retras molt breu, com per exemple, des de la *cache*. Així que, un dels objectius de l'optimització de codi és de mantenir les dades dins els registres i la *cache* el temps mes llarg possible. Per tant, és clar que el tamany de la *cache* i el nombre de registres escalars disponibles en un processador RISC poden limitar les possibilitats d'optimització d'un codi.

En la secció següent es presentaran les tècniques que ens han permés de fer un ús més eficient de la jerarquia de la memòria usada pels processadors RISC en general. Fem notar que, aquestes tècniques també són aplicables per a la implantació de codis eficients sobre processadors vectorials. En un apartat posterior farem una anàlisi quantitativa de les millores obtingudes gràcies a aquestes tècniques.

3 Programació òptima de codis d'àlgebra lineal sobre arquitectures RISC

3.1 Accés eficient a la memòria

Mopac 93 és una versió recent del Mopac, que és un software que va començar a desenvolupar-se en l'any 1967. Durant tot aquest temps han anat eixint versions del paquet, i, encara que el seu *kernel* de càlcul ha anat canviant des dels inicis, el que és cert és que arrastra molta metodologia de programació pròpia de les primeres arquitectures de les màquines de càlcul.

Per exemple, una de les característiques més significatives del codi és el seu èmfasi en estalviar memòria, com ara en l'emmagatzenament de les matrius simètriques de manera comprimida. Els algorismes, però, que inclou el Mopac 93 per a tractar àlgebra matricial amb aquestes matrius comprimides, no tenen en compte per a res la jerarquia de la memòria per a efectuar els càlculs, i per tant, no són gens eficients per a màquines RISC. Els optimitzadors dels compiladors no poden fer gran cosa per fer més eficients els algorismes degut a la complexitat intrínseca d'aquests.

Ací tenim un exemple per a ilustrar açò. Es tracta d'un algorisme extret del Mopac 93 i que efectua una multiplicació matriu-vector, on la matriu és simètrica i està emmagatzemada en forma compacta, i.e. els elements de la matriu es fiquen en un vector unidimensional ordenats per files.

```
        KK=0
        DO 30 J=1,N
           SUM=0.D0
           DO 10 K=1,J
              KK=KK+1
10         SUM=SUM+FAO(KK)*VECTOR(K,I)
           IF(J.EQ.N) GOTO 30
           J1=J+1
           K2=KK
           DO 20 K=J1,N
              K2=K2+K-1
20         SUM=SUM+FAO(K2)*VECTOR(K,I)
30      WS(J)=SUM
```

La matriu simètrica s'emmagatzema al vector **FAO**, i la matriu **VECTOR** és en realitat un conjunt de pseudo-vectors propis de **FAO**. El que fa l'algorisme és una operació matriu-vector del tipus **`WS=FAO*VECTOR(I)`**. En la figura 2 s'observa gràficament el mode d'accés a la matriu.

Com es veu, durant el procés de multiplicació matriu-vector el bucle 10 recorre la matriu **FAO** tot i accedint a posicions consecutives de memòria, i.e. fent un ús eficient de la *cache*, mentre que el bucle 20 hi accedeix a *bots*, fent molt car l'accés a les posicions de memòria.

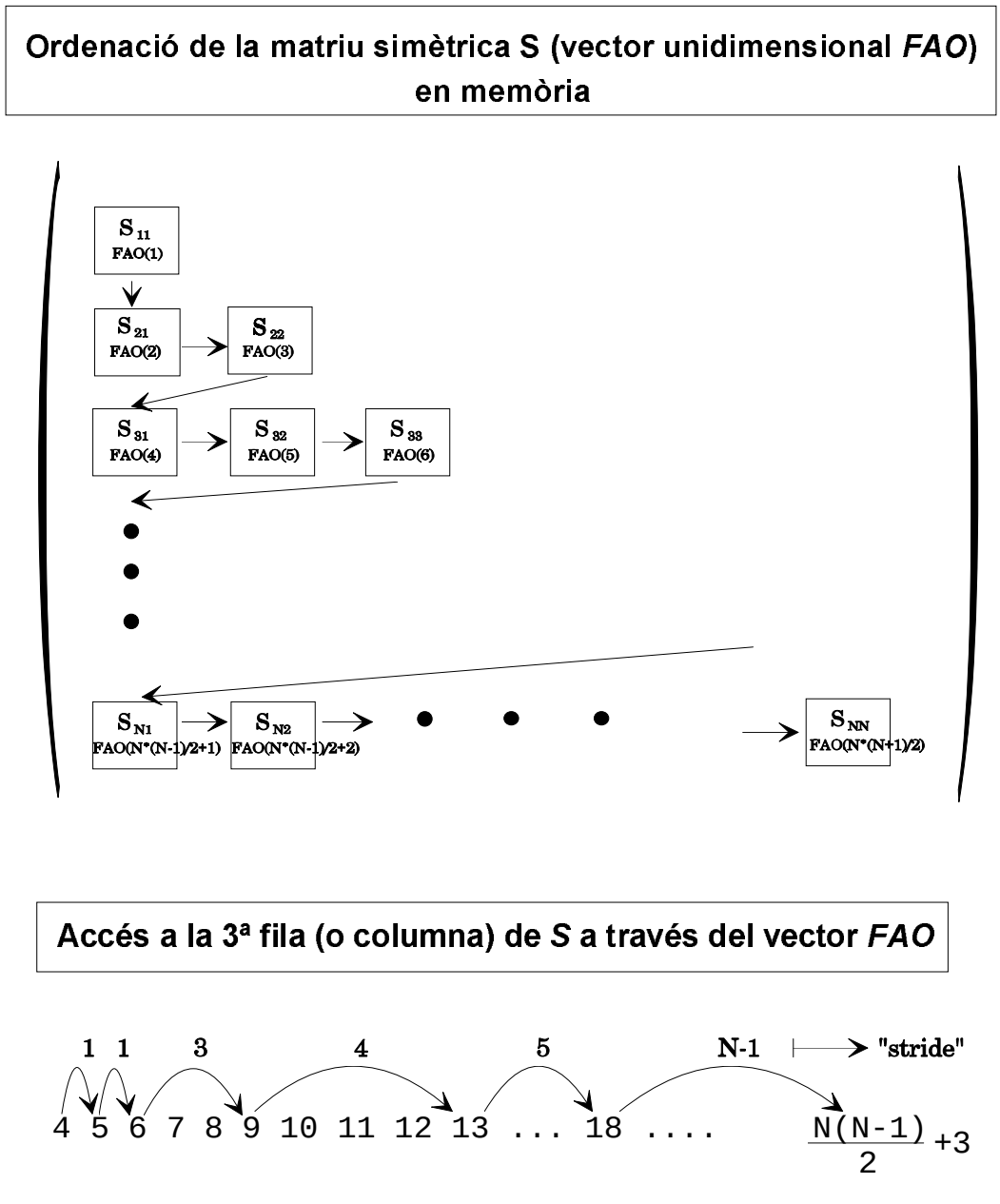

Figura 2: S'observa l'accés ineficient a la memòria

Una posible solució passa per desempaquetar la matriu abans de fer l'operació matriu-vector i reescriure l'algorisme tenint en compte les regles bàsiques d'accés eficient a memòria, i.e., accedint per columnes en el cas de FORTRAN. Ací es mostra el nou codi :

```
C Convertim la matriu FAO a una bidimensional autentica :
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          fao2(i,j) = fao(k)
          if (i .ne. j) fao2(j,i) = fao(k)
        end do
      end do

C Canvi de l'algorisme per a matriu completa
         DO 30 J=1,N
          SUM=0.D0
           DO 10 K=1,N
  10         SUM=SUM+FAO2(k,j)*VECTOR(K,I)
  30     WS(J)=SUM
```

Aquest nou algorisme ja accedeix de manera eficient a la memòria, a costa de :

1. Reservar memòria per a la forma completa de FAO

2. Usar CPU per a fer el desempaquetament.

La primera conseqüència no ens deuria de preocupar molt, ja que si be en l'època en què es va fer Mopac la memòria RAM era cara i era importantíssim el fet de poder estalviar-ne, avui en dia les baixades de preu d'aquesta fan que siga factible disposar d'importants quantitats de memòria en un únic sistema. Respecte al segon punt, el temps de CPU usat en desempaquetar la matriu es compensa de lluny en la millora de l'eficiència del nou algorisme.

Corrent els dos codis sobre una plataforma IBM RS/6000-950, amb tamanys de matriu de l'ordre de 100x100 hem trobat que el segon codi és un factor 1.5 més ràpid que el primer.

3.2 Desenrrotllament de bucles

El desenrrotllament de bucles no és una tènica nova i s'utilitza normalment pels compiladors de manera automàtica durant el procés de generació de codi màquina. L'interés d'aquest desenrrotllament de bucles és de permetre al processador, d'una banda, de fer un bon "pipilene" dels càlculs i també d'aprofitar el gran nombre de registres del processador, i d'altra de reduir els temps morts degut a les branques condicionals (*conditional branching*).

Encara que els compiladors actuals semblen relativament sofisticats, en general, no desenrrotllen més que el bucle més intern, ja que normalment hi existeix dependència entre ells. En aquest cas s'ha de recórrer a desenrrotllar els bucles de càlcul "*à la main*" per a remediar-ho.

Posem un exemple per a il·lustrar açò. Una operació molt comuna en qualsevol paquet de càlcul, i en particular en el Mopac 93, n'és l'operació matriu-matriu.

1. Sense desenrrotllament

```
      do 90,j=1,n
        do 80,i=1,m
          do 70,l=1,k
            c(i,j)=c(i,j)+a(i,l)*b(l,j)
          continue
        continue
      continue
```

2. Desenrrotllant

```
      do 70,j=1,n,2
        do 60,i=1,m,2
          t11=c(i,j)
         t21=c(i+1,j)
          t12=c(i,j+1)
          t22=c(i+1,j+1)
          do 50,l=1,k
            b1=b(l,j)
            b2=b(l,j+1)
            a1=a(i,l)
            a2=a(I+1,l)
            t1=b1*a1
            t2=b1*a2
            u1=b2*a1
            u2=b2*a2
            t11=t11+t1
            t21=t21+t2
            t12=t12+u1
            t22=t22+u2
50        continue
          c(i,j)=t11
          c(i+1,j)=t21
          c(i,j+1)=t21
          c(i+1,j+1)=t22
60      continue
70    continue
```

Analitzant els bucles més interns del codi d'amunt, podem constatar que:

- En el codi no desenrrotllat els elements de les matrius A i B, una vegada carregats, a sols s'usen una vegada, o siga, no hi ha *reusabilitat* de les dades. Çò és, els elements de A (o de B) són carregats **`M*N*K`** vegades.
- En el codi desenrrotllat, els elements de les matrius A i B s'usen dues voltes una vegada carregats. Quantitativament parlant, els elements de A (o B), són carregats **`M*N*K/2`** vegades. Així mateix, els elements de la matriu C són carregades una vegada i utilitzats dues. Remarquem també que s'usen operacions registre-registre, molt més ràpides que les operacions memòria-memòria en un RISC.

Per tant, podem preveure un guany de velocitat d'un factor 2. A la taula 1 tenim dades reals sobre increments de velocitat obtingudes sobre un processador IBM RS/6000-550 (veure referències [2], [3]).

M=N=K	Codi desenrrotllat	Codi no desenrrotllat	Guany
64	53.2	26.62	2.0
128	17.6	6.0	2.92
192	17.1	6.1	2.83
256	16.3	5.69	2.86

Taula 1: Comparació de prestacions (en MFlops) dels dos codis

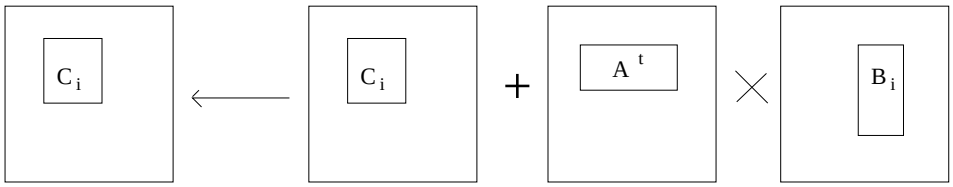

Figura 3: Bloqueig aplicat a multiplicació matriu-matriu

3.3 El blocatge

Quan s'intenta escriure un codi eficient des del punt de vista de càlcul, és necessari de procedir de manera que el número de defectes de *cache* (*cache missing*) siga el més petit possible. En efecte, els processadors RISC en general accedeixen a les seues memòries centrals per mitjà d'una memòria *cache*. Quan el processador intenta accedir a una dada i no existeix a la *cache* (defecte de *cache*) el temps d'accés a la memòria central és normalment un ordre de magnitud superior que si aquesta dada hi estigués (10 cicles de rellotge contra 1, tipicament). És necessari, doncs, mantenir les dades dins la *cache* utilitzant-les mentre siga possible per tal d'esmortir la latència que el processador ha hagut de pagar per tal de carregar-les.

Les memòries *cache* tenen, en general, una estratègia de reemplaçament de dades del tipus **LRE** (*Least Recently Used*). Així que una bona estratègia per tal de mantenir més temps una dada dins la *cache*, és referenciar-la freqüentment per a que no esdevinga mai la *menys recentment utilitzada*. Açò és el que s'anomena *localitat temporal*. També, s'ha de reduir el número de dades manipulades per què totes puguen cohabitar en la *cache* amb el mínim de conflictes possible. A la figura 3 donem un esquema que ilustra la tècnica de blocatge per a una multiplicació matriu-matriu (més addició).

És evident que, per a una talla òptima **NB** del tamany de bloc (que, òbviament, dependrà del tamany de la *cache*), el número de defectes de *cache* serà mínim, i com a conseqüència el número de dades referenciades en memòria; aleshores podrem dir que tenim una bona localitat temporal. Per tal de trobar aquest tamany òptim la solució ideal és recórrer a les llibreries BLAS ja optimitzades per a cada plataforma, i que normalment són suministrades pel fabricant.

Usant tècniques de desenrrotllament i blocatge es pot arribar a millorar de manera espectacular les prestacions de les rutines BLAS (i sobre tot el nivell 3 de BLAS). A la taula 2 tenim una comparació de les prestacions de la rutina **DGEMM** (que efectua operacions matricials tipus $C \longleftarrow C + A \times B$) BLAS standard respecte del BLAS

optimitzat amb aquestes tècniques (veure [2]).

Processador	Rutina	Tamany de les matrius			
		64	128	192	256
HP 730	DGEMM standard	23.2	27.8	22.5	13.7
	DGEMM bloquejat	33.3	41.9	43.8	44.7
DEC 3000/300-AXP	DGEMM standard	14.7	12.4	10.3	10.3
	DGEMM bloquejat	49.6	44.9	43.7	41.9
IBM RS/6000-950	DGEMM standard	38.0	29.8	28.6	29.5
	DGEMM bloquejat	53.2	57.1	58.3	58.9
SUN SPARC 10/41	DGEMM standard	12.7	12.1	12.1	11.1
	DGEMM bloquejat	18.1	19.0	18.6	17.6

Taula 2: Eficiència, en MFlops, de la implementació bloquejada de DGEMM sobre processadors RISC

4 Profiling i descripció esquemàtica de les optimitzacions efectuades sobre el codi del Mopac 93

Mopac 93 es passa gran part del temps efectuant àlgebra matricial de distint caire. Açò s'ha determinat amb l'ajuda dels anomentats *profilers*, çò és, software que ens dóna informació sobre el temps que consumeix cadascuna de les rutines que formen part de la nostra aplicació de càlcul.

Hi ha dos profilers que existeixen en la majoria de plataformes UNIX comercialitzades actualment, el *prof* i el *gprof*. Aquests tenen el desavantatge de que funcionen sobre versions *no optimitzades* de l'aplicació en qüestió. Tot i que la informació que proporcionen *gprof* i *prof* és molt útil, el fet de que actuen sobre codi no optimitzat ens pot encobrir el vertader comportament de l'aplicació en l'ús real, o siga, optimitzada.

Per això, s'ha preferit d'escollir el profiler *tprof* que ve conjuntament amb el sistema operatiu AIX de IBM, i que, tot i seguint una metodologia de mostreig (*sampling*), permet d'obtenir informació sobre temps d'execució de rutines d'aplicacions optimitzades.

La manera de funcionament del tprof és senzilla. L'usuari especifica al tprof el nom de programa a ser processat, i el tprof l'executa i després produeix un conjunt de fitxers que contenen informes de diferent caire. En el sistema operatiu AIX es produeix periodicament una interrupció per a permitir a una rutina de gestió interna del kernel (*"housekeeping" kernel routine*) que s'execute. Aquesta gestió interna es fa 100 vegades per segon. Quan s'invoca el comando tprof, la rutina de gestió interna del kernel grava l'identificador del procés (*process ID*) i l'adressa de la instrucció en execució quan ocurreix la interrupció. Amb la informació de l'adressa de la instrucció i l'identificador del procés, les rutines d'anàlisi del tprof poden carregar temps de CPU a processos, a subrutines i inclús a línies de programa (encara que, en la meua experiència, aquesta última capacitat no funciona bé).

Per tal de veure sobre quines rutines s'havia de centrar la nostra atenció a l'hora de fer l'optimització es varen triar 5 inputs (veure apèndix A per a un llistat dels inputs escollits), i, a partir del profiling del Mopac amb cadascun de ells, es varen determinar quines eren les rutines que més CPU consumien, o per parlar amb més propietat, aquelles subrutines on l'apuntador de fluix d'execució de programa (PC, Program Counter) es passava més temps. Ací a sota tenim un exemple d'un tros d'informació interessant d'una les eixides del programa tprof aplicat al Mopac per l'input numero 5 (05.inp) :

```
.
.
.
  Total Ticks For mopac.exe(    USER) =  70082

         Subroutine   Ticks      %     Source   Address  Bytes
       =============  ======   ======    =======   =======  =====
               .diag   21191     30.2    diag.f    432052   1088
                 .ss   10675     15.2    ss.f      199492   2000
           ._xldipow    8089     11.5   _xldipow.f 199164    328
             .densit    5174      7.4    densit.f  419672    496
             .interp    2864      4.1    interp.f  421876   5496
                .bfn    2643      3.8    bfn.f     201492    760
              .daxpy    2573      3.7    rsp.f     141156    440
               .ddot    2129      3.0    rsp.f     140732    424
                .jab    1554      2.2    jab.f     238220   3536
             .mamult    1374      2.0    mamult.f  431612    440
.
.
.
```

on, un tick representa 1 centèsima de segon de CPU.

Com veiem, les rutines que més CPU consumeixen en aquest cas són : *diag*, *ss*, *_xldipow*, etc... Les rutines que comencen per un caràcter de subratllat són del sistema, i en principi, no es poden optimitzar, i ens haurem de centrar en les demés.

Com que el llistat de rutines pot ser molt gran, s'ha fet un petit programa que, a més d'eliminar informació no desitjada, permet de tallar la llista quan s'ultrapasse un cert percentatge de CPU que es demana interactivament a l'usuari. També es calcula l'error comés en la mesura del temps, simplement agafant l'arrel quadrada (el mètode de mostreig temporal fa que l'estadística aplicable siga la de *Poisson*) del número de ticks per cada rutina. Açò permet de poder interpretar de manera molt més ràpida i efectiva la informació suministrada per *tprof*. Ací tenim un exemple de l'eixida de tprof processada amb aquest programa :

```
  Total Ticks For mopac.exe(    USER) =  70082 Run : mopac.05.tprof
Suma total de ticks de les rutines : 70082 . Ratio representat: 83.4 %

         Subroutine   Ticks      %           Source   Address  Bytes
       =============  ======  ======        =======   =======  =====
               .diag   21191    30.2±0.7      diag.f    432052   1088
                 .ss   10675    15.2±1.0        ss.f    199492   2000
           ._xldipow    8089    11.5±1.1  _xldipow.f    199164    328
             .densit    5174     7.4±1.4    densit.f    419672    496
             .interp    2864     4.1±1.9    interp.f    421876   5496
                .bfn    2643     3.8±1.9       bfn.f    201492    760
              .daxpy    2573     3.7±2.0       rsp.f    141156    440
```

```
          .ddot    2129     3.0±2.2       rsp.f    140732    424
           .jab    1554     2.2±2.5       jab.f    238220   3536
        .mamult    1374     2.0±2.7    mamult.f    431612    440
```

Una vegada fets els *profiles* dels diferents inputs, hem procedit a l'optimització de les principals rutines que més CPU consumeixen. Farem notar que, gairebé totes aquestes rutines tenen càlculs d'àlgebra lineal, per la qual cosa, la nostra tasca es reduirà a detectar les possibles cridades a BLAS dins del codi, tot i modificant l'algorisme de càlcul lleugerament si s'escau.

Passem a descriure ara les distintes optimitzacions efectuades al **Mopac 93**. Farem notar també que, com que la plataforma de desenvolupament ha estat una IBM RS/6000-950, totes les xifres de consum de CPU i speed-ups estaran referides a aquesta plataforma si no es diu el contrari.

4.1 Rutina diag

Part de l'optimització feta per a aquesta subrutina ha estat explicada en l'apartat **3.1**, encara que ací farem la descripció completa de tots els afinaments aconseguits.

El bucle principal de diag és el seguent :

```
   DO 60 I=LUMO,N
      KK=0
      DO 30 J=1,N
         SUM=0.D0
         DO 10 K=1,J
            KK=KK+1
10       SUM=SUM+FAO(KK)*VECTOR(K,I)
         IF(J.EQ.N) GOTO 30
         J1=J+1
         K2=KK
         DO 20 K=J1,N
            K2=K2+K-1
20       SUM=SUM+FAO(K2)*VECTOR(K,I)
30    WS(J)=SUM
      DO 50 J=1,NOCC
         IJ=IJ+1
         SUM=0.D0
         DO 40 K=1,N
40       SUM=SUM+WS(K)*VECTOR(K,J)
         IF(TINY.LT.ABS(SUM)) TINY=ABS(SUM)
50    FMO(IJ)=SUM
60 CONTINUE
```

on, el que es fa bàsicament és multiplicar la matriu simètrica en forma comprimida FAO, per un conjunt de pseudo-vectors propis seu. La manera d'accelerar el procés de multiplicació ha passat per varies etapes :

1. Posar la matriu FAO en forma completa per després accedir-hi a la matriu de manera eficient. Aquest procés ja ha estat detallat més amunt i no insisterem més en la seua justificació. Després de fer aquesta modificació, l'algorisme queda així :

```
C Convertim la matriu FAO a una bidimensional autentica :
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          fao2(i,j) = fao(k)
          if (i .ne. j) fao2(j,i) = fao(k)
        end do
      end do

C       canvi de l'algorisme per a matriu completa
      DO 60 I=LUMO,N
         DO 30 J=1,N
          SUM=0.D0
           DO 10 K=1,N
  10         SUM=SUM+FAO2(k,j)*VECTOR(K,I)
  30     WS(J)=SUM
         DO 50 J=1,NOCC
            IJ=IJ+1
            SUM=0.D0
            DO 40 K=1,N
   40       SUM=SUM+WS(K)*VECTOR(K,J)
            IF(TINY.LT.ABS(SUM)) TINY=ABS(SUM)
   50    FMO(IJ)=SUM
   60 CONTINUE
```

Aquest canvi d'algorisme du a una millora en la velocitat de la rutina diag que queda palesa al seu profile :

```
   Total Ticks For COmopac.exe(    USER) =  62294 Run : COmopac.05.tprof
 Suma total de ticks de les rutines : 62294 . Ratio representat: 81.4 %

          Subroutine   Ticks     %           Source   Address  Bytes
        =============  ======  ======        =======   =======  =====
                .diag   13879    22.3±0.8    COdiag.f    432052   1112
                  .ss   10785    17.3±1.0        ss.f    199492   2000
           ._xldipow    7847    12.6±1.1  _xldipow.f    199164    328
             .densit    5183     8.3±1.4    densit.f    419672    496
             .interp    2876     4.6±1.9    interp.f    421876   5496
                .bfn    2596     4.2±2.0       bfn.f    201492    760
              .daxpy    2459     3.9±2.0       rsp.f    141156    440
               .ddot    2159     3.5±2.2       rsp.f    140732    424
                .jab    1535     2.5±2.6       jab.f    238220   3536
             .mamult    1401     2.2±2.7    mamult.f    431612    440
```

o siga, la rutina diag que abans consumia 21200 ticks aprox, ara ha passat a consumir-ne 13900, el que significa un speed-up de 1.5.

2. Ús de la rutina *dgemv* (multiplicació matriu-vector) afinada per al sistema pel fabricant en compte del codi FORTRAN de dalt :

```
C Convertim la matriu FAO a una bidimensional autentica :
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          fao2(i,j) = fao(k)
          if (i .ne. j) fao2(j,i) = fao(k)
```

```
        end do
      end do

      one = 1.0d0
      zero = 0.0d0
      incx = 1

      DO 60 I=LUMO,N
      call dgemv('N',n,n,one,fao2,maxorb,vector(1,i),
     &    incx,zero,ws,incx)

         DO 50 J=1,NOCC
            IJ=IJ+1
            SUM=0.D0
            DO 40 K=1,N
40          SUM=SUM+WS(K)*VECTOR(K,J)
            IF(TINY.LT.ABS(SUM)) TINY=ABS(SUM)
50       FMO(IJ)=SUM
60    CONTINUE
```

Aquesta modificació fa que entre en funcionament, bàsicament, la tècnica del desenrrotllament. El profiling ens diu que diag consumeix ara 11394 ticks, la qual cosa significa un speed-up de 1.84 sobre l'algorisme original.

3. Ús de les rutines *dgemm* i *dsymm* (multiplicació matriu-matriu) del BLAS 3. Aquesta modificació del codi s'ha pogut fer degut a que l'operació multiplicació matriu-vector es fa per a tots els pseudo-vectors propis, emmagatzemant els resultats en el vector temporal WS, per a tornar a multiplicar una altra vegada WS pels pseudo-vectors propis. Aquest procediment s'ha condensat en l'algorisme de sota, tot i usant una nova matriu temporal anomenada WS2 per tal d'emmagatzemar tots els vectors WS necessaris :

```
C Convertim la matriu FAO a una matriu triangular superior
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          fao2(j,i) = fao(k)
        end do
      end do
C      call ftimer('Desempaquetat de la FAO *********')
      one = 1.0d0
      zero = 0.0d0
      incx = 1
C Multipliquem matriu FAO pels vectors propis.
      call dsymm('L','U',n,n-nocc,one,fao2,maxorb,
     &  vector(1,lumo),mdim,zero,ws2,maxorb)
C Tornem a multiplicar la matriu WS per d'altres vectors
      call dgemm('T','N',n-nocc,nocc,n,one,ws2,maxorb,
     &  vector,mdim,zero,fmo2,maxorb)
C      call ftimer('SQUARING *********')

C       Mirem quin es el maxim valor absolut a fmo2
      IJ=0
      DO 60 I=1,N-NOCC
```

```
        DO 50 J=1,NOCC
           IJ=IJ+1
           sum=fmo2(i,j)
           IF(TINY.LT.ABS(SUM)) TINY=ABS(SUM)
  50     FMO(IJ)=SUM
 60    continue
```

Al codi de dalt es poden observar dues coses :

- El fet de que la matriu FAO siga simètrica ha permés d'usar la rutina *dsymm* en compte de *dgemm* (normalment les implementacions de *dsymm* són més ràpides que les de *dgemm*), al temps que disminueix el temps de desempaquetament de la matriu FAO (a sols omplim els elements de FAO2 que estan sota la diagonal principal).
- Al final hem aprofitat el bucle de reempaquetament de la matriu resultat FMO, per tal d'obtenir l'element de matriu de màxim valor absolut i depositar-lo a la variable TINY, que ens exigia l'algorisme anterior.

L'ús de rutines BLAS 3 fa que s'activen totes les optimitzacions que hem relacionat adés, i.e. accés òptim a la memòria, desenrrotllament de bucles i, per fi, el blocatge (entre d'altres). Degut a açò el temps d'execució de la rutina deuria ésser més reduit encara que abans. Açò ens ho confirma el profiling : la rutina *diag* consumeix a sols 6194 ticks, la qual cosa significa un speed-up de 3.2 sobre el codi original.

Com a conseqüència d'aquesta última millora i per a aquest input (05.dat), s'ha aconseguit un speed-up de 1.27 per al programa sencer Mopac 93. Per altra banda, aquesta millora en la velocitat requereix reservar espai en memòria per a tres matrius més : FAO2, WS2 i FMO2, la qual cosa implica, per a un mopac compilat per a 45 atoms pesats, uns 2.3 MB aproximadament més de memòria principal per tal poder-les emmagatzemar. Veure apartat **5.1** per a discussió sobre ús addicional de memòria.

4.2 Rutina densit

En aquesta rutina s'efectua el càlcul d'una matriu per ella mateixa, tot i multiplicant per diferents constants depenent de quina columna de la matriu resultat s'estiga obtenint. Al BLAS existeix una rutina que efectua multiplicacions de matrius per elles mateixa. La rutina s'anomena SYRK, pertany al BLAS 3, i fa una operació de l'estil $C \longleftarrow \alpha \cdot A \cdot A^t + \beta \cdot C$.

Naturalment, al pertànyer SYRK al BLAS 3, totes les tècniques esmentades al capítol 3 seran aplicables a ella, i podem esperar per tant, un guany considerable en la velocitat.

Ací a sota tenim el codi original :

```
DO 40 I=1,NORBS
   DO 30 J=1,I
      L=L+1
      SUM2=0.D0
      SUM1=0.D0
```

```
            DO 10 K=NL2,NU2
   10       SUM2=SUM2+C(I,K)*C(J,K)
            SUM2=SUM2*2.D0
            DO 20 K=NL1,NU1
   20       SUM1=SUM1+C(I,K)*C(J,K)
   30    P(L)=(SUM2+SUM1*FRAC)*SIGN
   40 P(L)=CONST+P(L)
```

i ací el nostre usant BLAS :

```
        zero=0.d0
        one=1.d0
C       Efectuem la primera multiplicacio de la sub-matriu C :
       call dsyrk('U','N',NORBS,nu2-nl2+1,2.d0*sign,c(1,nl2),mdim,
      &  zero,c2,maxorb)
C       Ara la segona part :
       call dsyrk('U','N',NORBS,nu1-nl1+1,frac*sign,c(1,nl1),mdim,
      &  one,c2,maxorb)
C       Ara recuperem la matriu p, en la forma empaquetada :
       l=0
       do i=1,norbs
         do j=1,i
           l=l+1
           p(l)=c2(j,i)
         enddo
         p(l)=p(l)+const
       enddo
```

Com es veu, s'ha hagut de cridar dues vegades a la rutina *dsyrk* per tal de tenir en compte els diferents factors multiplicatius aplicats durant l'algorisme original. Al final, com que la matriu resultant és simètrica, s'ha aprofitat el bucle de reempaquetament per a afegir la constant **CONST** a la primera fila de la matriu **P**.

El profiling ens diu que aquesta modificació en la rutina densit fa que s'execute un factor 2.37 més ràpid que amb l'algorisme original.

4.3 Rutina mamult

La funció d'aquesta rutina és simplement calcular la multiplicació de dues matrius simètriques en forma comprimida i emmagatzemar el resultat en una altra de les mateixes característiques. Normalment, però, la multiplicació de dues matrius simètriques no necessàriament dòna lloc a una altra simètrica, però el codi en aquest sentit és clar : a sols es guarda la part triangular inferior de la matriu resultant. Això pot ser degut a varies causes, com per exemple que les matrius en els operands tinguen més propietats que asseguren la simetria de la matriu resultant, o simplement, que a l'autor del codi a sols li interesse guardar la part triangular inferior de la matriu resultant. En qualsevol cas, nosaltres usarem la rutina *dsymm* del BLAS 3 per tal d'accelerar els càlculs.

Codi original de mamult :

```
      L=0
      DO 40 I=1,N
         II=((I-1)*I)/2
         DO 40 J=1,I
```

```
            JJ=((J-1)*J)/2
            L=L+1
            SUM=0.D0
            DO 10 K=1,J
   10       SUM=SUM+A(II+K)*B(JJ+K)
            DO 20 K=J+1,I
   20       SUM=SUM+A(II+K)*B(((K-1)*K)/2+J)
            DO 30 K=I+1,N
               KK=(K*(K-1))/2
   30       SUM=SUM+A(KK+I)*B(KK+J)
   40 C(L)=SUM+ONE*C(L)
```

Codi usant la rutina *dsymm* de BLAS 3 :

```
C desempaquetem les matrius inicials :
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          a2(j,i) = a(k)
          b2(j,i) = b(k)
          c2(j,i) = c(k)
          if (i .ne. j) then
C A sols cal desempaquetar la part superior d'A
C            a2(j,i) = a(k)
            b2(i,j) = b(k)
            c2(i,j) = c(k)
          endif
        end do
      end do
C Efectuem multiplicacio matrius :
      one1 = 1.d0
C La matriu A es simetrica, aixi que utilitzem la BLAS3 dsymm  :
      call dsymm('L','U',n,n,one1,a2,maxorb,b2,maxorb,one,c2,maxorb)

C Tornem a empaquetar c2. A sols agafem la part triangular inferior.
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          c(k) = c2(i,j)
        end do
      end do
```

Aquesta senzilla modificació du a una millora en velocitat d'un factor 3.7 sobre el codi original de mamult.

4.4 Rutina interp

Aquesta és una rutina d'interpolació per a forçar la convergència del mètode **SCF** (*Self Consistent Field*). Es pot apreciar durant tota la rutina molt de codi dedicat a efectuar operacions d'àlgebra matricial. En particular, s'ha aconseguit de substituir codi en 8 llocs d'*interp* per cridades a rutines de BLAS.

Ací tenim el codi original d'*interp* :

```
      SUBROUTINE INTERP(N,NP,NQ,MODE,E,FP,CP,VEC,FOCK,P,H,VECL)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INCLUDE 'SIZES'
      DIMENSION FP(MPACK), CP(N,N)
      DIMENSION VEC(N,N), FOCK(N,N),
     1          P(N,N), H(N*N), VECL(N*N)
*****************************************************************
*
* INTERP: AN INTERPOLATION PROCEDURE FOR FORCING SCF CONVERGANCE
*         ORIGINAL THEORY AND FORTRAN WRITTEN BY R.N. CAMP AND
*         H.F. KING, J. CHEM. PHYS. 75, 268 (1981)
*****************************************************************
*
* ON INPUT N     = NUMBER OF ORBITALS
*          NP    = NUMBER OF FILLED LEVELS
*          NQ    = NUMBER OF EMPTY LEVELS
*          MODE  = 1, DO NOT RESET.
*          E     = ENERGY
*          FP    = FOCK MATRIX, AS LOWER HALF TRIANGLE, PACKED
*          CP    = EIGENVECTORS OF FOCK MATRIX OF ITERATION -1
*                  AS PACKED ARRAY OF N*N COEFFICIENTS
*
* ON OUTPUT CP   = BEST GUESSED SET OF EIGENVECTORS
*           MODE = 2 OR 3 - USED BY CALLING PROGRAM
*****************************************************************
      DIMENSION THETA(MAXORB)
      COMMON /KEYWRD/ KEYWRD
      COMMON /NUMCAL/ NUMCAL
      COMMON/FIT/NPNTS,IDUM2,XLOW,XHIGH,XMIN,EMIN,DEMIN,X(12),F(12),
     1 DF(12)
      COMMON /CHANEL/ IFILES(30)
      EQUIVALENCE(IW,IFILES(6))
      LOGICAL DEBUG
      CHARACTER*241 KEYWRD
      SAVE EOLD, XOLD, ZERO, FF, ICALCN, DEBUG
      DATA ICALCN/0/
      DATA ZERO,FF/0.0D0,0.9D0/
      DATA EOLD /0.D0/, XOLD /0.D0/
      IF(ICALCN.NE.NUMCAL)THEN
         DEBUG=(INDEX(KEYWRD,'INTERP').NE.0)
         ICALCN=NUMCAL
      ENDIF
C
C         FF=FACTOR FOR CONVERGENCE TEST FOR 1D SEARCH.
C
      MINPQ=MIN0(NP,NQ)
      NP1=NP+1
      NP2=MAX0(1,NP/2)
      IF(MODE.EQ.2) GO TO 100
C
C     (MODE=1 OR 3 ENTRY)
C     TRANSFORM FOCK MATRIX TO CURRENT MO BASIS.
C     ONLY THE OFF DIAGONAL OCC-VIRT BLOCK IS COMPUTED.
C     STORE IN FOCK ARRAY
C
      II=0
      DO 40 I=1,N
         I1=I+1
         DO 30 J=1,NQ
            DUM=ZERO
            DO 10 K=1,I
   10       DUM=DUM+FP(II+K)*CP(K,J+NP)
            IF(I.EQ.N) GO TO 30
            IK=II+I+I
            DO 20 K=I1,N
               DUM=DUM+FP(IK)*CP(K,J+NP)
```

```
   20       IK=IK+K
   30    P(I,J)=DUM
   40 II=II+I
      DO 70 I=1,NP
         DO 60 J=1,NQ
            DUM=ZERO
            DO 50 K=1,N
   50       DUM=DUM+CP(K,I)*P(K,J)
            FOCK(I,J)=DUM
   60    CONTINUE
   70 CONTINUE
      IF(MODE.EQ.3) GO TO 90
C
C     CURRENT POINT BECOMES OLD POINT (MODE=1 ENTRY)
C
      DO 80 I=1,N
         DO 80 J=1,N
   80 VEC(I,J)=CP(I,J)
      EOLD=E
      XOLD=1.0D0
      MODE=2
      RETURN
C
C     (MODE=3 ENTRY)
C     FOCK CORRESPONDS TO CURRENT POINT IN CORRESPONDING REPRESENTATION.
C     VEC DOES NOT HOLD CURRENT VECTORS. VEC SET IN LAST MODE=2 ENTRY.
C
   90 NPNTS=NPNTS+1
      IF(DEBUG)WRITE(IW,'(''   INTERPOLATED ENERGY:'',F13.6)')
     1E*23.060542301389D0
      IPOINT=NPNTS
      GO TO 490
C
C    (MODE=2 ENTRY) CALCULATE THETA, AND U, V, W MATRICES.
C                   U ROTATES CURRENT INTO OLD MO.
C                   V ROTATES CURRENT INTO CORRESPONDING CURRENT MO.
C                   W ROTATES OLD INTO CORRESPONDING OLD MO.
C
  100 J1=1
      DO 130 I=1,N
         IF(I.EQ.NP1) J1=NP1
         DO 120 J=J1,N
            P(I,J)=ZERO
            DO 110 K=1,N
  110       P(I,J)=P(I,J)+CP(K,I)*VEC(K,J)
  120    CONTINUE
  130 CONTINUE
C
C     U = CP(DAGGER)*VEC IS NOW IN P ARRAY.
C     VEC IS NOW AVAILABLE FOR TEMPORARY STORAGE.
C
      IJ=0
      DO 160 I=1,NP
         DO 150 J=1,I
            IJ=IJ+1
            H(IJ)=0.D0
            DO 140 K=NP1,N
  140       H(IJ)=H(IJ)+P(I,K)*P(J,K)
  150    CONTINUE
  160 CONTINUE
      CALL RSP(H,NP,NP,THETA,VECL)
      DO 170 I=NP,1,-1
         IL=I*NP-NP
         DO 170 J=NP,1,-1
  170 VEC(J,I)=VECL(J+IL)
      DO 190 I=1,NP2
```

```
         DUM=THETA(NP1-I)
         THETA(NP1-I)=THETA(I)
         THETA(I)=DUM
         DO 180 J=1,NP
            DUM=VEC(J,NP1-I)
            VEC(J,NP1-I)=VEC(J,I)
  180    VEC(J,I)=DUM
  190 CONTINUE
      DO 200 I=1,MINPQ
         THETA(I)=MAX(THETA(I),ZERO)
         THETA(I)=MIN(THETA(I),1.D0)
  200 THETA(I)=ASIN(SQRT(THETA(I)))
C
C     THETA MATRIX HAS NOW BEEN CALCULATED, ALSO UNITARY VP MATRIX
C     HAS BEEN CALCULATED AND STORED IN FIRST NP COLUMNS OF VEC MATRIX.
C     NOW COMPUTE WQ
C
      DO 230 I=1,NQ
         DO 220 J=1,MINPQ
            VEC(I,NP+J)=ZERO
            DO 210 K=1,NP
  210       VEC(I,NP+J)=VEC(I,NP+J)+P(K,NP+I)*VEC(K,J)
  220    CONTINUE
  230 CONTINUE
      CALL SCHMIT(VEC(1,NP1),NQ,N)
C
C     UNITARY WQ MATRIX NOW IN LAST NQ COLUMNS OF VEC MATRIX.
C     TRANSPOSE NP BY NP BLOCK OF U STORED IN P
C
      DO 250 I=1,NP
         DO 240 J=1,I
            DUM=P(I,J)
            P(I,J)=P(J,I)
  240    P(J,I)=DUM
  250 CONTINUE
C
C     CALCULATE WP MATRIX AND HOLD IN FIRST NP COLUMNS OF P
C
      DO 290 I=1,NP
         DO 260 K=1,NP
  260    H(K)=P(I,K)
         DO 280 J=1,NP
            P(I,J)=ZERO
            DO 270 K=1,NP
  270       P(I,J)=P(I,J)+H(K)*VEC(K,J)
  280    CONTINUE
  290 CONTINUE
      CALL SCHMIB(P,NP,N)
C
C     CALCULATE VQ MATRIX AND HOLD IN LAST NQ COLUMNS OF P MATRIX.
C
      DO 330 I=1,NQ
         DO 300 K=1,NQ
  300    H(K)=P(NP+I,NP+K)
         DO 320 J=NP1,N
            P(I,J)=ZERO
            DO 310 K=1,NQ
  310       P(I,J)=P(I,J)+H(K)*VEC(K,J)
  320    CONTINUE
  330 CONTINUE
      CALL SCHMIB(P(1,NP1),NQ,N)
C
C     CALCULATE (DE/DX) AT OLD POINT
C
      DEDX=ZERO
      DO 360 I=1,NP
```

```
         DO 350 J=1,NQ
            DUM=ZERO
            DO 340 K=1,MINPQ
  340       DUM=DUM+THETA(K)*P(I,K)*VEC(J,NP+K)
  350    DEDX=DEDX+DUM*FOCK(I,J)
  360 CONTINUE
C
C     STORE OLD POINT INFORMATION FOR SPLINE FIT
C
      DEOLD=-4.0D0*DEDX
      X(2)=XOLD
      F(2)=EOLD
      DF(2)=DEOLD
C
C     MOVE VP OUT OF VEC ARRAY INTO FIRST NP COLUMNS OF P MATRIX.
C
      DO 370 I=1,NP
         DO 370 J=1,NP
  370 P(I,J)=VEC(I,J)
      K1=0
      K2=NP
      DO 400 J=1,N
         IF(J.EQ.NP1) K1=NP
         IF(J.EQ.NP1) K2=NQ
         DO 390 I=1,N
            DUM=ZERO
            DO 380 K=1,K2
  380       DUM=DUM+CP(I,K1+K)*P(K,J)
  390    VEC(I,J)=DUM
  400 CONTINUE
C
C     CORRESPONDING CURRENT MO VECTORS NOW HELD IN VEC.
C     COMPUTE VEC(DAGGER)*FP*VEC
C     STORE OFF-DIAGONAL BLOCK IN FOCK ARRAY.
C
  410 II=0
      DO 450 I=1,N
         I1=I+1
         DO 440 J=1,NQ
            DUM=ZERO
            DO 420 K=1,I
  420       DUM=DUM+FP(II+K)*VEC(K,J+NP)
            IF(I.EQ.N) GO TO 440
            IK=II+I+I
            DO 430 K=I1,N
               DUM=DUM+FP(IK)*VEC(K,J+NP)
  430       IK=IK+K
  440    P(I,J)=DUM
  450 II=II+I
      DO 480 I=1,NP
         DO 470 J=1,NQ
            DUM=ZERO
            DO 460 K=1,N
  460       DUM=DUM+VEC(K,I)*P(K,J)
            FOCK(I,J)=DUM
  470    CONTINUE
  480 CONTINUE
C
C     SET LIMITS ON RANGE OF 1-D SEARCH
C
      NPNTS=2
      IPOINT=1
      XNOW=ZERO
      XHIGH=1.570796326795D0/THETA(1)
C
C     1.5708 IS MAXIMUM ROTATION ANGLE (90 DEGREE = 3.14159/2 RADIAN).
```

```
C
      XLOW=-0.5D0*XHIGH
C
C     CALCULATE (DE/DX) AT CURRENT POINT AND
C     STORE INFORMATION FOR SPLINE FIT
C     ***** JUMP POINT FOR MODE=3 ENTRY *****
C
  490 DEDX=ZERO
      DO 500 K=1,MINPQ
  500 DEDX=DEDX+THETA(K)*FOCK(K,K)
      DENOW=-4.0D0*DEDX
      ENOW=E
      IF(IPOINT.LE.12) GO TO 520
  510 FORMAT(//,'EXCESSIVE DATA PNTS FOR SPLINE.',/
     1,'IPOINT =',I3,'MAXIMUM IS 12.')
C
C     PERFORM 1-D SEARCH AND DETERMINE EXIT MODE.
C
  520 X(IPOINT)=XNOW
      F(IPOINT)=ENOW
      DF(IPOINT)=DENOW
      CALL SPLINE
      IF((EOLD-ENOW).GT.FF*(EOLD-EMIN).OR.IPOINT.GT.10) GO TO 550
C
C     (MODE=3 EXIT) RECOMPUTE CP VECTORS AT PREDICTED MINIMUM.
C
      XNOW=XMIN
      DO 540 K=1,MINPQ
         CK=COS(XNOW*THETA(K))
         SK=SIN(XNOW*THETA(K))
         IF(DEBUG)WRITE(IW,'('' ROTATION ANGLE:'',F12.4)')SK*57.29578
         DO 530 I=1,N
            CP(I,K)   =CK*VEC(I,K)-SK*VEC(I,NP+K)
  530    CP(I,NP+K)=SK*VEC(I,K)+CK*VEC(I,NP+K)
  540 CONTINUE
      MODE=3
      RETURN
C
C     (MODE=2 EXIT) CURRENT VECTORS GIVE SATISFACTORY ENERGY IMPROVEMENT
C     CURRENT POINT BECOMES OLD POINT FOR THE NEXT 1-D SEARCH.
C
  550 IF(MODE.EQ.2) GO TO 570
      DO 560 I=1,N
         DO 560 J=1,N
  560 VEC(I,J)=CP(I,J)
      MODE=2
  570 ROLD=XOLD*THETA(1)*57.29577951308232D0
      RNOW=XNOW*THETA(1)*57.29577951308232D0
      RMIN=XMIN*THETA(1)*57.29577951308232D0
      IF(DEBUG)WRITE(IW,590) XOLD,EOLD*23.060542301389D0,DEOLD,ROLD
     1,             XNOW,ENOW*23.060542301389D0,DENOW,RNOW
     2,             XMIN,EMIN*23.060542301389D0,DEMIN,RMIN
      EOLD=ENOW
      IF(NPNTS.LE.200) RETURN
      WRITE(IW,600)
      DO 580 K=1,NPNTS
  580 WRITE(IW,610) K,X(K),F(K),DF(K)
      WRITE(IW,620)
      RETURN
  590 FORMAT(
     1/14X,3H X ,10X,6H F(X) ,9X,7H DF/DX ,21H   ROTATION (DEGREES),
     2/10H      OLD ,F10.5,3F15.10,
     3/10H  CURRENT ,F10.5,3F15.10,
     4/10H PREDICTED,F10.5,3F15.10/)
  600 FORMAT(3H  K,10H     X(K) ,15H       F(K)    ,10H     DF(K))
  610 FORMAT(I3,F10.5,2F15.10)
```

```
  620 FORMAT(10X)
      END
```

I ací les substitucions per rutines BLAS 3 efectuades sobre el codi anterior :

```
      SUBROUTINE INTERP(N,NP,NQ,MODE,E,FP,CP,VEC,FOCK,P,H,VECL)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INCLUDE 'SIZES'
      DIMENSION FP(MPACK), CP(N,N)
      DIMENSION VEC(N,N), FOCK(N,N),
     &  P(N,N), H(N*N), VECL(N*N)
      dimension fp2(maxorb,maxorb)

*********************************************************************
*
* INTERP: AN INTERPOLATION PROCEDURE FOR FORCING SCF CONVERGANCE
*         ORIGINAL THEORY AND FORTRAN WRITTEN BY R.N. CAMP AND
*         H.F. KING, J. CHEM. PHYS. 75, 268 (1981)
*********************************************************************
*
* ON INPUT N     = NUMBER OF ORBITALS
*          NP    = NUMBER OF FILLED LEVELS
*          NQ    = NUMBER OF EMPTY LEVELS
*          MODE  = 1, DO NOT RESET.
*          E     = ENERGY
*          FP    = FOCK MATRIX, AS LOWER HALF TRIANGLE, PACKED
*          CP    = EIGENVECTORS OF FOCK MATRIX OF ITERATION -1
*                  AS PACKED ARRAY OF N*N COEFFICIENTS
*
* ON OUTPUT CP   = BEST GUESSED SET OF EIGENVECTORS
*           MODE = 2 OR 3 - USED BY CALLING PROGRAM
*********************************************************************
      DIMENSION THETA(MAXORB)
      COMMON /KEYWRD/ KEYWRD
      COMMON /NUMCAL/ NUMCAL
      COMMON/FIT/NPNTS,IDUM2,XLOW,XHIGH,XMIN,EMIN,DEMIN,X(12),F(12),
     1 DF(12)
      COMMON /CHANEL/ IFILES(30)
      EQUIVALENCE(IW,IFILES(6))
      LOGICAL DEBUG
      CHARACTER*241 KEYWRD
      SAVE EOLD, XOLD, ZERO, FF, ICALCN, DEBUG
      DATA ICALCN/0/
      DATA ZERO,FF/0.0D0,0.9D0/
      DATA EOLD /0.D0/, XOLD /0.D0/
      IF(ICALCN.NE.NUMCAL)THEN
         DEBUG=(INDEX(KEYWRD,'INTERP').NE.0)
         ICALCN=NUMCAL
      ENDIF
C
C         FF=FACTOR FOR CONVERGENCE TEST FOR 1D SEARCH.
C
      MINPQ=MIN0(NP,NQ)
      NP1=NP+1
      NP2=MAX0(1,NP/2)
      IF(MODE.EQ.2) GO TO 100
C
C     (MODE=1 OR 3 ENTRY)
C     TRANSFORM FOCK MATRIX TO CURRENT MO BASIS.
C     ONLY THE OFF DIAGONAL OCC-VIRT BLOCK IS COMPUTED.
C     STORE IN FOCK ARRAY
C
*****
C       call ftimer('Begin modif 1')
      zero=0.d0
C desempaquetem la matriu inicial, a una triangular superior, ja que a
```

```
C la inferior BLAS3 no en fa referencia :
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          fp2(j,i) = fp(k)
        end do
      end do
C
C Efectuem multiplicacio matrius :
      one1 = 1.d0
C La matriu fp2 es simetrica, aixi que utilitzem la BLAS3 dsymm  :
      call dsymm('L','U',n,nq,one1,fp2,maxorb,cp(1,np+1),n,zero,p,n)
      call dgemm('T','N',np,nq,n,one1,cp,n,p,n,zero,fock,n)
C       call ftimer('End modif 1')
*********
      IF(MODE.EQ.3) GO TO 90
C
C     CURRENT POINT BECOMES OLD POINT (MODE=1 ENTRY)
C
      DO 80 I=1,N
         DO 80 J=1,N
   80 VEC(j,i)=CP(j,i)
      EOLD=E
      XOLD=1.0D0
      MODE=2
      RETURN
C
C     (MODE=3 ENTRY)
C     FOCK CORRESPONDS TO CURRENT POINT IN CORRESPONDING REPRESENTATION.
C     VEC DOES NOT HOLD CURRENT VECTORS. VEC SET IN LAST MODE=2 ENTRY.
C
   90 NPNTS=NPNTS+1
      IF(DEBUG)WRITE(IW,'(''   INTERPOLATED ENERGY:'',F13.6)')
     1E*23.060542301389D0
      IPOINT=NPNTS
      GO TO 490
C
C    (MODE=2 ENTRY) CALCULATE THETA, AND U, V, W MATRICES.
C                   U ROTATES CURRENT INTO OLD MO.
C                   V ROTATES CURRENT INTO CORRESPONDING CURRENT MO.
C                   W ROTATES OLD INTO CORRESPONDING OLD MO.
C
  100 continue
C       call ftimer('Begin orig 2')
      one1=1.d0
      zero=0.d0
      call dgemm('T','N',np,np,n,one1,cp,n,vec,n,zero,p,n)
      call dgemm('T','N',n,nq,n,one1,cp,n,vec(1,np1),n,zero,p(1,np1),n)
C      J1=1
C       DO 130 I=1,N
C          IF(I.EQ.NP1) J1=NP1
C          DO 120 J=J1,N
C             P(I,J)=ZERO
C             DO 110 K=1,N
C   110       P(I,J)=P(I,J)+CP(K,I)*VEC(K,J)
C   120    CONTINUE
C   130 CONTINUE
C       call ftimer('End orig 2')
C
C     U = CP(DAGGER)*VEC IS NOW IN P ARRAY.
C     VEC IS NOW AVAILABLE FOR TEMPORARY STORAGE.
C
      IJ=0
      DO 160 I=1,NP
         DO 150 J=1,I
```

```
            IJ=IJ+1
            H(IJ)=0.D0
            DO 140 K=NP1,N
  140       H(IJ)=H(IJ)+P(I,K)*P(J,K)
  150     CONTINUE
  160   CONTINUE
C       call ftimer('End orig 3')
      CALL RSP(H,NP,NP,THETA,VECL)
C       call ftimer('Begin orig 4')
      DO 170 I=NP,1,-1
         IL=I*NP-NP
         DO 170 J=NP,1,-1
  170 VEC(J,I)=VECL(J+IL)
      DO 190 I=1,NP2
         DUM=THETA(NP1-I)
         THETA(NP1-I)=THETA(I)
         THETA(I)=DUM
         DO 180 J=1,NP
            DUM=VEC(J,NP1-I)
            VEC(J,NP1-I)=VEC(J,I)
  180    VEC(J,I)=DUM
  190 CONTINUE
C       call ftimer('End orig 4')
      DO 200 I=1,MINPQ
         THETA(I)=MAX(THETA(I),ZERO)
         THETA(I)=MIN(THETA(I),1.D0)
  200 THETA(I)=ASIN(SQRT(THETA(I)))
C
C     THETA MATRIX HAS NOW BEEN CALCULATED, ALSO UNITARY VP MATRIX
C     HAS BEEN CALCULATED AND STORED IN FIRST NP COLUMNS OF VEC MATRIX.
C     NOW COMPUTE WQ
C
C       call ftimer('Begin orig 5')
      call dgemm('T','N',nq,minpq,np,one1,p(1,np1),n,
     &  vec,n,zero,vec(1,np1),n)
C       call ftimer('End orig 5')
      CALL SCHMIT(VEC(1,NP1),NQ,N)
C
C     UNITARY WQ MATRIX NOW IN LAST NQ COLUMNS OF VEC MATRIX.
C     TRANSPOSE NP BY NP BLOCK OF U STORED IN P
C
C       call ftimer('Begin orig 6')
C Copiem el block npxnp de p a la matriu auxiliar fp2
      do i=1,np
        do j=1,np
          fp2(j,i)=p(j,i)
        end do
      end do
C       Efectuem operacio fp2(dagger)*vec, i dipositem el resultat a p
      call dgemm('T','N',np,np,np,one1,fp2,maxorb,vec,n,zero,p,n)
C       DO 250 I=1,NP
C          DO 240 J=1,I
C             DUM=P(I,J)
C             P(I,J)=P(J,I)
C   240    P(J,I)=DUM
C   250 CONTINUE
C
C     CALCULATE WP MATRIX AND HOLD IN FIRST NP COLUMNS OF P
C
C       DO 290 I=1,NP
C          DO 260 K=1,NP
C   260    H(K)=P(I,K)
C          DO 280 J=1,NP
C             P(I,J)=ZERO
C             DO 270 K=1,NP
C   270       P(I,J)=P(I,J)+H(K)*VEC(K,J)
```

```
C   280    CONTINUE
C   290 CONTINUE
C       call ftimer('End orig 6')
      CALL SCHMIB(P,NP,N)
C
C     CALCULATE VQ MATRIX AND HOLD IN LAST NQ COLUMNS OF P MATRIX.
C
C       call ftimer('Begin orig 7')
C       Multipliquem usant dgemm :
      call dgemm('N','N',nq,nq,nq,one1,
     &  p(np1,np1),n,vec(1,np1),n,zero,p(1,np1),n)
C       DO 330 I=1,NQ
C          DO 300 K=1,NQ
C   300    H(K)=P(NP+I,NP+K)
C          DO 320 J=NP1,N
C             P(I,J)=ZERO
C             DO 310 K=1,NQ
C   310       P(I,J)=P(I,J)+H(K)*VEC(K,J)
C   320    CONTINUE
C   330 CONTINUE
C       call ftimer('End orig 7')
      CALL SCHMIB(P(1,NP1),NQ,N)
C
C     CALCULATE (DE/DX) AT OLD POINT
C
C       call ftimer('Begin orig 8')
C     Multipliquem p pels termes del vector theta. El resultat el clavem
C     una altra vegada en la part superior esquerra de p que no es
C     tornara a usar.
      do i=1,minpq
        do j=1,np
          p(j,i)=p(j,i)*theta(i)
        end do
      enddo
C     Ara multipliquem la matriu p per vec
      one1=1.d0
      call dgemm('N','T',np,nq,minpq,one1,p,n,
     &  vec(1,np1),n,zero,fp2,maxorb)
C     finalment, obtenim (DE/DX)
      dedx=zero
      do i=1,nq
        do j=1,np
          dedx=dedx+fp2(j,i)*fock(j,i)
        end do
      end do
C       DEDX=ZERO
C       DO 360 I=1,NP
C          DO 350 J=1,NQ
C             DUM=ZERO
C             DO 340 K=1,MINPQ
C   340       DUM=DUM+THETA(K)*P(I,K)*VEC(J,NP+K)
C   350    DEDX=DEDX+DUM*FOCK(I,J)
C   360 CONTINUE
C       call ftimer('End orig 8')
C
C     STORE OLD POINT INFORMATION FOR SPLINE FIT
C
      DEOLD=-4.0D0*DEDX
      X(2)=XOLD
      F(2)=EOLD
      DF(2)=DEOLD
C
C     MOVE VP OUT OF VEC ARRAY INTO FIRST NP COLUMNS OF P MATRIX.
C
C       call ftimer('Begin orig 9')
      DO 370 I=1,NP
```

```
        DO 370 J=1,NP
  370 P(j,i)=VEC(j,i)
      one1=1.d0
      zero=0.d0
      call dgemm('N','N',n,np,np,one1,cp,n,p,n,zero,vec,n)
      call dgemm('N','N',n,nq,nq,one1,cp(1,np1),n,p(1,np1),
     &  n,zero,vec(1,np1),n)
C       K1=0
C       K2=NP
C       DO 400 J=1,N
C          IF(J.EQ.NP1) K1=NP
C          IF(J.EQ.NP1) K2=NQ
C          DO 390 I=1,N
C             DUM=ZERO
C             DO 380 K=1,K2
C   380       DUM=DUM+CP(I,K1+K)*P(K,J)
C   390    VEC(I,J)=DUM
C   400 CONTINUE
C       call ftimer('End orig 9')
C
C     CORRESPONDING CURRENT MO VECTORS NOW HELD IN VEC.
C     COMPUTE VEC(DAGGER)*FP*VEC
C     STORE OFF-DIAGONAL BLOCK IN FOCK ARRAY.
C
*******
C       call ftimer('Begin modif 10')
      zero=0.d0
C desempaquetem la matriu inicial, a una triangular superior, ja que a
C la inferior BLAS3 no en fa referencia :
      k=0
      do i=1,n
        do j=1,i
          k=k+1
          fp2(j,i) = fp(k)
        end do
      end do
C Efectuem multiplicacio matrius :
      one1 = 1.d0
C La matriu fp2 es simetrica, aixi que utilitzem la BLAS3 dsymm  :
      call dsymm('L','U',n,nq,one1,fp2,maxorb,vec(1,np+1),n,zero,p,n)
      call dgemm('T','N',np,nq,n,one1,vec,n,p,n,zero,fock,n)
C       call ftimer('End modif 10')
***********
C
C     SET LIMITS ON RANGE OF 1-D SEARCH
C
      NPNTS=2
      IPOINT=1
      XNOW=ZERO
      XHIGH=1.570796326795D0/THETA(1)
C
C     1.5708 IS MAXIMUM ROTATION ANGLE (90 DEGREE = 3.14159/2 RADIAN).
C
      XLOW=-0.5D0*XHIGH
C
C     CALCULATE (DE/DX) AT CURRENT POINT AND
C     STORE INFORMATION FOR SPLINE FIT
C     ***** JUMP POINT FOR MODE=3 ENTRY *****
C
  490 DEDX=ZERO
      DO 500 K=1,MINPQ
  500 DEDX=DEDX+THETA(K)*FOCK(K,K)
      DENOW=-4.0D0*DEDX
      ENOW=E
      IF(IPOINT.LE.12) GO TO 520
  510 FORMAT(//,'EXCESSIVE DATA PNTS FOR SPLINE.',/
```

```
     1,'IPOINT =',I3,'MAXIMUM IS 12.')
C
C     PERFORM 1-D SEARCH AND DETERMINE EXIT MODE.
C
  520 X(IPOINT)=XNOW
      F(IPOINT)=ENOW
      DF(IPOINT)=DENOW
      CALL SPLINE
      IF((EOLD-ENOW).GT.FF*(EOLD-EMIN).OR.IPOINT.GT.10) GO TO 550
C
C     (MODE=3 EXIT) RECOMPUTE CP VECTORS AT PREDICTED MINIMUM.
C
C       call ftimer('Begin orig 11')
      XNOW=XMIN
      DO 540 K=1,MINPQ
         CK=COS(XNOW*THETA(K))
         SK=SIN(XNOW*THETA(K))
         IF(DEBUG)WRITE(IW,'('' ROTATION ANGLE:'',F12.4)')SK*57.29578
         DO 530 I=1,N
            CP(I,K)   =CK*VEC(I,K)-SK*VEC(I,NP+K)
  530    CP(I,NP+K)=SK*VEC(I,K)+CK*VEC(I,NP+K)
  540 CONTINUE
C       call ftimer('End orig 11')
      MODE=3
      RETURN
C
C     (MODE=2 EXIT) CURRENT VECTORS GIVE SATISFACTORY ENERGY IMPROVEMENT
C     CURRENT POINT BECOMES OLD POINT FOR THE NEXT 1-D SEARCH.
C
  550 IF(MODE.EQ.2) GO TO 570
C       call ftimer('Begin orig 12')
      DO 560 I=1,N
         DO 560 J=1,N
  560 VEC(j,i)=CP(j,i)
C       call ftimer('End orig 12')
      MODE=2
  570 ROLD=XOLD*THETA(1)*57.29577951308232D0
      RNOW=XNOW*THETA(1)*57.29577951308232D0
      RMIN=XMIN*THETA(1)*57.29577951308232D0
      IF(DEBUG)WRITE(IW,590) XOLD,EOLD*23.060542301389D0,DEOLD,ROLD
     1,             XNOW,ENOW*23.060542301389D0,DENOW,RNOW
     2,             XMIN,EMIN*23.060542301389D0,DEMIN,RMIN
      EOLD=ENOW
      IF(NPNTS.LE.200) RETURN
      WRITE(IW,600)
      DO 580 K=1,NPNTS
  580 WRITE(IW,610) K,X(K),F(K),DF(K)
      WRITE(IW,620)
      RETURN
  590 FORMAT(
     1/14X,3H X ,10X,6H F(X) ,9X,7H DF/DX ,21H   ROTATION (DEGREES),
     2/10H      OLD ,F10.5,3F15.10,
     3/10H  CURRENT ,F10.5,3F15.10,
     4/10H PREDICTED,F10.5,3F15.10/)
  600 FORMAT(3H  K,10H     X(K) ,15H       F(K)    ,10H     DF(K))
  610 FORMAT(I3,F10.5,2F15.10)
  620 FORMAT(10X)
      END
```

Aquests 8 reemplaçaments d'algorismes d'àlgebra lineal per cridades a rutines de BLAS 3 han dut a un amillorament de la velocitat d'execució de la rutina *interp* d'un factor 2.4. Després d'aquestes modificacions, certs resultats finals han canviat a partir de la 6 xifra decimal. Parlarem d'açò més endavant a l'apartat **6.2**.

4.5 Rutina rsp

La rutina *rsp* en realitat és molt curta i el que fa bàsicament és cridar a rutines del EISPACK versió 3 (que data del 1983) per tal de trobar els valors i vectors propis d'una matriu simètrica real en forma empaquetada. Aquesta rutina és responsable de les cridades que es fan a les rutines *daxpy* i *ddot* del BLAS 1 que apareixen sempre entre les primeres 10 posicions dels profiles, i que, depenent de l'input, poden a arribar a consumir un alt percentatge de CPU.

Nosaltres hem volgut substituir les cridades a EISPACK per cridades a LAPACK, un altre conjunt de subrutines de càlcul matricial més modern, i que té en compte, per tant, les arquitectures RISC predominants avui en dia i que usa tots les rutines BLAS quan li és possible. A més, s'intenta sempre fer ús del nivell de BLAS més alt possible, per tal de millorar-ne l'eficiència, tot i prenent avantatge de la possible implementació eficient dels nivells alts del BLAS per a la plataforma de treball escollida.

Ací tenim el codi original de rsp on es criden les rutines de EISPACK :

```
      SUBROUTINE RSP(A,N,NVECT,ROOT,VECT)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INCLUDE 'SIZES'
      DIMENSION A(*), ROOT(N), VECT(N,N)
      DIMENSION RWORK(8*MAXALL), IWORK(MAXALL)
C     Cridada a rutines EISPACK
      CALL EVVRSP(-1,N,NVECT,N*N, N,A,RWORK,IWORK,ROOT,VECT,0,I)
      RETURN
      END
```

Ací la substitució per la cridada a la rutina addient de LAPACK :

```
      SUBROUTINE RSP(A,N,NVECT,ROOT,VECT)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      INCLUDE 'SIZES'
      DIMENSION A(*), ROOT(N), VECT(N,N)
      DIMENSION RWORK(8*MAXALL)
C     desempaquetem la matriu a diagonalitzar, i la guardem en vect,
C     per que no l'utilitzem de moment
      in=0
      do i=1,n
        do j=1,i
          in=in+1
          vect(j,i)=a(in)
        end do
      enddo
C     cridem la rutina de diagonalitzacio de lapack :
      call dsyev('V','U',n,vect,n,root,rwork,8*maxall,i)
      RETURN
      END
```

L'ús de les rutines de LAPACK no ha repercutit en una millora global de la velocitat de Mopac; ans bé, la velocitat d'execució ha baixat dràsticament segons inputs. La raó d'aquest comportament rau en què al Mopac s'estan usant unes rutines EISPACK modificades per E.T. Elbert del AMES-LABORATORY USODE en 1985 per tal de fer que el procediment de SCF convergesca el més ràpidament possible, a costa de perdre precissió en el càlcul de valors propis. Amb la versió de LAPACK el número

d'iteracions SCF abans d'arribar a la convergència en el cas de l'input 01.dat és de 29 mentre que amb el codi que usa el EISPACK modificat aquest número és de tan sols 10. Això es tradueix en què l'executable amb LAPACK i per a l'input 01.dat tarda quasibé tres vegades més que amb el codi original. Per a l'input 05.dat, però, el número d'iteracions SCF és exactament el mateix abans de la modificació que després, però els temps d'execució són pràcticament els mateixos.

Per tant, per tal d'optimitzar més aquest procés de trobada de valors propis hi hauria que saber quines condicions calen per tal de fer que el número d'iteracions SCF siga mínim, i aplicar-ho al cas de la rutina SYEV de LAPACK. Però com que nosaltres no disposem d'eixos coneiximents, aquesta modificació no entrarà en la versió final optimitzada de Mopac 93.

4.6 Rutina ss

Els *profilings* ens diuen que aquesta rutina consumeix molta CPU, i si a més, tenim en compte que des d'ací també es crida a les rutines *_xldipow* (rutina del sistema que eleva un número real a una potència entera) i *bfn* que són unes grans consumidores de CPU, ens adonarem que una petita optimització ací ens pot comportar grans beneficis en el rendiment global del Mopac.

Ací tenim el codi original de la rutina *ss* :

```
      DOUBLE PRECISION FUNCTION SS(NA,NB,LA1,LB1,M1,UA,UB,R1)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      LOGICAL FIRST
      DIMENSION FA(0:13),AFF(0:2,0:2,0:2),AF(0:19),BF(0:19),
     1BI(0:12,0:12)
      SAVE AFF, FA, BI, FIRST
      DATA FIRST /.TRUE./
      DATA AFF/27*0. D0/
      DATA FA/1.D0,1.D0,2.D0,6.D0,24.D0,120.D0,720.D0,5040.D0,40320.D0,
     1362880.D0,3628800.D0,39916800.D0,479001600.D0,6227020800.D0/
      M=M1-1
      LB=LB1-1
      LA=LA1-1
      R=R1/0.529177249D+00
      IF(FIRST) THEN
         FIRST=.FALSE.
C
C           INITIALISE SOME CONSTANTS
C
C                 BINOMIALS
C
         DO 10 I=0,12
            BI(I,0)=1.D0
            BI(I,I)=1.D0
   10    CONTINUE
         DO 20 I=0,11
            I1=I-1
            DO 20 J=0,I1
               BI(I+1,J+1)=BI(I,J+1)+BI(I,J)
   20    CONTINUE
         AFF(0,0,0)=1.D0
         AFF(1,0,0)=1.D0
         AFF(1,1,0)=1.D0
         AFF(2,0,0)=1.5D0
         AFF(2,1,0)=1.7320508076D0
         AFF(2,2,0)=1.2247448714D0
         AFF(2,0,2)=-0.5D0
      ENDIF
```

```
      P=(UA+UB)*R*0.5D0
      B=(UA-UB)*R*0.5D0
      QUO=1/P
      AF(0)=QUO*EXP(-P)
      DO 30 N=1,19
         AF(N)=N*QUO*AF(N-1)+AF(0)
   30 CONTINUE
      CALL BFN(B,BF)
      SUM=0.D0
      LAM1=LA-M
      LBM1=LB-M
C
C          START OF OVERLAP CALCULATION PROPER
C
      DO 50 I=0,LAM1,2
         IA=NA+I-LA
         IC=LA-I-M
         DO 50 J=0,LBM1,2
            IB=NB+J-LB
            ID=LB-J-M
            SUM1=0.D0
            IAB=IA+IB
C
C   In the Manual ka = K6
C                 kb = K5
C                 Pa = K1
C                 Pb = K2
C                 qa = K3
C                 qb = K4
C
            DO 40 K1=0,IA
               DO 40 K2=0,IB
                  DO 40 K3=0,IC
                     DO 40 K4=0,ID
                        DO 40 K5=0,M
                           IAF=IAB-K1-K2+K3+K4+2*K5
                           DO 40 K6=0,M
                              IBF=K1+K2+K3+K4+2*K6
                              SUM1=SUM1+BI(ID,K4)*BI(IC,K3)*
     1BI(IB,K2)*BI(IA,K1)*BI(M,K5)*BI(M,K6)*(1-2*MOD(M+K2+K4+K5+K6,2))
     2*AF(IAF)*BF(IBF)
   40       CONTINUE
            SUM=SUM+SUM1*AFF(LA,M,I)*AFF(LB,M,J)
   50 CONTINUE
      SS=SUM*R**(NA+NB+1)*UA**NA*UB**NB/(2.D0**(M+1))*
     1SQRT(UA*UB/(FA(NA+NA)*FA(NB+NB))*((LA+LA+1)*(LB+LB+1)))
      RETURN
      END
```

Com es pot observar, hi ha una sèrie de 8 bucles *aniuats* (*nested loops*), i una operació més o menys complicada en l'interior. S'ha intentat d'aplicar canvis en l'ordre dels bucles, variar l'ordre de les operacions en el bucle més intern, etc ..., per tal de veure si les prestacions augmentaven, però sense cap èxit : les versions modificades sempre eren més lentes, o com a màxim igual de ràpides que el codi original.

Degut a la impossibilitat d'optimitzar més el codi sèrie, es va optar per intentar d'aplicar tècniques de processament paral·lel per tal d'accelerar la velocitat de la rutina. L'entorn de programació escollit per fer tal de fer aquesta paral·lelització ha estat el PVM (Parallel Virtual Machine) que permet la comunicació entre processos que correguen en plataformes diferents. Es tracta doncs d'un entorn de programació amb paradigma de memòria distribuïda, efectuant-se la comunicació de dades entre els diferents procesos a través d'intercanvi de missatges explícits.

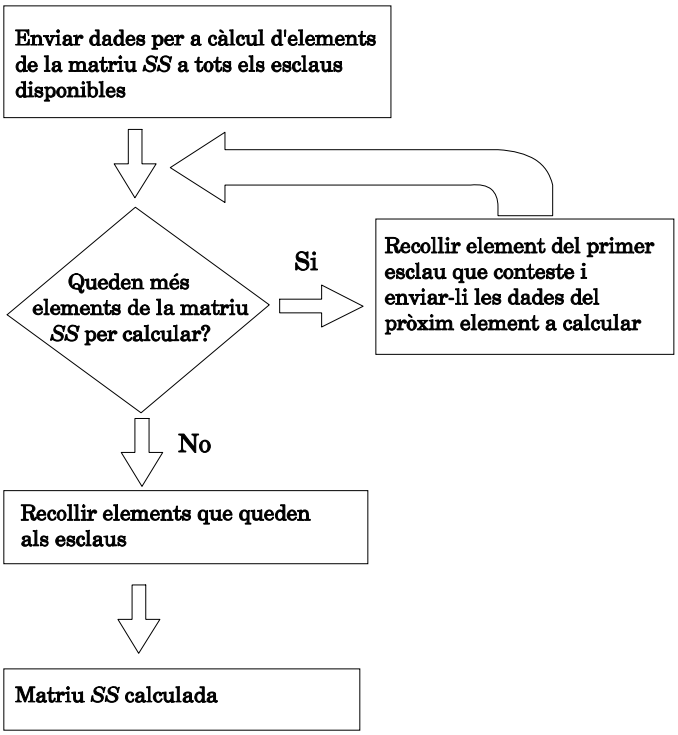

Figura 4: Implementació paral·lela de ss, amb distribució dinàmica de càrrega

L'algorisme dissenyat per a la implementació paral·lela es pot veure a a figura 4. Com es veu, s'ha optat per una distribució dinàmica de càrrega, que sempre sol donar millors resultats que la estàtica, ja que l'algorisme de càlcul ens ho ha permés.

Ací està el codi que corre en la part del *master* :

```
      program mrsp
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      include 'SIZES'
      include '/usr/local/pvm/include/fpvm3.h'
      parameter ( max_slaves = 8)
      parameter (nel=3)
      integer mytid,tids(1:max_slaves)
      integer info, msgtype
      integer first_row(max_slaves)
      integer nb_slaves
      logical recollir
      character*18 node


      dimension s(nel,nel,nel),ul1(nel),ul2(nel)

c
c --  Enroll this program in PVM
```

```
c

      call pvmfmytid(mytid)
      write(*,*)' master id',mytid
c
c --  Initialize number of slaves
c
      write(*,*)' How many slaves will you used (1-8)'
      read(*,*)nb_slaves
      node = 'PVMsss'
      call pvmfspawn(node,PVMDEFAULT,'RS6K',nb_slaves,tids,numt)
      if (numt.lt.nb_slaves) then
        print*, ' trouble spawning',node
        print*, ' Check tids for error code '
        call shutdown (numt,tids)
        stop
      endif
c
c --  Print out task ID's of spawned tasks and check for problems
c
      do 100 i=1,nb_slaves
        print*,'tid ',i,tids(i)
 100  continue
c
c --  Begin user program
c
      nocc=2
C       n=nel
C     inicialitzem la matriu ss
      call ftimer('Comencem')
      do i=1,nel
        ul1(i)=sqrt(dble(i))*0.01d0
        ul2(i)=i*0.002d0
        do j=1,nel
          do k=1,nel
            s(k,j,i)=0.d0
          end do
        end do
      end do
      call ftimer('Matriu plena')
      write(6,'(3(F8.2))') (s(1,1,i2),i2=1,nel)
      r=2.d0
      total=0.d0
      iescl=1
      in=1
      itot=nel*nel*nel
      msgtype=1
      recollir=.FALSE.
      call ftimer('Begin bucle omplir :')
      do i2=1,10
      do i=1,nel
        do j=1,nel
          do k=1,nel
C   enviar info a cadascun dels esclaus. 4 enters i 3 double's
              call pvmfinitsend(PVMDATARAW, info)
              call pvmfpack( INTEGER4, iescl, 1, 1, info)
              call pvmfpack( INTEGER4, i, 1, 1, info)
              call pvmfpack( INTEGER4, i, 1, 1, info)
              call pvmfpack( INTEGER4, i, 1, 1, info)
              call pvmfpack( INTEGER4, j, 1, 1, info)
              call pvmfpack( INTEGER4, k, 1, 1, info)
              call pvmfpack( REAL8, ul1(i), 1, 1, info)
              call pvmfpack( REAL8, ul2(j), 1, 1, info)
              call pvmfpack( REAL8, r, 1, 1, info)
              call pvmfsend( tids(iescl), msgtype, info )
D              print *, 'enviada info a esclau :',tids(iescl)
```

```
              in=in+1
C     Hem enviat la info a tots els esclaus?
              if ((iescl.lt.nb_slaves).and.(.not.recollir)) then
C               Si no, incrementar el numero d'esclau, i tornar a enviar
                iescl=iescl+1
              else
C     Ja tenim tots els esclaus alimentats. Ens posem a recollir info.
                recollir=.TRUE.
              endif
              if (recollir) then
C             Hem de comencar a recollir i preparar-se per a tornar a
C               enviar info a l'esclau que envia, si s'escau
D                print *,'Abans de rebre info'
   30           call pvmfrecv(-1, msgtype, info)
                call pvmfunpack(INTEGER4, iescl, 1, 1, info)
                call pvmfunpack(INTEGER4, if, 1, 1, info)
                call pvmfunpack(INTEGER4, jf, 1, 1, info)
                call pvmfunpack(INTEGER4, kf, 1, 1, info)
                call pvmfunpack(REAL8, s(if,jf,kf), 1, 1, info)
D                print *,'Despres de rebre info de esclau ',
D     &            iescl,if,jf,kf
                total=total+s(if,jf,kf)
C                 Recollim els esclaus que ens queden al final
                if (in.gt.itot.and.in.lt.itot+nb_slaves) then
                  in=in+1
                  goto 30
                end if
              endif
C             s(k,j,i)=ss(i,j,k,j,k,ul1(i),ul2(j),r)
            end do
          end do
        end do
        end do
        call ftimer('End bucle omplir :')
        do i=1,3
          do j=1,3
            write(6,150) (s(i2,j,i),i2=1,3)
          end do
        end do
        print *, 'Suma total : ',total
C       Matem els esclaus que hem creat ...
        call shutdown(nb_slaves,tids)
 150    format(3(F20.15))
        end

        subroutine shutdown( nproc, tids )
        integer nproc, tids(*)
c
c       Kill all tasks I spawned and then myself
c
        do 10 i=1, nproc
          call pvmfkill( tids(i), info )
 10     continue
        call pvmfexit( info )
        return
        end
```

I ací la part de l'*slave* :

```
C       DOUBLE PRECISION FUNCTION SS(NA,NB,LA1,LB1,M1,UA,UB,R1)
        program PVMss
        include '/usr/local/pvm/include/fpvm3.h'
        IMPLICIT DOUBLE PRECISION (A-H,O-Z)
        integer mytid, info, msgtype, mtid
        LOGICAL FIRST
        DIMENSION FA(0:13),AFF(0:2,0:2,0:2),AF(0:19),BF(0:19),
```

```
     1BI(0:12,0:12)
      SAVE AFF, FA, BI, FIRST
      DATA FIRST /.TRUE./
      DATA AFF/27*0. D0/
      DATA FA/1.D0,1.D0,2.D0,6.D0,24.D0,120.D0,720.D0,5040.D0,40320.D0,
     1362880.D0,3628800.D0,39916800.D0,479001600.D0,6227020800.D0/
c
c --- Enroll this program
c
      call pvmfmytid(mytid)
D      write(*,*)' from ',mytid
c
c --- Get the master's task id
c
      call pvmfparent(mtid)
c
c --- Begin user program
C       Rebem parametres del mestre
  110 msgtype=1
      call pvmfrecv(mtid, msgtype, info)
      call pvmfunpack(INTEGER4, iescl, 1, 1, info)
      call pvmfunpack(INTEGER4, na, 1, 1, info)
      call pvmfunpack(INTEGER4, nb, 1, 1, info)
      call pvmfunpack(INTEGER4, la1, 1, 1, info)
      call pvmfunpack(INTEGER4, lb1, 1, 1, info)
      call pvmfunpack(INTEGER4, m1, 1, 1, info)
      call pvmfunpack(REAL8, ua, 1, 1, info)
      call pvmfunpack(REAL8, ub, 1, 1, info)
      call pvmfunpack(REAL8, r1, 1, 1, info)
D      write(*,*)' from ',mytid,' index ',iescl,nb,la1,lb1,m1

      M=M1-1
      LB=LB1-1
      LA=LA1-1
      R=R1/0.529177249D+00
      IF(FIRST) THEN
         FIRST=.FALSE.
C
C           INITIALISE SOME CONSTANTS
C
C                  BINOMIALS
C
         DO 10 I=0,12
            BI(I,0)=1.D0
            BI(I,I)=1.D0
   10    CONTINUE
         DO 20 I=0,11
            I1=I-1
            DO 20 J=0,I1
               BI(I+1,J+1)=BI(I,J+1)+BI(I,J)
   20    CONTINUE
         AFF(0,0,0)=1.D0
         AFF(1,0,0)=1.D0
         AFF(1,1,0)=1.D0
         AFF(2,0,0)=1.5D0
         AFF(2,1,0)=1.7320508076D0
         AFF(2,2,0)=1.2247448714D0
         AFF(2,0,2)=-0.5D0
      ENDIF
      P=(UA+UB)*R*0.5D0
      B=(UA-UB)*R*0.5D0
      QUO=1/P
      AF(0)=QUO*EXP(-P)
      DO 30 N=1,19
         AF(N)=N*QUO*AF(N-1)+AF(0)
   30 CONTINUE
```

```
      CALL BFN(B,BF)
      SUM=0.D0
      LAM1=LA-M
      LBM1=LB-M
C
C          START OF OVERLAP CALCULATION PROPER
C
      DO 50 I=0,LAM1,2
         IA=NA+I-LA
         IC=LA-I-M
         DO 50 J=0,LBM1,2
            IB=NB+J-LB
            ID=LB-J-M
            SUM1=0.D0
            IAB=IA+IB
C
C   In the Manual ka = K6
C                 kb = K5
C                 Pa = K1
C                 Pb = K2
C                 qa = K3
C                 qb = K4
C
            DO 40 K1=0,IA
               DO 40 K2=0,IB
                  DO 40 K3=0,IC
                     DO 40 K4=0,ID
                        DO 40 K5=0,M
                           IAF=IAB-K1-K2+K3+K4+2*K5
                           DO 40 K6=0,M
                              IBF=K1+K2+K3+K4+2*K6
                              SUM1=SUM1+BI(ID,K4)*BI(IC,K3)*
     1BI(IB,K2)*BI(IA,K1)*BI(M,K5)*BI(M,K6)*(1-2*MOD(M+K2+K4+K5+K6,2))
     2*AF(IAF)*BF(IBF)
   40       CONTINUE
            SUM=SUM+SUM1*AFF(LA,M,I)*AFF(LB,M,J)
   50 CONTINUE
      SS=SUM*R**(NA+NB+1)*UA**NA*UB**NB/(2.D0**(M+1))*
     &  SQRT(UA*UB/(FA(NA+NA)*FA(NB+NB))*((LA+LA+1)*(LB+LB+1)))

C       tornem la informacio al master :
D      write(*,*) ' from ',mytid, 'Abans d''enviar coses al master'
       call pvmfinitsend(PVMDATAINPLACE, info)
       call pvmfpack( INTEGER4, iescl, 1, 1, info)
       call pvmfpack( INTEGER4, la1, 1, 1, info)
       call pvmfpack( INTEGER4, lb1, 1, 1, info)
       call pvmfpack( INTEGER4, m1, 1, 1, info)
       call pvmfpack( REAL8, ss, 1, 1, info)
       call pvmfsend( mtid, msgtype, info )
D      write(*,*) ' from ',mytid, 'Despres d''enviar coses al master'
       goto 110
       return
       END
```

Aquesta implementació paral·lela ha estat provada amb un cluster de workstations format per una IBM RS/6000-550, una IBM RS/6000-950 i una IBM RS/6000-330 connectades a través d'un bus ethernet.

Les prestacions obtingudes han estat, però, molt dolentes i per sota de les de la versió sèrie. El motiu és el següent : la rutina *ss* no consumeix molt per ella mateixa, i ix tan amunt en els profiles degut a que és cridada dezenes de milers de vegades durant un càlcul típic. Per tant, en la versió paral·lalela els missatges que s'han d'intercanviar entre els processos *master* i *slave* pugen a centenars de milers, i tot i que siguen missatges curts, a soles l'*overhead* que representa la cridada a les rutines PVM que

s'encarreguen de les comunicacions fa que aquesta aproximació no siga eficient per tal d'accelerar els càlculs.

En realitat, açò últim és conseqüència de la "regla d'or" que diu que els sistemes de pas de missatge explícit en sistemes de memòria distribuïda són aptes a sols per a efectuar paral·lalelitzacions de grau gros (o siga, paralel·lització de rutines que es criden poques vegades i consumeixen molta CPU), i no per a les de grau fi, com és el cas de la rutina *ss*. Seria interessant, però, comprovar quina és l'eficiència d'aquesta implentació paral·lela sobre màquines de memòria compartida. Nosaltres no ho fem per falta de disponibilitat de màquines d'aquest caire. De moment, deixarem clar que la rutina *ss* serà una de les que més CPU consumirà en la versió final optimitzada del Mopac 93, i s'hauria de fer un èmfasi especial en reduir el temps d'execució d'aquesta rutina en ulteriors intents d'optimització (possiblement, usant tècniques de càlcul paral·lel).

4.7 Versió optimitzada definitiva

Fins ara hem descrit les optimitzacions que s'han intentat fer en les rutines del Mopac que més CPU consumeixen. Ha arribat el moment de juntar les optimitzacions que han donat un bon resultat i elaborar-ne una nova versió del Mopac amb aquestes, per tal de veure quin n'és el rendiment global.

Hem descrit abans com les modificacions efectuades sobre les rutines **diag, mamult, densit i interp** han estat les més efectives des del punt de vista del guany en velocitat. Per tant, hem el·laborat una versió de Mopac optimitzada amb les modificacions efectuades sobre aquestes quatre rutines.

El nou codi s'ha compilat sobre una IBM RS/6000-590 i enllaçant (*linking*) amb dues versions de les BLAS. La primera amb les BLAS que venen amb el sistema operatiu per defecte, que ja estan optimitzades sobre el codi original de Dongarra et al. En la segona enllacem amb les BLAS existents a la llibreria matemàtica més general de IBM, les ESSL, i que estan prou més optimitzades que les anteriors i que donen lloc a unes prestacions molt properes al rendiment punta dels processadors POWER i POWER2 de IBM.

Ací van les dades que comparen a les tres versions de Mopac : l'original, la enllaçada amb les llibreries BLAS "standard" que venen amb el sistema operatiu AIX, i la enllaçada amb les BLAS presents a les ESSL. Les proves s'han efectuat a una IBM RS/6000 mod 590 amb el processador POWER 2 i amb 256 MB de RAM, amb la versió del compilador FORTRAN XLF 3.1. Els resultats es poden veure a la taula 3, on en la primera fila es representa el temps d'execució per a cadascun dels inputs en segons, i en les dues files posteriors, es donen els speed-ups.

Com es veu, per a la versió de Mopac optimitzada i enllaçada amb les BLAS del sistema operatiu s'aconsegueix un speed-up d'entre un factor 1.34 a 1.84 i amb les BLAS de les ESSL l'speed-up oscil·la entre 1.43 i 2.26 sobre el Mopac original.

També incloem a mode d'il·lustració, speed-ups obtinguts amb una IBM RS/6000 model 390 (veure taula 4), que té un processador POWER 2 igual que el model anterior (i també la mateixa velocitat de rellotge, i.e. 66 MHz) però amb un bus de comunicació memòria-processador bastant menys ràpid. A sols varem poder fer proves amb les BLAS que venien per defecte amb el sistema operatiu, ja que les ESSL no hi eren presents al sistema.

Versió del programa	Fitxer d'input				
	01	02	03	04	05
Codi original (segons)	388	703	1457	997	255
Enllaçat amb BLAS (speed-up)	1.84	1.64	1.65[1]	1.41	1.34
Enllaçat a ESSL (speed-up)	1.97	1.84	2.26[1]	1.53	1.43

Taula 3: Comparació de prestacions entre la versió original de Mopac i les versions optimitzades, per a un IBM RS/6000-590

Versió del programa	Fitxer d'input				
	01	02	03	04	05
Codi original (segons)	541	1003	2024[2]	1350	324
Enllaçat amb BLAS (speed-up)	2.11	1.87	1.83[2]	1.57	1.47

Taula 4: Comparació de prestacions entre la versió original de Mopac i les versions optimitzades, per a un IBM RS/6000-390

Observem una cosa completament predictible : els speed-ups són molt més el·levats en màquines amb un ample de banda de comunicació processador-memòria menor, donant lloc a un treball més vistós per part de les BLAS. Ara els speed-up per a la versió de Mopac optimitzada i enllaçada amb la versió de BLAS que ve amb el sistema operatiu puja a un factor entre 1.47 i 2.11, i seria probablement prou superior en una versió enllaçada amb les BLAS de les ESSL.

5 Efectes secundaris : major ús de memòria i conseqüències sobre precissió dels resultats finals

Normalment, sempre que s'efectúa algun tipus d'optimització sobre codi existent, hi apareixen efectes colaterals deguts a les modificacions en els algorismes inicials. En aquest apartat volem discutir els efectes més importants detectats derivats de l'aplicació de les optimitzacions.

5.1 Major ús de memòria

El principi fonamental de la majoria de les optimitzacions esmentades més amunt ha estat desempaquetar les matrius comprimides per tal de poder cridar després a les rutines de BLAS optimitzades. Naturalment, açò comporta el que la versió de Mopac amb aquestes optimitzacions consumesca més memòria que l'original. Tot i això, si ens fixem en totes i cadascuna de les optimitzacions, les matrius desempaquetades

[1] Aquestes mesures hi ha que prendre-les amb precaució, ja que el número de cicles de SCF no és el mateix que l'original. Recalquem però, que es continuen obtenint resultats per al calor de formació que concorden molt aproximadament amb els originals.

[2] Aquest càlcul ha durat 39 cicles de SCF per al codi original i 38 per a l'optimitzat.

s'usen temporalment, sense ser necessari conservar-ne el seu contingut entre cridades a subrutines consecutives. Això ens permet d'usar un espai comú de memòria per a totes les rutines que usen matrius descomprimides de manera temporal, i reduïr d'aquesta manera els requeriments totals de memòria.

Amb programes FORTRAN això s'aconsegueix mijantçant l'ús d'estructures **COMMON** on es reserva la màxima quantitat de memòria utilitzable per cadascuna de les rutines, de manera que tots hi puguen accedir a la mateixa memòria anomenem-la auxiliar. Les matrius d'ajuda s'agafen del màxim tamany possible per a la configuració del Mopac escollida en temps de compilació. A la taula 5 podem veure quin és l'increment en les necessitats de memòria per part del Mopac per a diferents configuracions.

(MAXHEV,MAXLIT)	Requeriment addicional de memòria (MB)
(45, 45)	2.38
(100, 100)	11.76
(150, 150)	26.46

Taula 5: Requeriment addicional de memòria per a la versió de Mopac optimitzada segons configuracions escollides en temps de compilació

MAXHEV i **MAXLIT** és el nombre d'àtoms pesats (majors que l'hidrògen) i de lleugers respectivament, escollits en la configuració de Mopac, abans de la compilació.

Com es veu, per a configuracions de molècules grans (un màxim de 150 àtoms pesats i 150 d'hidrògen), la necessitat de memòria addicional comença a fer-se apreciable (26.5 MB), i hi hauria que tenir ben en compte la quantitat de memòria disponible en el nostre sistema l'hora d'optar per usar la versió optimitzada del Mopac.

Tot i això, com ja hem dit abans, a les plataformes de càlcul científic actuals ja es normal trobar-nos que superen els 100 MB de memòria principal. A més, en els moderns sistemes operatius UNIX, comença a ser normal la capacitat de carregar pàgines de memòria en RAM a soles si el programa ho demana. Çò és, si per exemple estem usant una versió de Mopac compilada per acceptar configuracions (150,150), i realment volem calcular una configuració molecular, diguem-ne (45,45), segurament l'executable no necessitarà 26 MB addcionals, sinò que probablement amb 4 MB (diem 4 MB i no 2.38 MB per tal de contemplar els efectes d'usar pàgines de memòria completes) més li serà suficient a l'executable. Aquesta característica fa, doncs, que no siga especialement crític el tenir diferents versions compilades del Mopac optimitzat per a diferents configuracions, i que tenint una única versió (o dues com a màxim) del Mopac preparada per a molècules grans, el sistema operatiu s'encarregarà d'assignar dinàmicament la quantitat de memòria necessària per a qualsevol configuració.

5.2 Conseqüències sobre precissió dels resultats finals

El fet de fer les anteriors optimitzacions du amb elles un canvi en l'ordre d'execució de les operacions aritmètiques, algebraiques i, de vegades, algorísmiques implicades en el codi optimitzat, i això porta, en general, a canvis en la precissió final del resultat que depenen de la precissió en coma flotant utilitzada. Per exemple, considerem les operacions en coma flotant següents :

$$A = B \cdot C \cdot D$$
$$\bar{A} = C \cdot D \cdot B$$

Si treballarem amb algorismes de precissió infinita, aleshores podriem afirmar que A seria exactament igual a $\bar{A}$ ($A \equiv \bar{A}$). En la pràctica, però, ens hem de conformar amb una precissió limitada per raons de espai d'emmagatzimament i de velocitat d'execució. Usant precissió limitada, tindrem que, en general, no es compleix que $A \equiv \bar{A}$. Així que, una optimització que comporte un canvi en l'ordre de les operacions (i açò és molt comú), normalment durà a canvis significatius en els resultats finals. De fet, inclús amb versions idèntiques de codi, és relativament normal d'obtenir resultats diferents (normalment, les últimes xifres decimals) al córrer aquest codi en diferents plataformes RISC.

En les plataformes actuals sol haver tres classes de precissió aplicables a variables en coma flotant: simple (32 bits, mantissa de 7 xifres), doble (64 bits, mantissa de 15 xifres) i quadruple (128 bits, mantissa de 30 xifres). Les precisions simple i doble solen ser les més usades degut a que les operacions suma i multiplicació per a aquestes s'efectuen per hardware, normalment en un sol cicle de rellotge. Les operacions amb precisió quadruple s'efectuen per software i tarden dezenes o centenars de cicles de rellotge en efectuar-se. Mopac treballa normalment en aritmètica de doble precissió.

Els resultats obtinguts amb la versió optimitzada del Mopac difereixen, depenent dels inputs, a partir de la 6 o 7 xifra decimal dels obtinguts amb el Mopac sense optimitzar. Els punts on es donen les divergències més significatives (parlem de diferències en la 6 o 7 xifra decimal) s'han detectat clarament, i resulten estar a l'optimització de la rutina *densit* i a la corresponent a la modificació 8 de la rutina *interp*. Això era esperable, ja que ahí precisament és on més divergències hi existeixen respecte de l'algorisme original.

Aquestes divergències s'observen, naturalment, per a una mateixa plataforma (en aquest cas una IBM RS/6000 mod 550). Fem notar un fet curiós : els resultats del codi original difereixen a partir de la 7 xifra decimal entre una plataforma IBM RS/6000 550 i una DEC ALPHA 3000/300. Aplicant totes les optimitzacions per a la plataforma IBM, els resultats coincidien fins a la 8 xifra decimal amb els obtinguts amb la plataforma DEC ALPHA aplicant-hi a sols l'optimització de la rutina *densit*. Després d'aplicar totes les optimitzacions del programa a la plataforma DEC, els resultats tornaven a diferir a partir de la 7 xifra.

La conclusió que s'extrau de tot açò és que, encara que es treballe amb precissió de 15 xifres decimals, la propagació d'errors degut a la precissió finita fa que la precissió final dels resultats, depenga molt tant dels algorismes elegits com de la plataforma escollida per tal d'efectuar el càlcul. El que s'ha de fer aleshores és determinar on són els punts crítics del codi causants de les divergències més fortes i intentar fer aquestes parts més resistents a les pèrdues en la precissió, ja que moltes vegades aquesta pèrdua pot ser reduida o fins i tot eliminada, tenint cura de l'ordre de les operacions efectuades i dels algorismes usats.

També, hem de veure si les divergències son suficientment grans des del punt de vista de la precissió requerida per les nostres necessitats, per tal de determinar si ens és necessari dedicar temps i esforç a detectar problemes de pèrdua de precissió que solen ser sempre massa delicats com per a que siga una tasca fàcil el solucionar-los.

Versió de Mopac	Valor obtingut d'energia electrònica
Original	-55627.239352 eV
Aplicant optimització en rutina *densit*	-55627.238871 eV
Aplicant optimització en rutina *densit* i *interp*	-55627.323429 eV

Taula 6: Diferències en precissió de resultats finals com a conseqüència de l'optimització del Mopac

Prenem com a exemple les divergències en quant a l'energia electrònica obtinguda en les versions optimitzades respecte al valor resultant del Mopac 93 original. A la taula 6 tenim llistades aquestes diferències segons la versió del Mopac usada. Podem observar que les diferències què es troben entre les diferents versions per a l'energia electrònica són de l'ordre de magnitud de **kT** (com a màxim, 5 kT), on **k** és la constant de Boltzmann i **T** la temperatura en Kelvin (això equival a 0.026 eV), el que vol dir que estem en precissions de l'ordre de les fluctuacions en energia degudes a l'agitació tèrmica. Naturalment, no té gaire sentit físic el càlcul d'energies electròniques per sota d'eixa precissió. Aleshores podem contentar-nos amb les diferències esmentades en la 6 o 7 xifra, encara que, des del punt de vista purament numèric estiga clar que estem tenint un problema de pèrdua de precissió més o menys greu.

En qualsevol cas, si fóra necessària l'obtenció de totes les xifres decimals de la versió de Mopac original, això seria factible simplement deshabilitant les optimitzaciones responsables de les divergències i localitzades, com ja hem dit adés, a les rutines *densit* i al bucle número 8 d'*interp*.

Finalment, posem de relleu la convinença d'expressar les operacions matricials en termes de cridades a BLAS per tal d'homogeneitzar les possibles diferències de màquina a màquina, i fer que la propagació d'errors siga la mínima possible. En efecte, recordem que les llibreries BLAS han estat dissenyades per experts que han tingut molt en compte el problema de la propagació d'errors, a més del tema de les prestacions en el càlcul. Aquest factor queda palés encara més quan es fa ús de les llibreries BLAS optimitzades pel propi fabricant per a la seua plataforma en particular, ja que d'aquesta manera ell pot traure profit del seu coneiximent de l'arquitectura per tal d'assegurar-nos, a banda d'unes prestacions quan a velocitat immillorables, uns resultats el més paregut possible als exactes.

6 Conclusions

De tot el que hem exposat podem obtenir les següents conclusions :

- L'ús eficient de la memòria és fonamental per a obtindre el màxim rendiment de les plataformes actuals de càlcul basades en arquitectures RISC.
- Les rutines BLAS constitueixen una eina molt apropiada per tal de crear codis eficients independents de màquina, alhora que donen bones garanties sobre el bon comportament dels algorismes quan a propagació d'errors deguts als problemes associats als càlculs de coma flotant de precissió limitada

També queden obertes certes vies de treball per tal de continuar la tasca d'optimització :

- Intentar acurtar el temps de càlcul del Mopac tot i disminuint el número total d'iteracions SCF per tal d'arribar a la convergència. Per tal de fer açò, però, hi hauria que adquirir una cert grau de coneiximent i experiència en teoria de SCF.
- Intentar aplicar tècniques de paral·lelisme de grau gros al Mopac, per tal de poder crear-ne una versió apta per ser executada en màquines de memòria distribuïda, asequibles avui en dia per qualsevol centre de càlcul o departament mitjà. Novament, açò també implicaria certa experiència en la mecànica de càlcul del programa, per tal de seleccionar quins serien els bons llocs per tal d'aplicar el paral·lelisme de grau gros.

Apèndix A. Inputs usats en les proves de velocitat

A sota es donen els inputs que s'han fet servir durant les proves de velocitat del Mopac.

Input 01.dat :

```
PREC MNDO  T=400000
(ZnO)22 modelo II.1CO(28)

 O     .000000  0     .000000  0     .000000  0    0    0    0       -.6628
Zn    1.973561  0     .000000  0     .000000  0    1    0    0        .5056
 O    1.973561  0  110.841931  0     .000000  0    2    1    0       -.5120
Zn    1.973576  0  110.842228  0  180.000000  0    3    2    1        .5138
 O    1.973569  0  110.842376  0  180.000000  0    4    3    2       -.5083
Zn    1.973569  0  110.842228  0  180.000001  0    5    4    3        .4640
 O    1.973569  0  110.842228  0  180.000001  0    6    5    4       -.6291
Zn    1.973000  0  110.866497  0   56.427279  0    1    2    3        .5014
 O    1.973569  0  110.865954  0  -56.440985  0    8    1    2       -.5174
Zn    1.973000  0  110.866497  0  -56.427279  0    3    2    1        .4900
 O    1.973569  0  110.866415  0  180.000132  0   10    3    2       -.5310
Zn    1.973000  0  110.866184  0  -56.427099  0    5    4    3        .4530
 O    1.973569  0  110.866184  0  180.000000  0   12    5    4       -.5330
Zn    1.973000  0  110.866184  0  -56.427099  0    7    6    5        .4256
 O    1.973569  0  110.866184  0  180.000000  0   14    7    6       -.5823
Zn    1.973000  0  110.866184  0  180.000000  0   15   14    7        .4535
 O    1.973569  0  110.866184  0   56.440853  0   16   15   14       -.6376
Zn    1.973000  0  110.866184  0  180.000000  0   13   12    5        .4231
 O    1.973569  0  110.866184  0   56.440853  0   18   13   12       -.2160
Zn    1.973000  0  110.866184  0  180.000132  0   11   10    3        .4306
 O    1.973576  0  110.866102  0   56.440721  0   20   11   10       -.6380
Zn    1.973000  0  110.866184  0  179.999868  0    9    8    1        .4818
 O    1.992000  0  108.049441  0  -61.779573  0   22    9    8       -.4775
Zn    1.973561  0  108.044132  0 -120.010680  0   23   22    9        .6407
 O    1.973576  0  110.842228  0   61.796636  0   24   23   22       -.4261
Zn    1.973569  0  110.842376  0  180.000000  0   25   24   23        .0405
 O    1.973569  0  110.842228  0  180.000000  0   26   25   24       -.4449
Zn    1.973569  0  110.842228  0  180.000000  0   27   26   25        .5379
 O    1.973569  0  110.842228  0  180.000000  0   28   27   26       -.5016
Zn    1.973000  0  110.866184  0   56.427099  0   29   28   27        .5296
 O    1.973569  0  110.866184  0  -56.440853  0   30   29   28       -.3852
Zn    1.973000  0  110.866184  0   56.427099  0   27   26   25        .5478
 O    1.973569  0  110.866184  0  -56.440853  0   32   27   26       -.4318
Zn    1.973000  0  110.866102  0   56.427052  0   25   24   23        .5418
 O    1.973569  0  110.866184  0  -56.440721  0   34   25   24       -.4479
Zn    1.973000  0  108.049441  0     .000000  0   23   22    9        .5278
 O    1.973569  0  110.866184  0   61.779573  0   36   23   22       -.4972
Zn    1.973000  0  110.865954  0  180.000132  0   37   36   23        .6189
 O    1.973561  0  110.866497  0   56.440985  0   38   37   36       -.4980
Zn    1.973000  0  110.866415  0  179.999868  0   35   34   25        .5154
 O    1.973576  0  110.865872  0   56.440721  0   40   35   34       -.5260
Zn    1.973000  0  110.866184  0  180.000000  0   33   32   27        .5099
 O    1.973569  0  110.866184  0   56.440853  0   42   33   32       -.4694
Zn    1.973000  0  110.866184  0  180.000000  0   31   30   29        .6041
 C    1.992019  1  125.152886  1   37.901998  1   28   27   26        .3469
 O    1.155629  1  178.690770  1   48.970512  1   45   28   27       -.0308
 0     .000000  0     .000000  0     .000000  0    0    0    0
```

Input 02.dat :

```
MNDO SCFCRT=0.00000000001 T=200000
(ZnO)22 modelo II.1H2O(44)

 O     .000000  0     .000000  0     .000000  0    0    0    0       -.6637
Zn    1.973561  0     .000000  0     .000000  0    1    0    0        .5050
 O    1.973561  0  110.841931  0     .000000  0    2    1    0       -.5133
Zn    1.973576  0  110.842228  0  180.000000  0    3    2    1        .5120
 O    1.973569  0  110.842376  0  180.000000  0    4    3    2       -.5118
Zn    1.973569  0  110.842228  0  180.000001  0    5    4    3        .4551
 O    1.973569  0  110.842228  0  180.000001  0    6    5    4       -.6676
Zn    1.973000  0  110.866497  0   56.427279  0    1    2    3        .5012
 O    1.973569  0  110.865954  0  -56.440985  0    8    1    2       -.5173
Zn    1.973000  0  110.866497  0  -56.427279  0    3    2    1        .4904
 O    1.973569  0  110.866415  0  180.000132  0   10    3    2       -.5314
Zn    1.973000  0  110.866184  0  -56.427099  0    5    4    3        .4562
 O    1.973569  0  110.866184  0  180.000000  0   12    5    4       -.5312
Zn    1.973000  0  110.866184  0  -56.427099  0    7    6    5        .4294
 O    1.973569  0  110.866184  0  180.000000  0   14    7    6       -.5229
Zn    1.973000  0  110.866184  0  180.000000  0   15   14    7        .4534
 O    1.973569  0  110.866184  0   56.440853  0   16   15   14       -.6299
Zn    1.973000  0  110.866184  0  180.000000  0   13   12    5        .4213
 O    1.973569  0  110.866184  0   56.440853  0   18   13   12       -.2137
Zn    1.973000  0  110.866184  0  180.000132  0   11   10    3        .4295
 O    1.973576  0  110.866102  0   56.440721  0   20   11   10       -.6351
Zn    1.973000  0  110.866184  0  179.999868  0    9    8    1        .4824
 O    1.992000  0  108.049441  0  -61.779573  0   22    9    8       -.4768
Zn    1.973561  0  108.044132  0 -120.010680  0   23   22    9        .6416
 O    1.973576  0  110.842228  0   61.796636  0   24   23   22       -.4263
Zn    1.973569  0  110.842376  0  180.000000  0   25   24   23        .0725
 O    1.973569  0  110.842228  0  180.000000  0   26   25   24       -.4230
Zn    1.973569  0  110.842228  0  180.000000  0   27   26   25        .6269
 O    1.973569  0  110.842228  0  180.000000  0   28   27   26       -.4400
Zn    1.973000  0  110.866184  0   56.427099  0   29   28   27        .4814
 O    1.973569  0  110.866184  0  -56.440853  0   30   29   28       -.4015
Zn    1.973000  0  110.866184  0   56.427099  0   27   26   25        .5411
 O    1.973569  0  110.866184  0  -56.440853  0   32   27   26       -.4300
Zn    1.973000  0  110.866102  0   56.427052  0   25   24   23        .5412
 O    1.973569  0  110.866184  0  -56.440721  0   34   25   24       -.4478
Zn    1.973000  0  108.049441  0     .000000  0   23   22    9        .5276
 O    1.973569  0  110.866184  0   61.779573  0   36   23   22       -.4977
Zn    1.973000  0  110.865954  0  180.000132  0   37   36   23        .6177
 O    1.973561  0  110.866497  0   56.440985  0   38   37   36       -.4986
Zn    1.973000  0  110.866415  0  179.999868  0   35   34   25        .5132
 O    1.973576  0  110.865872  0   56.440721  0   40   35   34       -.5299
Zn    1.973000  0  110.866184  0  180.000000  0   33   32   27        .5026
 O    1.973569  0  110.866184  0   56.440853  0   42   33   32       -.5166
Zn    1.973000  0  110.866184  0  180.000000  0   31   30   29        .5446
 O    2.000000  1  126.075279  1   39.023495  1   44   31   30       -.2351
 H     .962163  1  118.573450  1   31.054707  1   45   44   31        .2647
 H     .949206  1  118.918944  1  166.501520  1   45   44   31        .2502
 0     .000000  0     .000000  0     .000000  0    0    0    0
```

Input 03.dat :

```
SYMMETRY AM1 PREC  T=999999
(ZnO)22 modelo II
otimizacao das distancias e angulos com simetria -1a.
 O     .000000  0     .000000  0     .000000  0    0    0    0       -.7099
Zn    1.973561  1     .000000  0     .000000  0    1    0    0        .6193
 O    1.973561  1  110.841931  1     .000000  0    2    1    0       -.6088
Zn    1.973576  1  110.842228  1  180.000000  0    3    2    1        .6290
 O    1.973569  1  110.842376  1  180.000000  0    4    3    2       -.6021
Zn    1.973569  1  110.842228  1  180.000001  0    5    4    3        .5770
 O    1.973569  1  110.842228  1  180.000001  0    6    5    4       -.6564
Zn    1.973000  1  110.866497  1   56.427279  0    1    2    3        .6144
 O    1.973569  1  110.865954  1  -56.440985  0    8    1    2       -.6113
Zn    1.973000  1  110.866497  1  -56.427279  0    3    2    1        .6225
 O    1.973569  1  110.866415  1  180.000132  0   10    3    2       -.6080
Zn    1.973000  1  110.866184  1  -56.427099  0    5    4    3        .5856
 O    1.973569  1  110.866184  1  180.000000  0   12    5    4       -.6064
Zn    1.973000  1  110.866184  1  -56.427099  0    7    6    5        .5391
 O    1.973569  1  110.866184  1  180.000000  0   14    7    6       -.6128
Zn    1.973000  1  110.866184  1  180.000000  0   15   14    7        .5430
 O    1.973569  1  110.866184  1   56.440853  0   16   15   14       -.6735
Zn    1.973000  1  110.866184  1  180.000000  0   13   12    5        .5374
 O    1.973569  1  110.866184  1   56.440853  0   18   13   12       -.3380
Zn    1.973000  1  110.866184  1  180.000132  0   11   10    3        .5443
 O    1.973576  1  110.866102  1   56.440721  0   20   11   10       -.6835
Zn    1.973000  1  110.866184  1  179.999868  0    9    8    1        .5833
 O    1.992000  1  108.049441  1  -61.779573  0   22    9    8       -.5962
Zn    1.973561  1  108.044132  1 -120.010680  0   23   22    9        .6570
 O    1.973576  1  110.842228  1   61.796636  0   24   23   22       -.5028
Zn    1.973569  1  110.842376  1  180.000000  0   25   24   23        .3672
 O    1.973569  1  110.842228  1  180.000000  0   26   25   24       -.4964
Zn    1.973569  1  110.842228  1  180.000000  0   27   26   25        .6350
 O    1.973569  1  110.842228  1  180.000000  0   28   27   26       -.5392
Zn    1.973000  1  110.866184  1   56.427099  0   29   28   27        .5970
 O    1.973569  1  110.866184  1  -56.440853  0   30   29   28       -.4866
Zn    1.973000  1  110.866184  1   56.427099  0   27   26   25        .5804
 O    1.973569  1  110.866184  1  -56.440853  0   32   27   26       -.5721
Zn    1.973000  1  110.866102  1   56.427052  0   25   24   23        .5929
 O    1.973569  1  110.866184  1  -56.440721  0   34   25   24       -.5898
Zn    1.973000  1  108.049441  1     .000000  0   23   22    9        .6156
 O    1.973569  1  110.866184  1   61.779573  0   36   23   22       -.6234
Zn    1.973000  1  110.865954  1  180.000132  0   37   36   23        .6875
 O    1.973561  1  110.866497  1   56.440985  0   38   37   36       -.6233
Zn    1.973000  1  110.866415  1  179.999868  0   35   34   25        .6108
 O    1.973576  1  110.865872  1   56.440721  0   40   35   34       -.6471
Zn    1.973000  1  110.866184  1  180.000000  0   33   32   27        .5988
 O    1.973569  1  110.866184  1   56.440853  0   42   33   32       -.5897
Zn    1.973000  1  110.866184  1  180.000000  0   31   30   29        .6401
 0     .000000  1     .000000  1     .000000  0    0    0    0
 2,  1,  3,  4,  5,  6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
 2,  1,  20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
 2,  1,  36, 37, 38, 39, 40, 41, 42, 43, 44,
 3,  2,  4,  5,  6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
 3,  2,  20, 21, 22, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
 3,  2,  37, 38, 39, 40, 41, 42, 43, 44,
 23,  2,  24, 36,
 4,  3,  5,  6,  7,  26,  27, 28,  29,
 11,  3,  13,  15, 16, 18, 20, 22, 38, 42, 44, 40,
 8,  3, 17, 19, 21, 30, 32, 34, 39, 41, 43,
 9,  3,  10, 12, 14, 31, 33, 35,
 25, 3, 37
```

Input 04.dat :

```
AM1 PREC SYMMETRY T=200000
(ZnO)22 modelo II.1H2O(30)

 O     .000000  0     .000000  0     .000000  0    0    0    0       -.7115
Zn    1.973561  0     .000000  0     .000000  0    1    0    0        .6184
 O    1.973561  0  110.841931  0     .000000  0    2    1    0       -.6092
Zn    1.973576  0  110.842228  0  180.000000  0    3    2    1        .6275
 O    1.973569  0  110.842376  0  180.000000  0    4    3    2       -.6023
Zn    1.973569  0  110.842228  0  180.000001  0    5    4    3        .5733
 O    1.973569  0  110.842228  0  180.000001  0    6    5    4       -.6579
Zn    1.973000  0  110.866497  0   56.427279  0    1    2    3        .6135
 O    1.973569  0  110.865954  0  -56.440985  0    8    1    2       -.6117
Zn    1.973000  0  110.866497  0  -56.427279  0    3    2    1        .6222
 O    1.973569  0  110.866415  0  180.000132  0   10    3    2       -.6088
Zn    1.973000  0  110.866184  0  -56.427099  0    5    4    3        .5836
 O    1.973569  0  110.866184  0  180.000000  0   12    5    4       -.6038
Zn    1.973000  0  110.866184  0  -56.427099  0    7    6    5        .5356
 O    1.973569  0  110.866184  0  180.000000  0   14    7    6       -.6932
Zn    1.973000  0  110.866184  0  180.000000  0   15   14    7        .5175
 O    1.973569  0  110.866184  0   56.440853  0   16   15   14       -.6898
Zn    1.973000  0  110.866184  0  180.000000  0   13   12    5        .5351
 O    1.973569  0  110.866184  0   56.440853  0   18   13   12       -.3352
Zn    1.973000  0  110.866184  0  180.000132  0   11   10    3        .5444
 O    1.973576  0  110.866102  0   56.440721  0   20   11   10       -.6861
Zn    1.973000  0  110.866184  0  179.999868  0    9    8    1        .5818
 O    1.992000  0  108.049441  0  -61.779573  0   22    9    8       -.5982
Zn    1.973561  0  108.044132  0 -120.010680  0   23   22    9        .6554
 O    1.973576  0  110.842228  0   61.796636  0   24   23   22       -.5024
Zn    1.973569  0  110.842376  0  180.000000  0   25   24   23        .3574
 O    1.973569  0  110.842228  0  180.000000  0   26   25   24       -.4950
Zn    1.973569  0  110.842228  0  180.000000  0   27   26   25        .6315
 O    1.973569  0  110.842228  0  180.000000  0   28   27   26       -.6150
Zn    1.973000  0  110.866184  0   56.427099  0   29   28   27        .6031
 O    1.973569  0  110.866184  0  -56.440853  0   30   29   28       -.4984
Zn    1.973000  0  110.866184  0   56.427099  0   27   26   25        .5828
 O    1.973569  0  110.866184  0  -56.440853  0   32   27   26       -.5717
Zn    1.973000  0  110.866102  0   56.427052  0   25   24   23        .5913
 O    1.973569  0  110.866184  0  -56.440721  0   34   25   24       -.5901
Zn    1.973000  0  108.049441  0     .000000  0   23   22    9        .6151
 O    1.973569  0  110.866184  0   61.779573  0   36   23   22       -.6240
Zn    1.973000  0  110.865954  0  180.000132  0   37   36   23        .6857
 O    1.973561  0  110.866497  0   56.440985  0   38   37   36       -.6240
Zn    1.973000  0  110.866415  0  179.999868  0   35   34   25        .6100
 O    1.973576  0  110.865872  0   56.440721  0   40   35   34       -.6484
Zn    1.973000  0  110.866184  0  180.000000  0   33   32   27        .5976
 O    1.973569  0  110.866184  0   56.440853  0   42   33   32       -.5922
Zn    1.973000  0  110.866184  0  180.000000  0   31   30   29        .6322
 O    2.000426  1   90.000000  1  151.779573  1   30   29   28       -.3354
 H     .942898  1  101.984429  1   -8.867645  1   45   30   29        .2924
 H     .982203  1  106.377379  1  100.229929  1   45   30   29        .2964
 0     .000000  0     .000000  0     .000000  0    0    0    0
 46,  1, 47
```

Input 05.dat :

```
PREC AM1  T=400000
(ZnO)22 modelo II.1CO(28)

 O     .000000  0     .000000  0     .000000  0    0    0    0       -.7113
Zn    1.973561  0     .000000  0     .000000  0    1    0    0        .6187
 O    1.973561  0  110.841931  0     .000000  0    2    1    0       -.6093
Zn    1.973576  0  110.842228  0  180.000000  0    3    2    1        .6277
 O    1.973569  0  110.842376  0  180.000000  0    4    3    2       -.6026
Zn    1.973569  0  110.842228  0  180.000001  0    5    4    3        .5735
 O    1.973569  0  110.842228  0  180.000001  0    6    5    4       -.6594
Zn    1.973000  0  110.866497  0   56.427279  0    1    2    3        .6135
 O    1.973569  0  110.865954  0  -56.440985  0    8    1    2       -.6116
Zn    1.973000  0  110.866497  0  -56.427279  0    3    2    1        .6222
 O    1.973569  0  110.866415  0  180.000132  0   10    3    2       -.6083
Zn    1.973000  0  110.866184  0  -56.427099  0    5    4    3        .5849
 O    1.973569  0  110.866184  0  180.000000  0   12    5    4       -.6063
Zn    1.973000  0  110.866184  0  -56.427099  0    7    6    5        .5378
 O    1.973569  0  110.866184  0  180.000000  0   14    7    6       -.6307
Zn    1.973000  0  110.866184  0  180.000000  0   15   14    7        .5367
 O    1.973569  0  110.866184  0   56.440853  0   16   15   14       -.7115
Zn    1.973000  0  110.866184  0  180.000000  0   13   12    5        .5377
 O    1.973569  0  110.866184  0   56.440853  0   18   13   12       -.3443
Zn    1.973000  0  110.866184  0  180.000132  0   11   10    3        .5450
 O    1.973576  0  110.866102  0   56.440721  0   20   11   10       -.6870
Zn    1.973000  0  110.866184  0  179.999868  0    9    8    1        .5818
 O    1.992000  0  108.049441  0  -61.779573  0   22    9    8       -.5978
Zn    1.973561  0  108.044132  0 -120.010680  0   23   22    9        .6544
 O    1.973576  0  110.842228  0   61.796636  0   24   23   22       -.5045
Zn    1.973569  0  110.842376  0  180.000000  0   25   24   23        .3468
 O    1.973569  0  110.842228  0  180.000000  0   26   25   24       -.5216
Zn    1.973569  0  110.842228  0  180.000000  0   27   26   25        .5881
 O    1.973569  0  110.842228  0  180.000000  0   28   27   26       -.5833
Zn    1.973000  0  110.866184  0   56.427099  0   29   28   27        .5982
 O    1.973569  0  110.866184  0  -56.440853  0   30   29   28       -.4835
Zn    1.973000  0  110.866184  0   56.427099  0   27   26   25        .5821
 O    1.973569  0  110.866184  0  -56.440853  0   32   27   26       -.5722
Zn    1.973000  0  110.866102  0   56.427052  0   25   24   23        .5917
 O    1.973569  0  110.866184  0  -56.440721  0   34   25   24       -.5901
Zn    1.973000  0  108.049441  0     .000000  0   23   22    9        .6151
 O    1.973569  0  110.866184  0   61.779573  0   36   23   22       -.6242
Zn    1.973000  0  110.865954  0  180.000132  0   37   36   23        .6859
 O    1.973561  0  110.866497  0   56.440985  0   38   37   36       -.6237
Zn    1.973000  0  110.866415  0  179.999868  0   35   34   25        .6099
 O    1.973576  0  110.865872  0   56.440721  0   40   35   34       -.6481
Zn    1.973000  0  110.866184  0  180.000000  0   33   32   27        .5978
 O    1.973569  0  110.866184  0   56.440853  0   42   33   32       -.5923
Zn    1.973000  0  110.866184  0  180.000000  0   31   30   29        .6357
 C    2.127305  1  120.323889  1   52.498669  1   28   27   26        .2676
 O    1.165038  1  174.090419  1   49.888957  1   45   28   27       -.0292
 0     .000000  0     .000000  0     .000000  0    0    0    0
```

Referències

[1] Alan Hinchliffe *1988 Computational Quantum Quemistry* . Ed. J. Wiley & Sons

[2] Abderrahim Qrichi Aniba *1994 Implémentation performante du BLAS de niveau 3 pour les processeurs RISC*. TR/PA/94/11 CERFACS

[3] Jack J. Dongarra, Peter Mayes, Giuseppe Radicatti di Brozzolo *1991 The IBM RISC System/6000 and Linear Algebra Operations*

[4] IBM Corp. *1993 AIX Version 3.2 for RISC System/6000. Optimization and Tuning Guide for Fortran, C, and C++*.