PIPS : un système d'analyse, de transformation et de compilation de programmes scientifiques

Corinne Ancourt
Fabien Coelho
François Irigoin
Ronan Keryell
} @cri.ensmp.fr

et moult autres contributeurs

—

Centre de Recherche en Informatique de

l'École des Mines de Paris

22 février 2008

PIPS = Paralléliseur Interprocédural de Programmes Scientifiques

- Projet démarré en 1988 avec financement DRET, puis ESPRIT, CNRS & École des Mines
- Faire avancer l'état de l'art
  - Analyse sémantique
  - Parallélisation & vectorisation automatique

  ⇝ Effet de bord : contribution séminale à « l'école française polyédrique » ☼
- Franchir le mur de l'interprocéduralité
- Travailler sur des programmes réels
  - ⇝ Fortran ☺

Aujourd'hui...

- Ouverture du cadre de la parallélisation
- Programmes complets
- Interprocédural
- Analyse ⇝ rétro-ingéniérie & vérification
- Transformations
- Compilation
- Optimisation (parallélisation = cas particulier)
- Faciliter le développement d'outils et extensibilité

Motivation économique

- Coût d'exécution, latence, taille des modèles 3D, faisabilité
  $\searrow$ *coût d'exécution* $\Rightarrow$ $\searrow$ *coût de maintenance matériel,* $\searrow$ *coût d'administration,...*
- Potentiel croissant : options de compilation, ILP, localité, parallélisme *gain de 1,2 en 1978, de 2 à 5 en 1998*
- Accélération de 2 = 18 mois *time-to-market*
- Choix de l'architecture matérielle (PE, SoC, MP-SoC...)
- SAGEM, IBM, LMC, RP-RORER,...

À la main ou automatiquement :

- Codes industriels : SAGEM, Renault, KFA, IFP, CEA, RP-RORER
- Académiques : ENSMP/CIG, CG
- Cell, GPGPU, IBM SP2, CRAY T3D, Suns, Maspar, Convex MPP1200
  Alliant, CRAY YMP, TMC CM5, IBM 3090...
- Gains de 2 à 50 en séquentiel
- Gains de 2 à 16 000 en parallèle (selon machine !)

- IBM SP2, Power2
- Gain séq : 2,5 (= 110 Mflop/s)
  - Re-parenthésage d’une expression
  - Déroulage d’une boucle (4)
  - Ordonnancement des calculs
- HPF : gain de 2,5 (4 PE, avec I/O)

```
!hpf$ independent
      DO J = 3,NP-2
!hpf$   independent
        DO I = 3, NP-2
          U(I,J,KP) =
            (2. * U(I,J,KM) - U(I,J,KP))
           - V(I,J) * ( 60. * U(I,J,KM)
           + (U(I+2,J,KM) + U(I-2,J,KM)
             + U(I,J-2,KM) + U(I,J+2,KM))
           - 16. * (U(I+1,J,KM) + U(I-1,J,KM)
                  + U(I,J-1,KM) + U(I,J+1,KM)))
        ENDDO
      ENDDO
```

- Traitement d'empreintes par logiciel
- Portage/optimisation : SUN, IBM, Maspar MP2, Convex, MMX, SSE...
- Gain séquentiel : 10, parallèle 16 000
  - Compactage des données, ILP, pipeline logiciel
  - Ordonnancement et fusion de boucles

1. Analyse de l'application : complexité,...
2. Changements d'algorithmes (en amont)
   e.g. méthode directe $\rightarrow$ gradient conjugué
3. Utilisation de librairies optimisées
   IMSL, Lapack, ESSL, ScaLapack, P-ESSL
4. Options de compilation
   IBM `xlf` : +150 options et sous-options !
5. Transformations de programmes
   - Selon langage, application, architecture
   - Futurs pragma ou options de compilation

- Processeurs : moult unités fonctionnelles, chargements doubles ou SIMD, multicœurs...
- Hiérarchie mémoire : ordres de grandeur en temps d'accès registre (0) $\leftrightarrow$ cache 1 (3) $\leftrightarrow$ cache 2 (10) $\leftrightarrow$ mémoire (80) $\leftrightarrow$ mémoire distante (1000) $\leftrightarrow$ disque (1000000)...
- Multiprocesseurs : volumes de mémoire et de cache plus grands $\rightsquigarrow$ parfois effets superlinéaires

Validité, décision, application

- Inversion, fusion, distribution... de boucles
- *Tiling* des itérations
- Restructuration du contrôle
- Restructuration d'expressions
- Annotations HPF pour le parallélisme : HPFC
- Annotation pour la distribution (PHRASE, SAFESCALE, FREIA)

Problèmes :

- Connaissances très spécifiques nécessaires
- Dégradation de la qualité du source optimisé...
- Portabilité des optimisations ?

Conséquences :

- Coût de développement et coût de maintenance (KLOCs)
- Source à source (PIPS)
- Éventuellement assistée de conseils (*pragma*)

Conclusion : importance de l'automatisation

- Rétro-ingéniérie
  - Compréhension de code
  - Simplification de code
  - Spécialisation de code
  - Vérification, test, preuve
- Synthèse de code & compilation de spécifications
- Optimisation d’expressions
- Environnement de développement

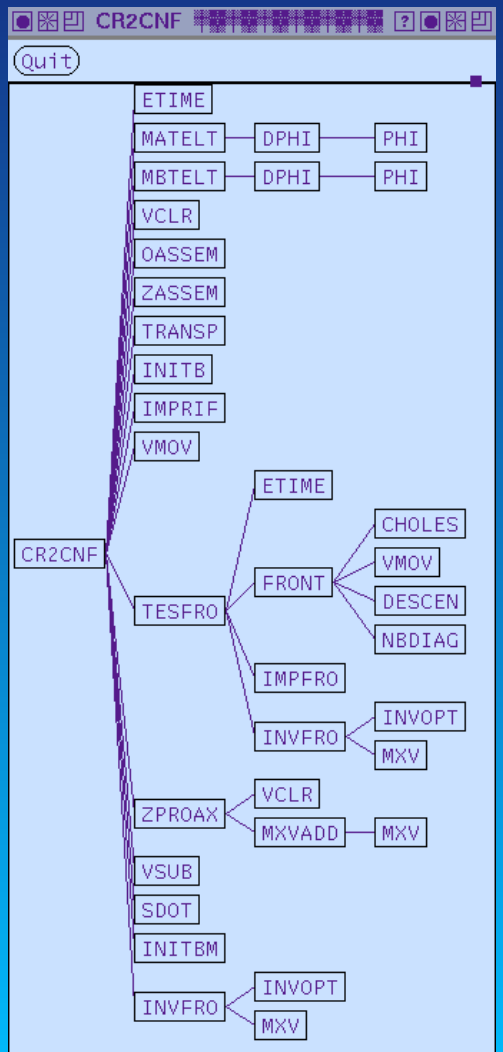
Motivation économique

- Beaucoup de codes existent dans l'industrie
- Capital souvent important (100k-1M lignes)
- Coûts importants en maintenance et évolution
- Estimation des coûts de maintenance contre coûts de développement
- Vérification de programme, certification, test
- Vérification de codes importés : *parallélisme, débordement de tableau*
- Détections d'erreurs (allocation,...)

- Utile pendant tout le cycle de vie : codage, mise au point, optimisation, certification, maintenance, réingéniérie
- Bug de l'an 2000
- EDF, CEA-DAM

Outils de rétro-ingéniérie nécessaires

Graphe d'appel



- Indispensable pour appréhender un gros code
- Outil de navigation

Détection de passage de paramètres cachés

```
C  <T(PHI1,PHI2,PHI3)-IN-EXACT-{PHI1==J, PHI1==JP, KP<=PHI2+1,
C    PHI2<=KP, L<=PHI3, PHI3<=2+L, K==KP-1, 2<=J, 3<=KP}>
      REAL*4 FUNCTION D(J,K,JP,KP)
C*************************************
C     CALCULE D=DISTANCE
C*************************************
C
      INTEGER J,K,JP,KP,L
      REAL*4 D,T(1:52,1:21,1:60)
      COMMON /CNI/ L
      COMMON /CT/ T
...
```

Epips

Buffers Files Tools Edit Search Epips Help

```
C   <X(PHI1,PHI2)-OUT-MAY-{1<=PHI1, PHI1<=3NP, PHI2<=NC, N==3NP}>
C   <Y(PHI1,PHI2)-OUT-MAY-{1<=PHI1, PHI1<=3NP, 2<=PHI2, PHI2<=NC,
C     N==3NP}>
C   <Z(PHI1)-OUT-MAY-{1<=PHI1, PHI1<=3NP, N==3NP, 2<=NC}>

C   <A(PHI1,PHI2,PHI3)-IN-MAY-{1<=PHI1, PHI1<=3NP, 1<=PHI2, PHI2<=3NP,
C     2<=PHI3, PHI3<=3, N==3NP, 2<=NC}>
C   <AF(PHI1,PHI2,PHI3)-IN-MAY-{1<=PHI1, PHI1<=3NP, 1<=PHI2, PHI2<=3NP,
C     1<=PHI3, PHI3<=NC, 2PHI3+3<=2NC+3NP, N==3NP}>
C   <Y(PHI1,PHI2)-IN-MAY-{1<=PHI1, PHI1<=3NP, 1<=PHI2, N==3NP}>

C***********************************************************************
      SUBROUTINE INVFRO(A,AF,Y,N,NC,X,Z,NP)
C ROUTINE DE RESOLUTION PAR DESCENTE REMONTEE FRONTALE DU SYSTEME A*X=Y
C  CALCUL DU SECOND MEMBRE MODIFIE : DESCENTE .
      INTEGER N,NC,NP,I,NNS
      REAL*8 A(1:N,1:N,1:3),AF(1:N,1:N,1:NC),Y(1:N,1:NC),X(1:N,1:NC),
     &Z(1:N),Z1(1:640),Z2(1:640),Z3(1:640)
      COMMON /TOTO/ Z1,Z2,Z3
```

- Aucune valeur de `Z1`, `Z2` ou `Z3` nécessaire (pas de région IN)
- Aucune valeur de `Z1`, `Z2` ou `Z3` utilisé ultérieurement (pas de région OUT)
- Utilisation localisée du `COMMON /TOTO/` par rapport à l'application complète CR2CNF

- Cerner la durée de vie de variables scalaires ou tableaux
  ⇝ structuration & modularité
- Transformation de programme sur scalaires, tableaux, COMMONs

```
      SUBROUTINE INVFRO(A,AF,Y,N,NC,X,Z,NP)
      REAL*8 Z2_P(1:640),Z1_P(1:640),A(1:N,1:N,1:3),AF(1:N,1:N,
     &Y(1:N,1:NC),X(1:N,1:NC),Z(1:N),Z1(1:640),Z2(1:640),Z3(1:6
      COMMON /TOTO/ Z1,Z2,Z3
```

  Remplacement des références à Z1 & Z2 par Z1_P & Z2_P

```
EXTRMAIN
  if
  then
C  <T(PHI1,PHI2,PHI3)-R-MAY-{J1<=PHI1, PHI1<=JA, 1<=PHI2, PHI2<=3+K1,
C    PHI2<=21, J2==2JA-1, 2<=J1, 2<=K1}>
C  <T(PHI1,PHI2,PHI3)-W-MAY-{PHI2==1, J1<=PHI1, PHI1+1<=2JA,
C    3+NC<=PHI3, PHI3<=5+NC, J2==2JA-1, 2<=J1, J1<=JA, 2<=K1}>
      EXTR
          do J
      C  <T(PHI1,PHI2,PHI3)-R-EXACT-{PHI1==J, PHI1==JP, KP<=PHI2+1,
      C    PHI2<=KP, L<=PHI3, PHI3<=2+L, K==KP-1, 2<=J, 3<=KP}>
             D
      C  <T(PHI1,PHI2,PHI3)-R-EXACT-{PHI1==J, PHI1==JP, KP<=PHI2+1,
      C    PHI2<=KP, L<=PHI3, PHI3<=2+L, K==KP-1, 2<=J, 3<=KP}>
             D
      C  <T(PHI1,PHI2,PHI3)-R-EXACT-{PHI1==J, PHI1==JP, KP<=PHI2+1,
      C    PHI2<=KP, L<=PHI3, PHI3<=2+L, K==KP-1, 2<=J, 3<=KP}>
```

```
                D
            enddo
endif
```

```
do i = ...
  a(i) = ...                    a(1:1000) = ...
  b(3*i + 4) = ...      ⇝       b(7:3004:3) = ...
enddo
```

- Effet de bord de la vectorisation
- Casser des grosses boucles en instructions élémentaires
- Retrouver formalisme mathématique à base de vecteurs & matrices

- Code parallélisé $\equiv$ manipulation point à point de champs de données
- Parallélisation $\equiv$ slicing par indépendance des itérations...
- Détection des conflicts entre itération : régions Read & Write, AILE :EXTR

```
            DO 300 J = J1, JA
C  <T(PHI1,PHI2,PHI3)-R-EXACT-{J==PHI1, PHI1<=52, K<=PHI2, PHI2<=1+K,
C    PHI2<=21, L<=PHI3, 1<=PHI3, PHI3<=2+L, PHI3<=60, J2==2JA-1,
C    K==K1, L==NI, J1<=J, J<=JA, 2<=J1, 2<=K}>
               S1 = D(J, K, J, K+1)
               ...
C  <T(PHI1,PHI2,PHI3)-W-EXACT-{PHI1==J, PHI2==1, PHI3==NC+3,
C    J2==2JA-1, K==K1, L==NI, J1<=J, J<=JA, 2<=J1, 2<=K1}>
               T(J,1,NC+3) = S2*S3/((S1-S2)*(S1-S3))
               ...
C  <T(PHI1,PHI2,PHI3)-R-EXACT-{PHI1==J, PHI2==1, PHI3==NC+3,
C    J+JH==J1+2JA-1, J2==2JA-1, K==K1, L==NI, J1<=J, J<=JA, 2<=J1,
C    2<=K1}>
C  <T(PHI1,PHI2,PHI3)-W-EXACT-{PHI1==JH, PHI2==1, PHI3==NC+3,
C    J+JH==J1+2JA-1, J2==2JA-1, K==K1, L==NI, J1<=J, J<=JA, 2<=J1,
C    2<=K1}>
               T(JH,1,NC+3) = T(J,1,NC+3)
```

```
300       CONTINUE
```

- Privatisation dans une itération (régions In/Out) + parallélisation à gros grain (régions Read/Write). OA118

```
CHPF$ INDEPENDENT, NEW(JI,JE,I,O(1:NFAC),PHW(1:NFAC))
      DO 998 J = 1, NKJ
         CALL PHWAK(PHW, J)
         CALL GRAD1(PHW, O)
CHPF$    INDEPENDENT, NEW(JI,JE)
         DO 116 I = 1, NKJ
            JE = JEX(I)
            JI = JIX(I)
            IF (1.LE.JE.AND.JE.LE.NFAC.AND.1.LE.JI.AND.JI.LE.NFAC)
     &       CC(J,I) = O(JE)-O(JI)
116         CONTINUE
998      CONTINUE
```

Motivation économique

- Coût des tests (Ariane 4 $\rightsquigarrow$ Ariane 5)
- Coût de développement
- Aérospatiale, Dassault-Aviation, EDF, ESA,...

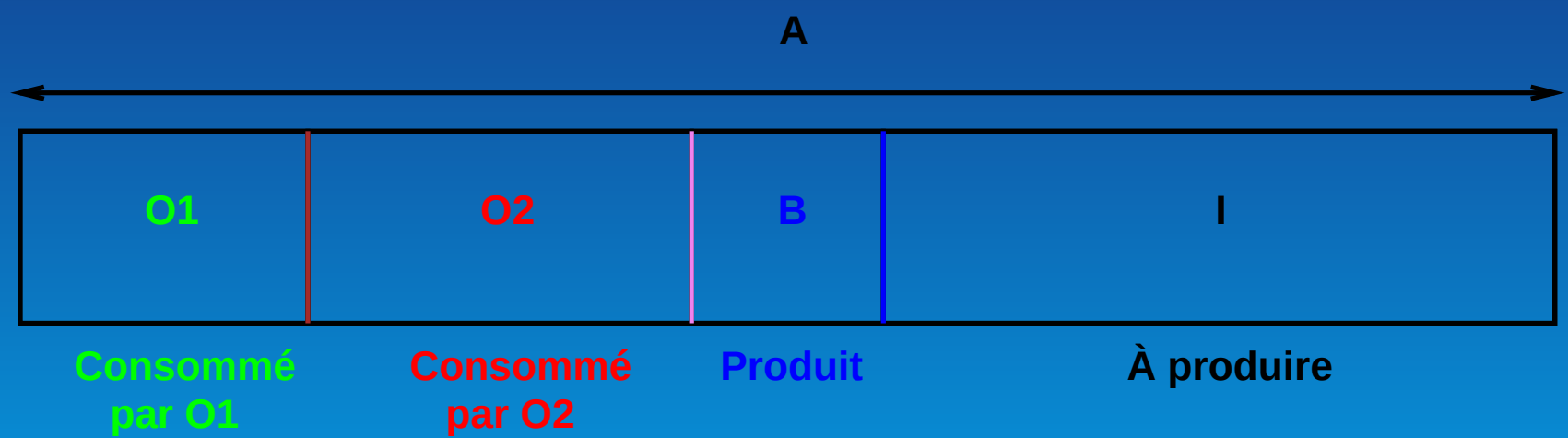
- Compréhension avec expressions symboliques
- Préconditions sur les variables scalaires entières
- Détection des variables inductives
- Complexité : $\mathcal{O}(N^2M) \rightsquigarrow$ 2 boucles en N & 1 en M imbriquées, preuve de terminaison

- Compréhension
  - Extraction d'invariants
  - Bornes sur variables
  - Preuves de propriétés
- Optimisation & spécialisation : élimination de code mort, évaluation partielle,...

Exemple de système de transition : 1 producteur (`b`) & 2 consommateurs (`o1` & `o2`)

Invariant

$$A = O_1 + O_2 + B + I$$



Bonne terminaison quand tout a été produit et que le buffer est vide

$$A = O_1 + O_2$$

```
      program pcpugh
c producer-consumer algorithm
      integer a, i, b, o1, o2
      real x
      data b, o1, o2 /0, 0, 0/
c Ask for the production amount
      read *, a
      if (a.lt.1) then
         stop
      endif
      i = a
      do k = 1, n
c        External event: 0,1,2
         read *, x
         if (x.gt.1.) then
c           Producer
            if (i.gt.0) then
               i = i - 1
               b = b + 1
            endif
         else if (x.lt.1.) then
c           Consumer 1
            if (b.gt.0) then
               o1 = o1 + 1
               b = b - 1
            endif
         else
c           Consumer 2
            if (b.gt.0) then
               o2 = o2 + 1
               b = b - 1
            endif
         endif
         if (i.eq.0 .and. b.eq.0) then
            print *,'The End',o1,o2,a
         endif
      enddo
      end
```

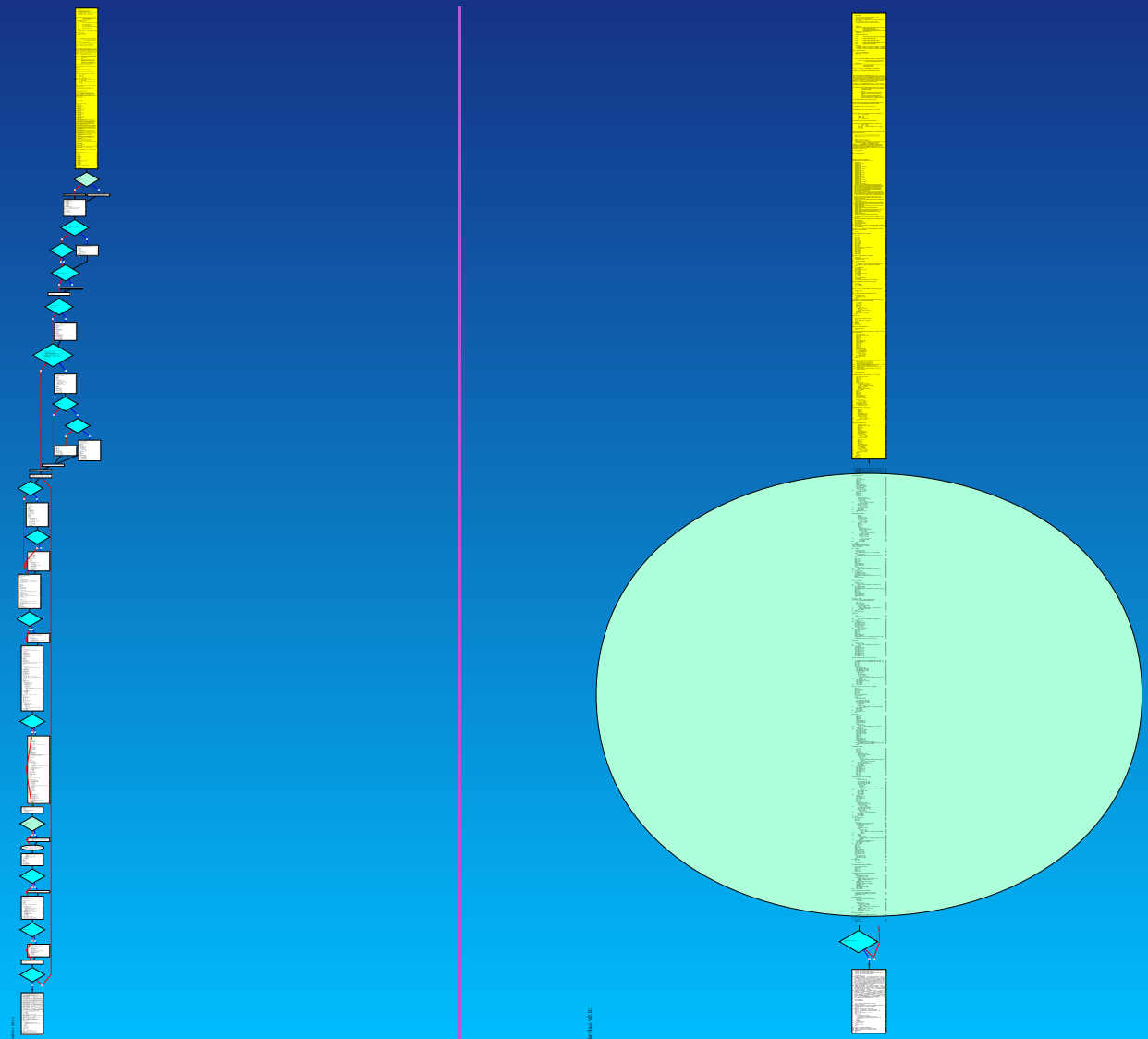
```
      PROGRAM PCPUGH
      ...
      DO K = 1, N
C  P(A,B,I,K,O1,O2) {B+I+O1+O2==A, 1<=A,
C    B+2O1+2O2+1<=K, 0<=B+O1+O2,
C    1<=K, K<=N, 0<=O1, 0<=O2}
      ...
      IF (I.EQ.0.AND.B.EQ.0) THEN
C  P(A,B,I,K,O1,O2) {O1+O2==A, B==0, I==0,
C    1<=A, A+O1+O2<=K, K<=N,
C    O1+O2+1<=K, 0<=O1, 0<=O2}
         PRINT *, 'The End', O1, O2, A
      ENDIF
      ENDDO
```

Les invariants peuvent être réinterprétés :

- $0 \leq B+O_1+O_2 \Longleftrightarrow I \leq A$
- $0 \leq O_1, 0 \leq O_2 \Longleftrightarrow B+I \leq A$
- $B+2\times O_1+2\times O_2+1 \leq K$: nombre de transitions effectuées $<$ nombre de transitions essayées

- Élimination des « scories » de mise au point dans les vieux codes
- Renormalisation (graphe de contrôle, détection de boucles)
- Adaptation du logiciel à son utilisation réelle
- Facilite la rétro-ingéniérie

Restructuration du graphe de contrôle



OCEAN (PerfectClub) : 16 tests non structurés $\rightsquigarrow$ 15 tests restructurés

- 3 `IF/THEN/ELSE`
- 0 `IF/THEN` (branche `ELSE` vide)
- 11 `IF/ELSE` (branche `THEN` vide)
- 1 `IF` vide !

```
2       IF (LMAPP.NE.2) THEN
           GOTO 3
        ENDIF
3       CONTINUE
```

- Transformer plat de spaghetti en `IF`/`THEN`/`ELSE` sans `GOTO`, `WHILE`,...
- Techniques des années 1960
- Bénéficier de la programmation structurée
  - Lisibilité
  - Maintenance
- Disponibles dans d'autres outils (Foresys, etc)
- Amélioration de la précision d'analyses ultérieures

Exemple d'OCEAN (PerfectClub) : 57 tests

```
Epips
Buffers Files Tools Edit Search Epips Help
C  P() {NFTVMT==0, NUM1==1, NUSHUF==0, NXACAC==0, NXLOG2==0,
C    NXPRNT==0, NXSCSC==0, NXSHUF==0, NXXCSR==0, NXXOUT==0,
C    NXXRCS==0, NXXXIN==0, TEMPHY:NCALL==0, ZETAPH:NCALL==0}
      CALL START(NX, NY)                                          0006
C  P(KDATA,KPF,KPP,KPRESS,KSM,KTA,KTAPE,KTF,KTP,KUF,KUP,KVF,KVP,KZF,
C    KZP,LMAPP,N1,N2,NOALS,NOTEMP,NUMBER) {KDATA==1, KPF==1,
C    KPP==249, KPRESS==1, KSM==1, KTA==50, KTAPE==0, KTF==1,
C    KTP==50, KUF==1, KUP==249, KVF==1, KVP==249, KZF==1, KZP==249,
C    LMAPP==1, N1==128, N2==128, NFTVMT==0, NOALS==1, NOTEMP==0,
C    NUM1==1, NUMBER==250, NUSHUF==0, NXACAC==0, NXLOG2==0,
C    NXPRNT==0, NXSCSC==0, NXSHUF==0, NXXCSR==0, NXXOUT==0,
C    NXXRCS==0, NXXXIN==0, TEMPHY:NCALL==0, ZETAPH:NCALL==0}
C
C IF  LMAPP = 1 , H  (THE SIZE OF THE BOX IN Y DIRECTION)  SHOULD BE
C SPECIFIED IN THE MAIN PROGRAM.
C
      H = PI                                                      0007
C  P(KDATA,KPF,KPP,KPRESS,KSM,KTA,KTAPE,KTF,KTP,KUF,KUP,KVF,KVP,KZF,
-----Emacs: EPips-0        13:35 Mail   (Fortran EPips)--L229--C3-- 2%----------
```

- Élimination des instructions infaisables : 1
- Élimination de tests toujours vrais ou faux : 25 (17 `THEN` & 8 `ELSE`)
- Élimination de boucles jamais exécutées : 0
- Garder un corps de boucle exécuté 1 seule fois : 0

$\Rightarrow$ Nombre de chemins de contrôle réduit par facteur 25–16 000 000

$\rightsquigarrow$ amélioration d'analyses ultérieures

- *Slicing* sur les E/S. OCEAN : 13 lignes supprimées
- Utilisation des chaînes In/Out possible (exemple sur code RPC)
- *Slicing* par effet de bord : enlever toutes les E/S et mettre un `print` d'une variable...

- Optimisation de code pour une architecture donnée
- Optimisation pour une utilisation donnée (bibliothèques)
- Déroulage des petites boucles internes
- Post-phase de clonage : séparation d'une routine à fonctionalités multiples (DYNA :MAKEPRF initialisation & calcul)

```
C     IF MODE=0 IN THE MIDDLE OF TIME STEP
C     KEYTAU=1
C     KEYLUM=1
C     KSTATE=-1
C     KVISCOS=0
C     IF MODE=1  IN THE END OF TIME STEP
C     KEYTAU=1
C     KEYLUM=1
C     KSTATE=1
C     KVISCOS=1
C     -----------------------------------------------
```

```
C     IF KEYTAU=0 DONT CALCULATE TAUA(J)
C     IF KEYLUM=0 DONT CALCULATE FRA(J) AND FCA(J)
C     IF KEYLUM=1 CALCULATE EXACT FRA AND FCA
C     IF KEYLUM=-1  CAL. FLUX ONLY FROM  DER.
C     IF KSTATE=-1 CALCULATE VARIABLES FROM DERIVATIVES
C     IF KSTATE=1 CALCULATE EXACT  VARIABLES FROM STATE
C     IF KVISCOS=0 DO NOT CALCULATE VISCOSITY
C     IF KVISCOS=1 CALCULATE VISCOSITY
C     ------------------------------------------------
C     IF KCONVEC=0  INSTATENIOUS  CONVECTION
C     IF KCONVEC=1  TIME DEPENDENT CONVECTION
C     ------------------------------------------------
```

```
C  P() INCX==1, INCY==1, N==49
      REAL*4 FUNCTION SDOT(N,X,INCX,Y,INCY)
      INTEGER N,INCX,INCY,IX,IY,I
      REAL*4 X(1:1),Y(1:1),SDOT

      SDOT = 0.0                          0001
      IX = 1                              0002
      IY = 1                              0003
      DO 10 I = 1, N                      0004
         SDOT = SDOT+X(IX)*Y(IY)          0005
         IX = IX+INCX                     006
         IY = IY+INCY                     0007
10       CONTINUE                         0008
      END
```

Après évaluation partielle, déroulage, évaluation partielle & élimination use-def

```
      SDOT = 0.0
      SDOT = SDOT+X(1)*Y(1)
      SDOT = SDOT+X(2)*Y(2)
      SDOT = SDOT+X(3)*Y(3)
      SDOT = SDOT+X(4)*Y(4)
      SDOT = SDOT+X(5)*Y(5)
      SDOT = SDOT+X(6)*Y(6)
      SDOT = SDOT+X(7)*Y(7)
...
      SDOT = SDOT+X(48)*Y(48)
      SDOT = SDOT+X(49)*Y(49)
```

```
C                                          46058140 (SUMMARY)
      SUBROUTINE CONVOL(NEW_IMAGE,IMAGE,ISI,ISJ,KERNEL,
     &HKSI,HKSJ)
      INTEGER ISI,ISJ,HKSI,HKSJ,I,J,KI,KJ
      REAL*4 NEW_IMAGE(1:ISI,1:ISJ),IMAGE(1:ISI,1:ISJ),
     &KERNEL(-HKSI:HKSI,-HKSJ:HKSJ),S

      DO I = 1, ISI
         DO J = 1, ISJ
            NEW_IMAGE(I,J) = IMAGE(I,J)
         ENDDO
      ENDDO
```

CONVOL générique

```
      DO 300 J = 1+HKSJ, ISJ-HKSJ
         DO 400 I = 1+HKSI, ISI-HKSI
            S = 0.
            DO 200 KI = -HKSI, HKSI
               DO 100 KJ = -HKSJ, HKSJ
                  S = S+IMAGE(I+KI,J+KJ)*KERNEL(KI,KJ)
100               CONTINUE
200            CONTINUE
            NEW_IMAGE(I,J) = S/((2*HKSI+1)*(2*HKSJ+1))
400         CONTINUE
300      CONTINUE
      END
```

Vérification accès tableaux CONVOL

```
C  <IMAGE(PHI1,PHI2)-R-EXACT-{1<=PHI1, PHI1<=ISI, 1<=PHI2, PHI2<=ISJ
C    1<=HKSI, 1<=HKSJ}>
C  <KERNEL(PHI1,PHI2)-R-EXACT-{0<=PHI1+HKSI, PHI1<=HKSI, 0<=PHI2+HKSJ
C    PHI2<=HKSJ, 1<=HKSI, 1+2HKSI<=ISI, 1<=HKSJ, 1+2HKSJ<=ISJ,
C    1<=ISJ}>
C  <NEW_IMAGE(PHI1,PHI2)-W-EXACT-{1<=PHI1, PHI1<=ISI, 1<=PHI2,
C    PHI2<=ISJ, 1<=HKSI, 1<=HKSJ}>
```

Clonage + évaluation partielle + 2 déroulages de boucle + élimination use-def :

```
C                                                    32532940 (SUMMARY)
...
      DO 300 J = 2, 511                                              0004
         DO 400 I = 2, 511                                           0005
            S = 0.                                                   0006
            S = S+IMAGE(I-1,J-1)*KERNEL(-1,-1)                       0009
            S = S+IMAGE(I-1,J)*KERNEL(-1,0)                          0009
            S = S+IMAGE(I-1,J+1)*KERNEL(-1,1)                        0009
            S = S+IMAGE(I,J-1)*KERNEL(0,-1)                          0009
            S = S+IMAGE(I,J)*KERNEL(0,0)                             0009
            S = S+IMAGE(I,J+1)*KERNEL(0,1)                           0009
            S = S+IMAGE(I+1,J-1)*KERNEL(1,-1)                        0009
            S = S+IMAGE(I+1,J)*KERNEL(1,0)                           0009
            S = S+IMAGE(I+1,J+1)*KERNEL(1,1)                         0009
```

CONVOL après optimisation

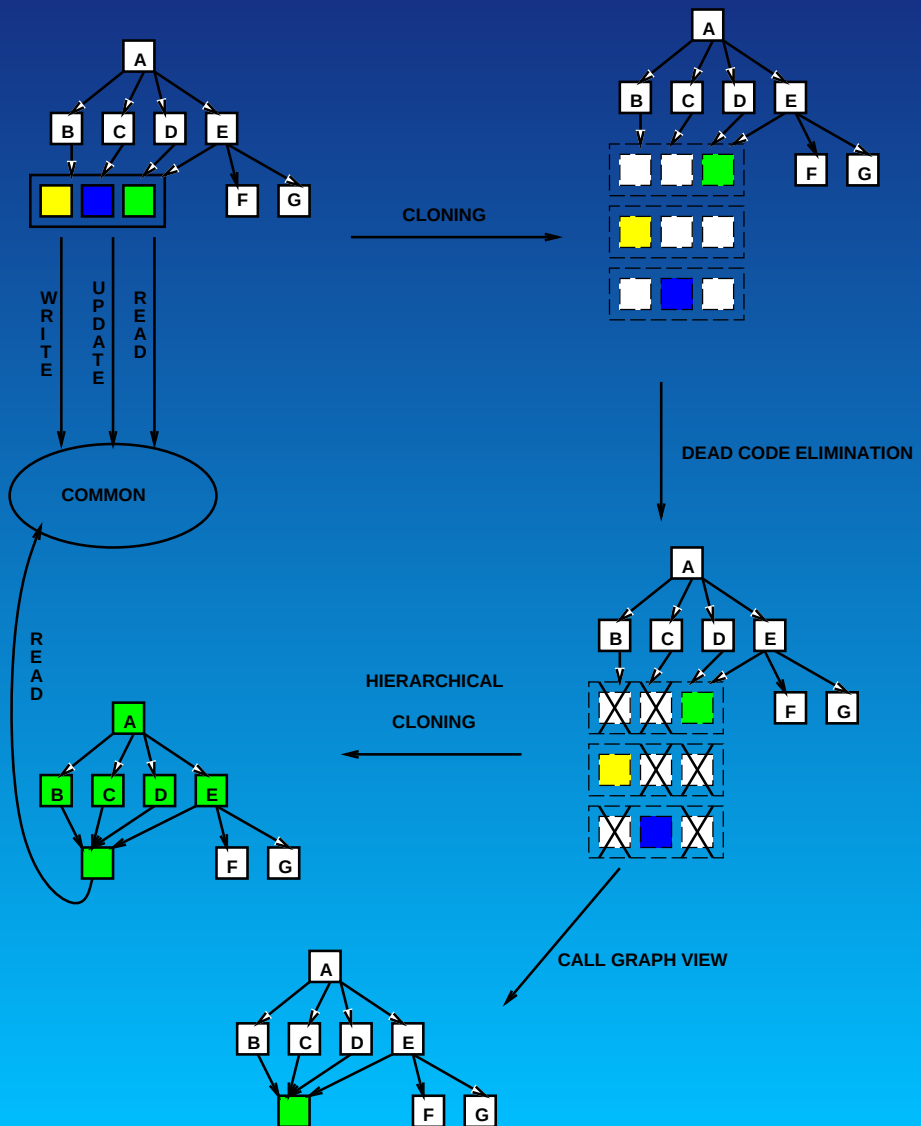
```
            NEW_IMAGE(I,J) = S/9                                    0012
400         CONTINUE
300       CONTINUE
```

	Estimation SS2 (cycles)	SuperSparc-II		UltraSPARC	
		f77 -O3	f77	f77 -O3	f77
Code initial	66358440	29.6s	195.2s	9.59s	105.3s
Code spécialisé	28612020	13.1s	69.2s	5.58s	34.0s
Rapports	2.32	2.26	2.82	1.71	3.09

Vérification d'un programme en cours de parallélisation

- Seulement 600 procédures parmi les 2000 d'une application
- Graphe des appels long de 82 pages : impossible à traiter manuellement
- 40 617 lignes de code
- La boucle *machin* dans la procédure *truc* est-elle vraiment parallèle ?

Graphe de flot de données quotient



- une variable non initialisée
- nettoyage des déclarations
  - réduction du nombre de commons : 36 %
  - réduction du nombre d’équivalence : 98 %
  - réduction du nombre de paramètres : 95 %
- vérification de la validité de la parallélisation

- Méthode polyédrique (PAF, UVSQ/PRISM)
- Pouvoir tracer les éléments de tableau
- Passage en assignation unique dynamique
- Détection des erreurs d’accès
- Restreint aux cas simples (contrôle statique, intraprocédural)

```
PARAMETER (N = 10)

DO J = 1, N
   DO I = 1, N                                  0002
      A(I,J) = REAL(I-N/2)/REAL(J)              0004
      B(I,J) = REAL(J-3)/REAL(I)                0005
   ENDDO
ENDDO
DO J = 1, N                                     0006
   DO I = 1, N                                  0008
      C(I,J) = 0.                               0010
      DO K = 1, N                               0011
         C(I,J) = C(I,J)+A(I,K)*B(K,J)          0013
      ENDDO
   ENDDO
ENDDO
```

ADFG de la multiplication de matrice

```
INS_100:
********
 Execution Domain for 100:
{
  I - 10 <= 0 ,
- I + 1 <= 0 ,
  J - 10 <= 0 ,
- J + 1 <= 0 ,
}
 ---Def-Use---> ins_130:
  Reference: C(I,J)
  Transformation: [J,I]
  Governing predicate:
{
  K - 1 <= 0 ,
}
  Execution Domain for 130:
{
  K - 10 <= 0 ,
- K + 1 <= 0 ,
  I - 10 <= 0 ,
- I + 1 <= 0 ,
  J - 10 <= 0 ,
- J + 1 <= 0 ,
}
```

- Régions IN : variable locale avec région IN $\rightsquigarrow$ erreur
- ADFG : source indéfinie $\rightsquigarrow$ erreur

- Application traitement du signal systématique
- Langage de type Fortran 90, Alpha
- Temps infini
- Séquence de nids de boucles parallèles
- Assignation unique
- Modulo

```
doall t,h
  call FFT(t,h)
enddo
doall t,f,v
  call BeamForming(t,f,v)
enddo
doall t,f,v
  call Energy(t,f,v)
enddo
doall t,v
  call ShortIntegration(t,v)
enddo
doall t,v
  call AzimutStabilization(t,v)
enddo
doall t,v
```

```
   call LongIntegration(t,v)
enddo
```

- Multiprocesseurs à mémoire distribuée
- Pas de topologie particulière
- Programmation SIMD/SPMD
  - Une tâche de calculs à la fois (temps d'exécution fixe)
  - Temps de synchronisation et communications
- Possibilité recouvrement calculs/communications

Programmation Logique par Contraintes

```
do t8 = 0, infini
  do t7 = 8*t8, 8*t8+7
    FFT
    FFT
    FORMATION DE VOIES ; ENERGIE
    do t5 = 8*t7,8*t7+7
       FFT
       FORMATION DE VOIES ;  ENERGIE
       REGROUPEMENT LARGE BANDE
    enddo
    FORMATION DE VOIES ; ENERGIE
    REGROUPEMENT LARGE BANDE
    REGROUPEMENT LARGE BANDE
    INTEGRATION COURTE
    STABILISATION EN AZIMUT
  enddo
 INTEGRATION LONGUE
```

```
 STABILISATION EN AZIMUT
enddo
```

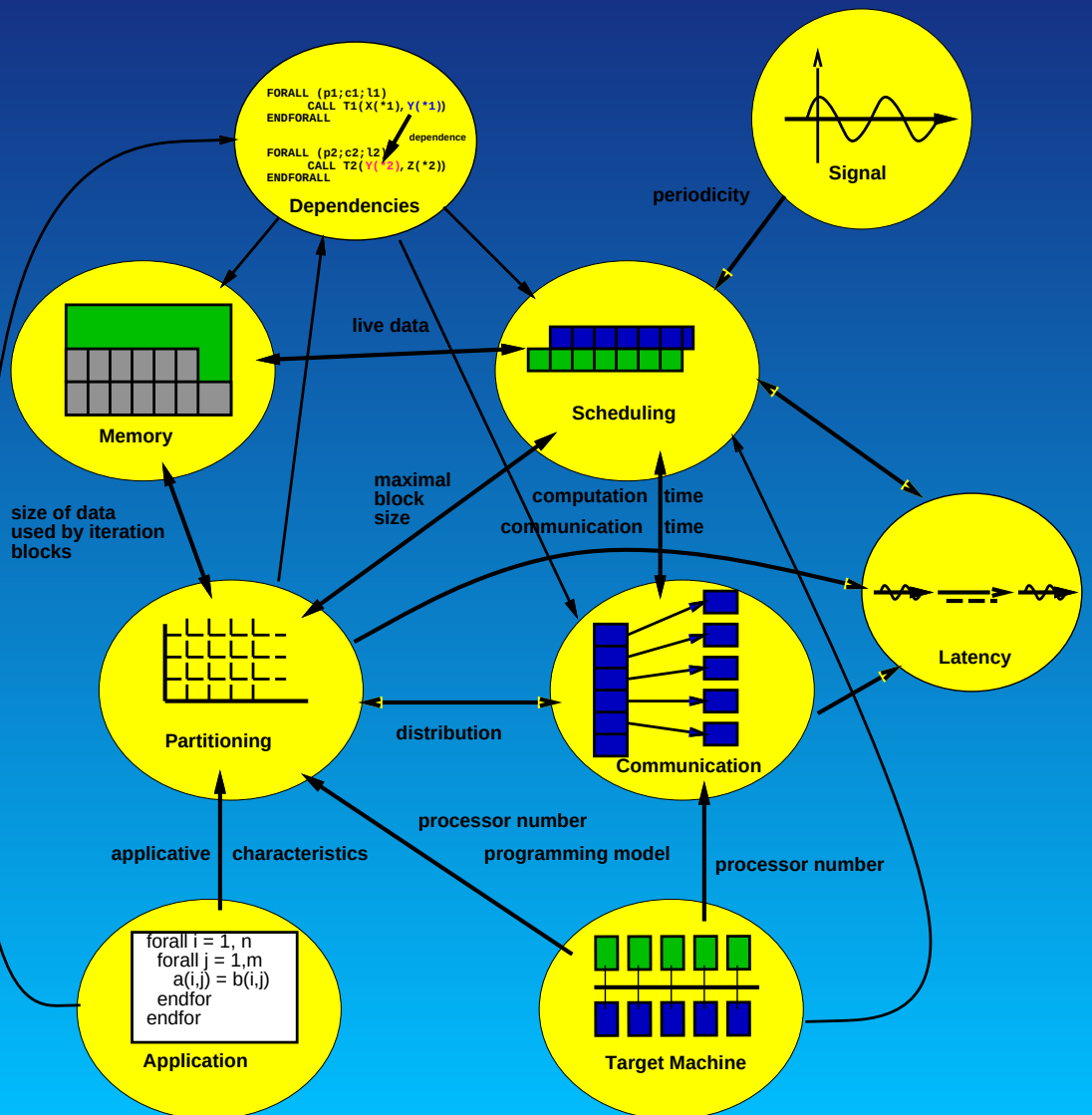
```
   do t=0,infinity
    do h=0,511
c           Read/IN Region:
c                 HYDRO(h,512*t:512*t+511)
c           Write/OUT Region:
c                 TABFFT(h,0:255,t)
     call FFTDbl(HYDRO(h,512*t:512*t+511), TABFFT(h,0:255,t))
    enddo
   enddo
```

```
real Hydro(0:N,0:511)
real Frequence(0:N,0:255,0:511)
real Voies(0:N,0:199,0:127)
real Energie(0:N,0:199,0:127)
real BL(0:N,0:127)
real INTC(0:N,0:127)
real VSTAB(0:N,0:127)
real OUTPUT(0:N,0:127)
DO t8 = 0, N
   DO t7 = 8*t8, 8*t8+7
      DO t5 = 8*t7, 8*t7 + 7
         DO t1 = 512*t5, 512*t5+511
            read 5, (Hydro(t1,h1), h1 = 0 , 511)
         DO h2 = 0, 511
            call FFTr(Frequence,t5, h2, Hydro, N)
```

```
      DO v3 = 0, 127
         DO f3 = 0, 199
            call FV(Voies, t5, f3, v3, Frequence, N)
            call MOD2(Energie(t5,f3,v3), Voies(t5,f3,v3))
      DO v5 = 0, 127
         call RtBL(BL, Energie,t5,v5,N)
   DO v6 = 0, 127
      call INTnL(INTC,t7,v6, BL, N)
   DO v7 = 0, 127
      call STABAz(VSTAB, INTC,t7,v7,N)
DO v8 = 0, 127
   call INTnL(OUTPUT,t8,v8, VSTAB, N)
print 6, (OUTPUT(t8,i), i = 0, 127)
```

```
DO t8 = 0, N
   real VSTAB(0:7,0:127)
   real OUTPUT(0:0,0:127)
   DO t7 = 8*t8, 8*t8+7
      real BL(0:7,0:127)
      real INTC(0:0,0:127)
      DO t5 = 8*t7, 8*t7 + 7
         real Hydro(0:511,0:511)
         real Frequence(0:0,0:255,0:511)
         real Energie(0:0,0:199,0:127)
         DO t1 = 0, 511
            read 5, (Hydro(t1,h1), h1 = 0 , 511)
         DO h2 = 0, 511
            call FFTr(Frequence,0, h2, Hydro, N)
```

```
      DO v3 = 0, 127
         DO f3 = 0, 199
            real Voies(0:0,0:199,0:127)
            call FV(Voies, 0, f3, v3, Frequence, N)
            call MOD2(Energie(0,f3,v3), Voies(0,f3,v3))
      DO v5 = 0, 127
         call RtBL(BL, Energie,t5,v5,N)
   DO v6 = 0, 127
      call INTnL(INTC,0,v6, BL, N)
   DO v7 = 0, 127
      call STABAz(VSTAB, INTC,t7,v7,N)
DO v8 = 0, 127
   call INTnL(OUTPUT,t8,v8, VSTAB, N)
print 6, (OUTPUT(0,i), i = 0, 127)
```

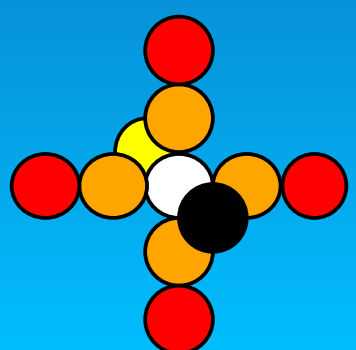

FORALL (p1;c1;l1)
CALL T1(X(*1),Y(*1))
ENDFORALL
dependence
FORALL (p2;c2;l2)
CALL T2(Y(*2),Z(*2))
ENDFORALL
Dependencies
Signal
periodicity
Memory
live data
Scheduling
maximal block size
size of data used by iteration blocks
computation time
communication time
Latency
Partitioning
distribution
Communication
processor number
applicative characteristics
programming model
processor number
forall i = 1, n
forall j = 1,m
a(i,j) = b(i,j)
endfor
endfor
Application
Target Machine

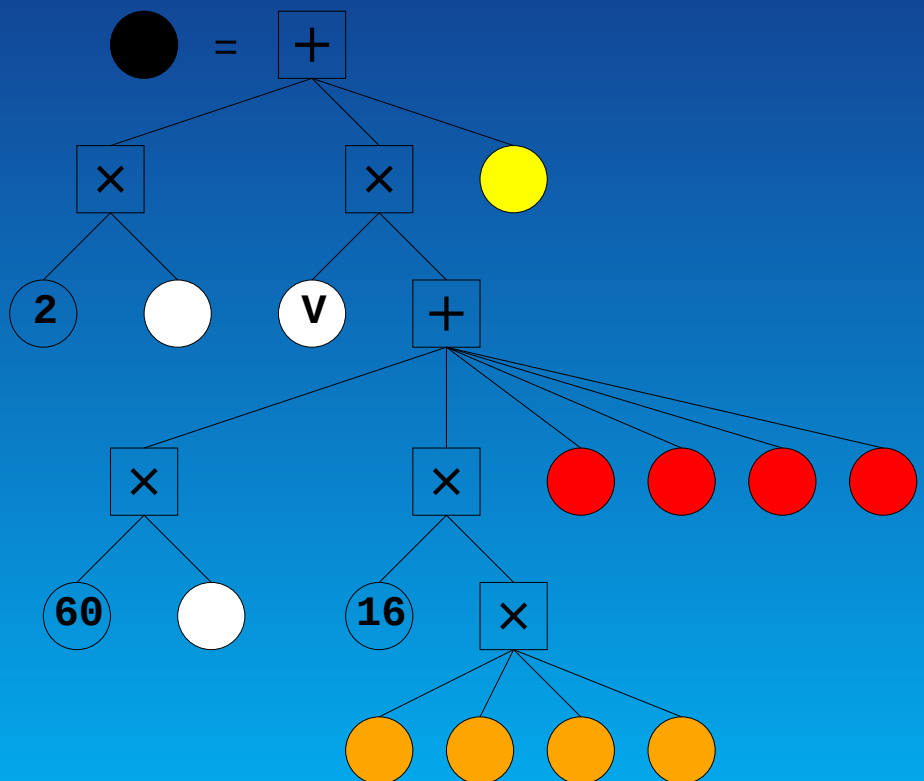
- Programmation concurrente par contraintes
- Traitement global du problème du placement (non-linéaire)
- Démonstration de faisabilité
- Études de modélisation (communications dans PSP RBE2)
- Extension en robustesse et applicabilité (thèse)
- Projet recherche exploratoire RNRT PROMPT
- plan d'étude amont DGA NARVAL action Paradis

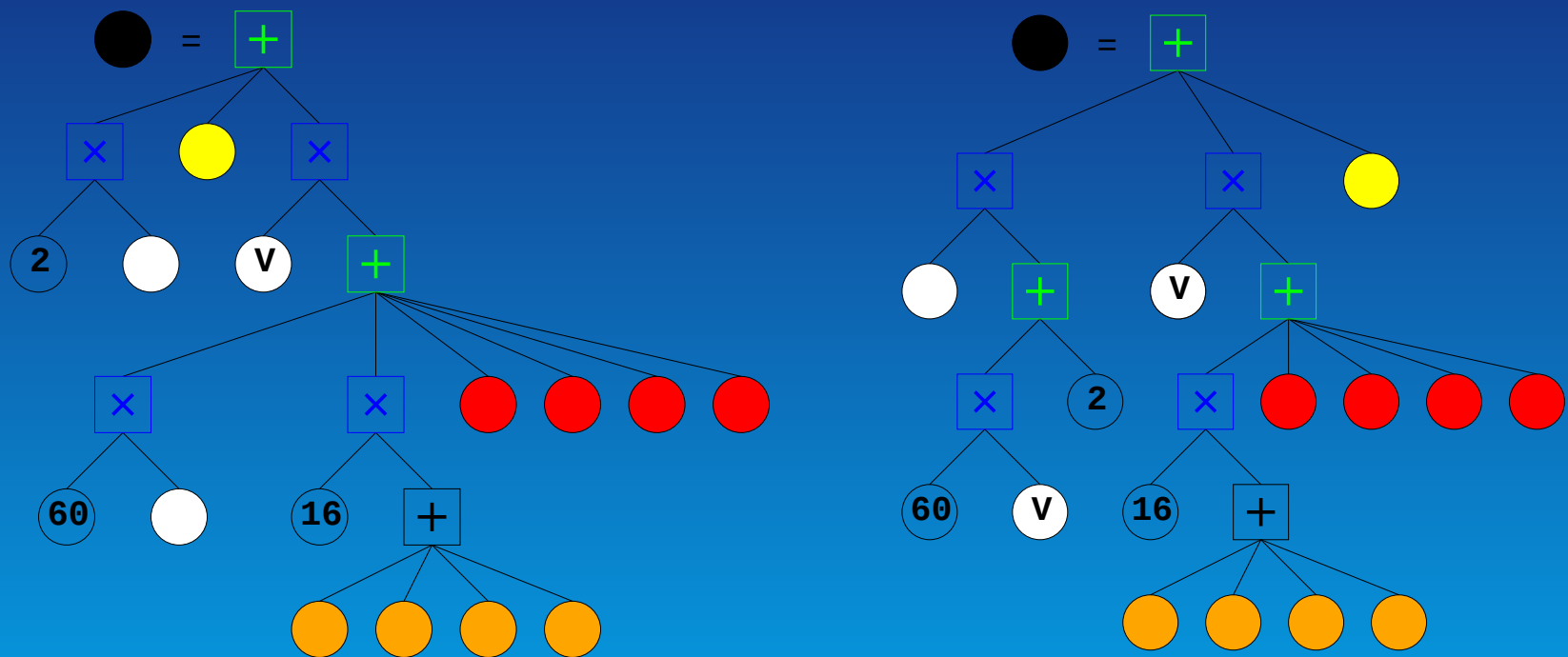
Optimisation de l'évaluation d'expressions

- Problème
  - Évaluation répétée de grosses expressions
  - Améliorer évaluation pour processeurs et DSP superscalaires & VLIW
- Approche
  - Explorer l'espace des expressions mathématiquement équivalentes
  - Étudier l'impact des transformations algébriques sur les performances
- Contraintes
  - Éviter l'explosion combinatoire
  - Prendre en compte ILP et nouvelles instructions

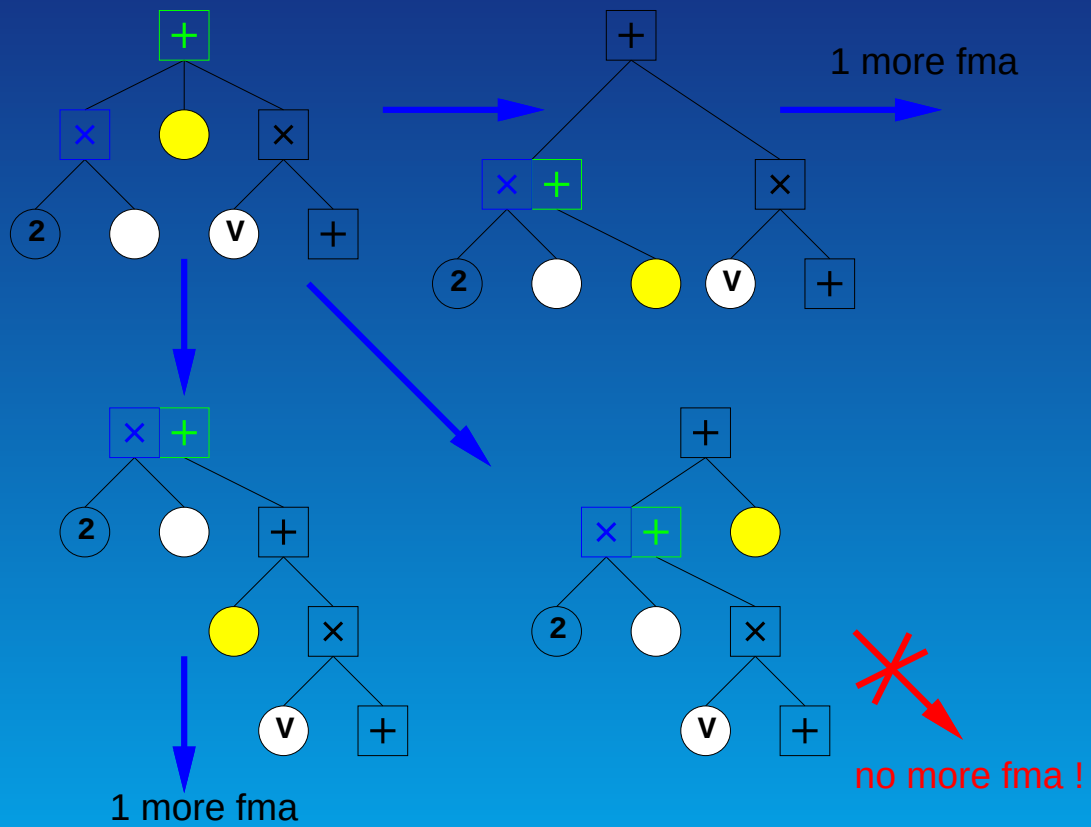
```
[...]
 DO J = 3,NP-2
   DO I = 3, NP-2
     U(I,J,KP) =
$       2 * U(I,J,KM) - U(I,J,KP)
$       - V(I,J) * ( 60*U(I,J,KM)
$       -16*( U(I+1,J,KM) + U(I-1,J,KM)
$       + U(I,J-1,KM) + U(I,J+1,KM))
$       + U(I+2,J,KM) + U(I-2,J,KM)
$       + U(I,J-2,KM) + U(I,J+2,KM))
   ENDDO
 ENDDO
[...]
```




=
+
×
×
2
V
+
×
×
60
16
×



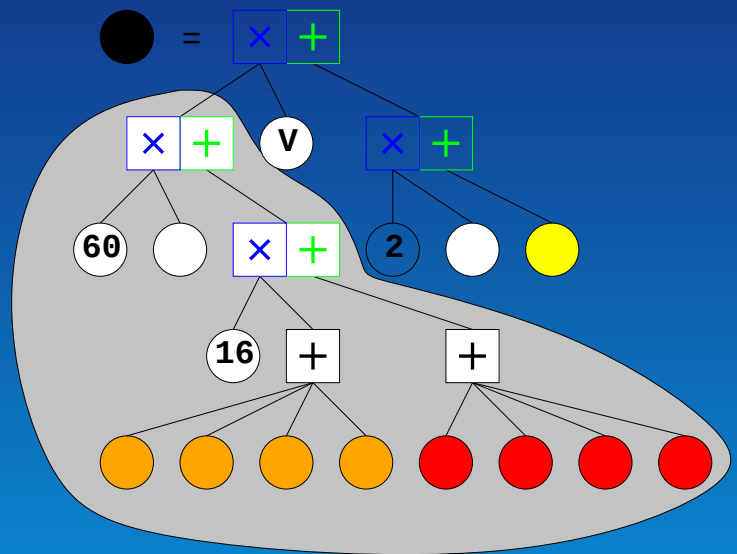
2 factorisations et 4 ×+



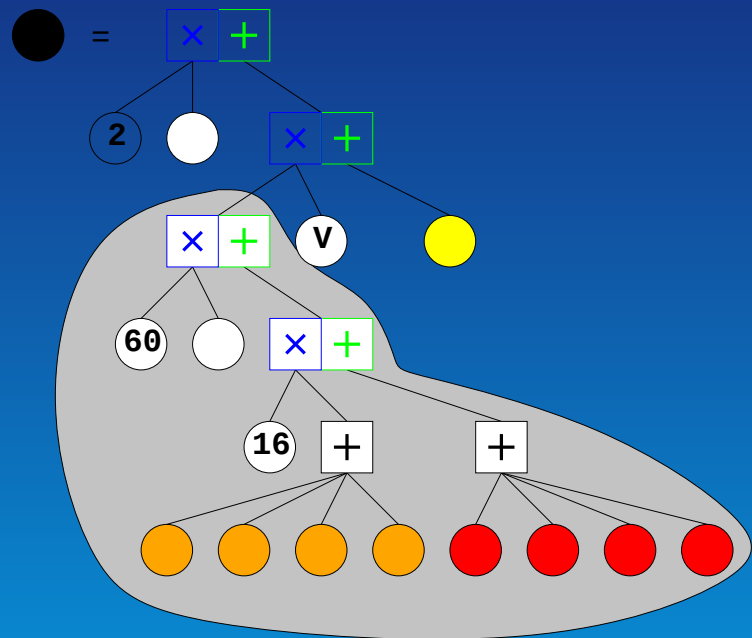
69300 expressions mathématiquement équivalentes

- Architecture
  - IBM RS6000 avec processeurs 120 & 160 MHz (P2SC 595)
  - 2 unités flottantes
    pipeline flottant à latence de 2
    } 4 opérations indépendantes
  - Instruction multiplication-addition (même latence que l'addition)
- Compilation
  - IBM XLFortran release 4.1 (-O3 -qarch=pwr2 -qhot)
  - Préprocesseur KAP pour Fortran IBM (V3.3)

ONDE24 - quelques résultats sur un P2SC 595 à 160MHz



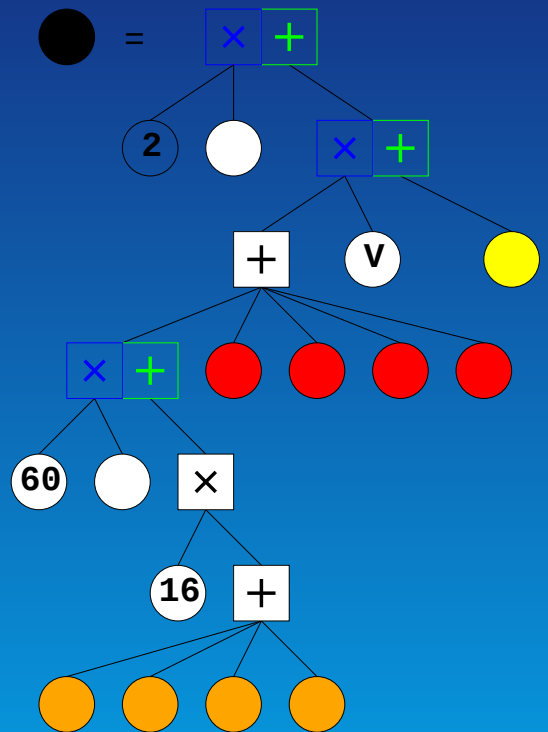
149 MFLOP/s



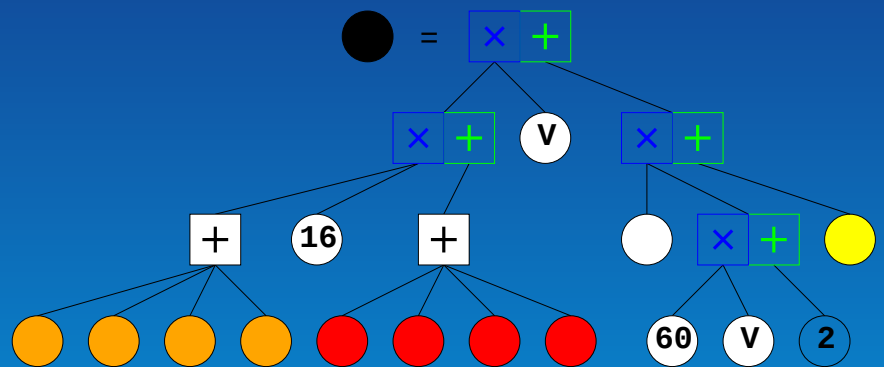
130 MFLOP/s

➡ 2 chemins critiques différents

ONDE24 - quelques résultats sur un P2SC 595 à 160MHz



102 MFLOP/s



162 MFLOP/s

➡ 60% d'augmentation des performances

- Notre approche

  Chercher dans l'espace des expressions algébriquement équivalentes

  Prendre une *bonne* expression finale

- Néanmoins

  L'espace des expressions est **trop large**

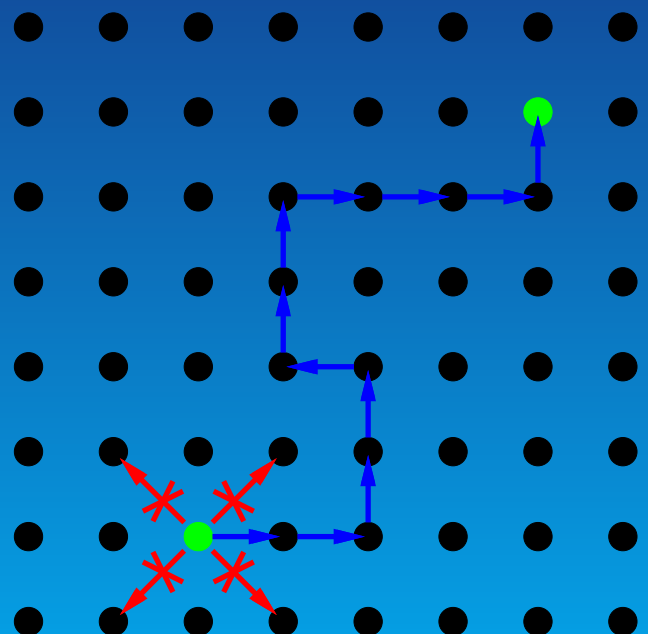
- D'où ... heuristique gloutonne

  Suivre un processus **itératif**

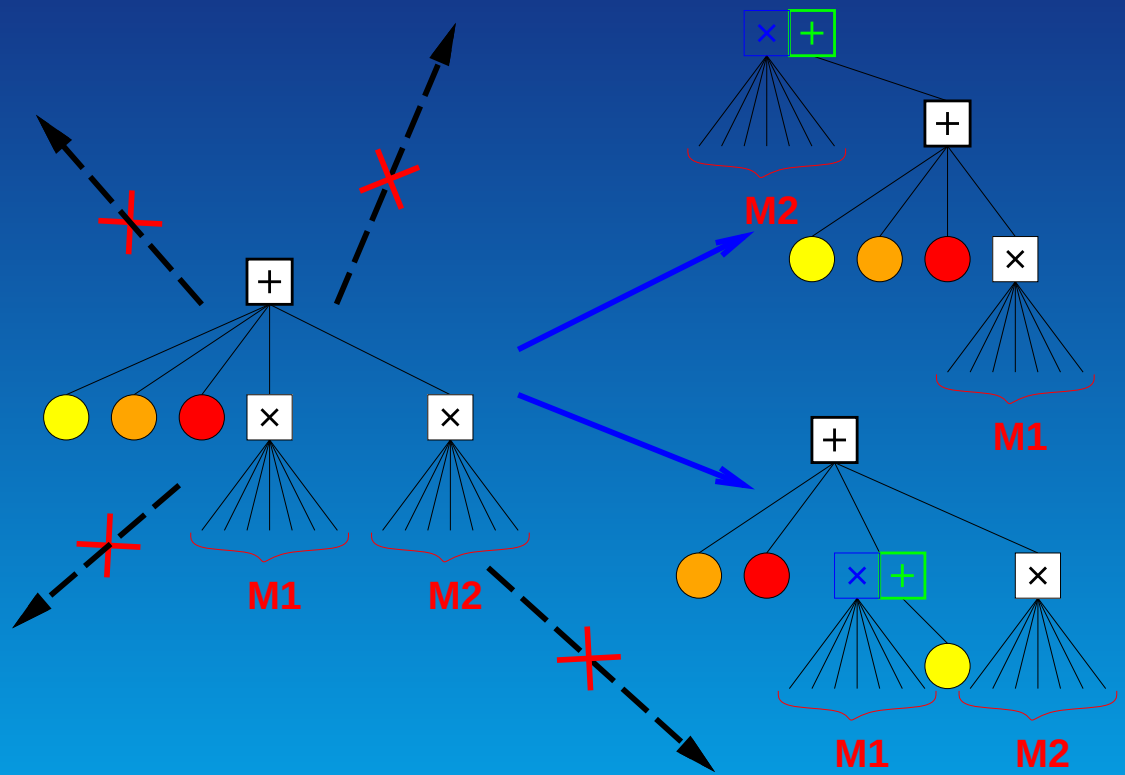
  Prendre des **décisions locales**

  Limiter l'espace des **transformations élémentaires**

  Trouver un **critère de choix** simple et efficace



2 **transformations élémentaires** intéressantes selectionnées



Critère de choix

➡ ~~Coût réel~~

➡ coût estimé

- $W = \sum_{i=1}^{n} w_i$
- $C = \max_{i=1}^{n} d_i$

1. Normalisation - *pattern matching*
   - Garantit que les expressions peuvent être librement réarrangées
   - Applique des simplifications algebriques simples
2. Factorisation - *heuristique gloutonne*
   - Réduit le nombre des opérations
   - Sélectionne l'arbre le mieux équilibré
   - Favorise la factorisation ultérieure de code invariant
3. Extraction de $\times+$ - *heuristique gloutonne*
   - Extrait autant de $\times+$ que possible
   - Sélectionne l'arbre le mieux équilibré

- Résultats expérimentaux prometteurs
  - Amélioration des performances jusqu'à un facteur 3
- Complètement imnplémenté
  - Utilise CAVEAT (CEA) et STORM (LIFO, Stony Brook)
  - Inclus dans PIPS
- Temps d'optimisation raisonnable dans un contexte de compilation
  - ONDE24 (500 lines) ➡ 4 sec (Sun Ultra1 143 MHz)

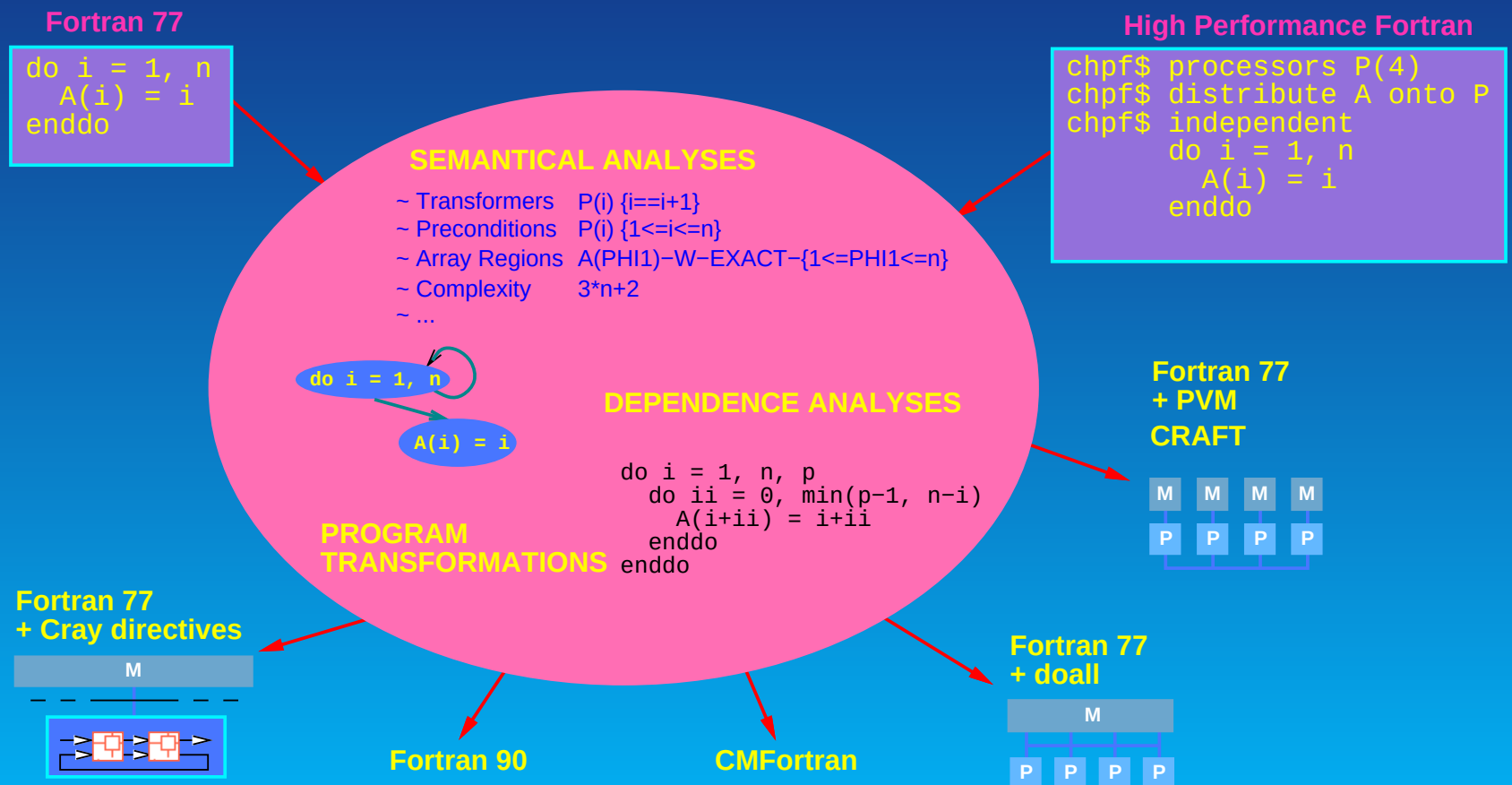
Pourquoi un établi ?

- Paralléliseurs et compilateurs pour
  - Superalculateurs & serveurs parallèles
  - Stations de travail superscalaires
  - Super DSP parallèles
  - ASP, FPGA, SoC, MP-SoC...
- Même usage d'analyses et transformations de programme avancées
  - Analyses semantiques interprocédurales
  - Optimisations de code
  - Vectorisation & parallelisation
  - Restructuration & ingénierie à rebour

Pourquoi un établi ?

- Factorise le tout dans un outil de développement commun pour réutiliser le code

PIPS — perspective de l'utilisateur



- PIPS $\equiv$ compilateur source à source (Fortran...) découpé en phases :
  - Analyses interprocédurales
  - Transformations
  - Générateurs de code
  - « *Pretty-printers* »
  - Infrastructure et ménage
- Usage intensif d'algèbre linéaire $\rightsquigarrow$ « bibliothèque C^3 » (matrices, vecteurs, systèmes de contraintes linéaires, forme normale de HERMITE,...)
- NewGen : gestionnaire de structures de données

- Effets d'instructions sur variables en lecture ou écriture/utilisées ou nécessaires
- Graphes de dépendance
- Préconditions — prédicats sur les variables entières des instructions
- Régions — sections de tableaux polyédriques. Étend les effects
- Graphe de flot de données, planification, base de temps et répartition
- Détection de réductions
- Complexités symboliques
- Fortran `lint`

- Ajout d’autres domaines d’interprétation abstraite : projet APRON

- Boucles : distribution, déroulage, strip-mining, échange, normalisation,...
- Évaluation partielle
- Élimination de code mort, élimination use-def, déspaghettification
- Privatisation tableaux & scalaires
- Atomiseurs
- Clonage
- Nettoyage des déclarations
- Optimiseur d’expressions EOLE
- Restructuration avec STF/ToolPack (emballeur en C)

- Parallélisation et vectorisation
  - Fortran 77 + directives OMP
  - Fortran 77 + directives HPF
  - Fortran 77 + `DOALL`
  - Expressions de tableau Fortran 90
  - Directives Fortran Cray
- Compilateur HPF (redistributions, I/O), bibliothèque PVM/MPI
- « méthode polyédrique : CM-Fortran, CRAFT
- « WP65 » : mémoire virtuelle paratagée compilée avec bibliothèque PVM

Utile pour :

- Spécialisation de code (e.g. après clonage ou inlining)
- Rétro-ingéniérie de vieux codes stratifiés/fossilisés
- Nettoyage après transformations source à source
- Simplification avant application d'autres analyses

Combinée avec profit avec
évaluation partielle, élimination use-def, restructuration graphe de contrôle

- Exemple avec `OCEAN`
  25 des 57 tests éliminés dans la procédure principale
  (8 true branches and 17 false branches)
- Élimination de code mort & autres analyses concernées
- Gestionnaire de consistance interprocédural PIPSmake
- Gestionnaire d'objets NewGen and PIPSdbm
- Interfaces
- Environnement

⇝ PIPS workbench

- Simulation de fluide 2D avec équations de BOUSSINESQ
- Beaucoup d'options (avec ou sans la température,...)
- Initialisations dans la sous-routine `START`

⇝ nécessite une information *interprocédurale* pour simplifier le code

Predicats sur variables entières scalaires

Epips

Buffers Files Tools Edit Search Epips Help

```
C  P() {NFTVMT==0, NUM1==1, NUSHUF==0, NXACAC==0, NXLOG2==0,
C    NXPRNT==0, NXSCSC==0, NXSHUF==0, NXXCSR==0, NXXOUT==0,
C    NXXRCS==0, NXXXIN==0, TEMPHY:NCALL==0, ZETAPH:NCALL==0}
      CALL START(NX, NY)                                          0006
C  P(KDATA,KPF,KPP,KPRESS,KSM,KTA,KTAPE,KTF,KTP,KUF,KUP,KVF,KVP,KZF,
C    KZP,LMAPP,N1,N2,NOALS,NOTEMP,NUMBER) {KDATA==1, KPF==1,
C    KPP==249, KPRESS==1, KSM==1, KTA==50, KTAPE==0, KTF==1,
C    KTP==50, KUF==1, KUP==249, KVF==1, KVP==249, KZF==1, KZP==249,
C    LMAPP==1, N1==128, N2==128, NFTVMT==0, NOALS==1, NOTEMP==0,
C    NUM1==1, NUMBER==250, NUSHUF==0, NXACAC==0, NXLOG2==0,
C    NXPRNT==0, NXSCSC==0, NXSHUF==0, NXXCSR==0, NXXOUT==0,
C    NXXRCS==0, NXXXIN==0, TEMPHY:NCALL==0, ZETAPH:NCALL==0}
C
C IF  LMAPP = 1 , H  (THE SIZE OF THE BOX IN Y DIRECTION)  SHOULD BE
C SPECIFIED IN THE MAIN PROGRAM.
C
      H = PI                                                      0007
C  P(KDATA,KPF,KPP,KPRESS,KSM,KTA,KTAPE,KTF,KTP,KUF,KUP,KVF,KVP,KZF,
-----Emacs: EPips-0        13:35 Mail   (Fortran EPips)--L229--C3-- 2%----------
```

Utilise préconditions calculées pour enlever les instructions jamais exécutées

- Instructions avec préconditions fausses
- Branches de test toujours vraies ou toujours fausses
  Calcule la condition de toute manière si effet de bord *!*
- Boucle exécutée 0 fois
- Boucle exécutée 1 fois

⇝ Élimination itérative use-def, restructuration graphe du flot de contrôle et évaluation partielle

Pour appliquer une transformation ou avoir une analyse :

¿ Quelles analyses sont nécessaires pour quel module ?

¡ Il suffit de demander !

Calcul à la demande de style `make`

`suppress_dead_code` > `MODULE`.`code`

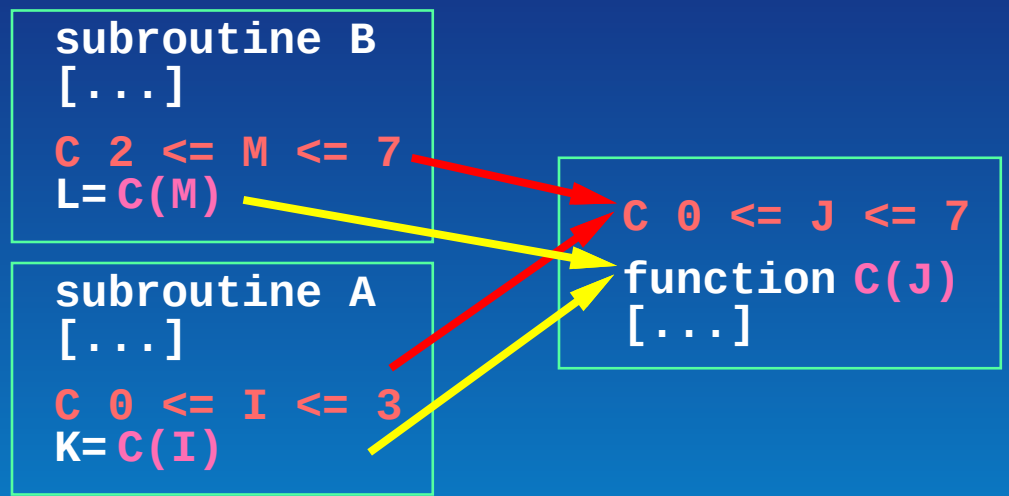
< `MODULE`.`code`

< `MODULE`.`proper_effects`

< `MODULE`.`preconditions`

⇝ étendu pour traiter les problèmes interprocéduraux

Interprocéduralité
avec CALLERS
et CALLEES



preconditions_inter > MODULE . preconditions
< MODULE . code
< MODULE . transformers
< CALLERS . preconditions

Gère ressources PIPS... Tout est ressource

- Charge automatiquement en mémoire les ressources fichiers si nécessaire
- Indépendent de l'architecture et du placement mémoire
- Lit & écrit des objets NewGen
- Objets non-NewGen avec des méthodes utilisateur
- Interface de base de toutes les phases PIPS
  - Persistance ⇝ experiences multi-run
  - Points de préemption aux limites de phase
  - Phases externes possibles

Éviter gestion explicite de structures de données complexes :

```
bool suppress_dead_code(string module)
{
  /* get resources from PIPS DBM */
  statement s = db_get_resource(DBR_CODE, module);
  [...]

  gen_recurse(s,                /* start recursion from */
    statement_domain,           /* domain to visit */
    dead_statement_filter,      /* top-down decision */
    dead_statement_rewrite)     /* bottom-up transformation */

  [...] /* update resources to PIPS DBM */
}
```

Outil NewGen

Génération automatique depuis fichiers LaTeX

Representation interne hiérarchique de PIPS :

```
statement = label:entity x number:int x ordering:int
          x comments:string x instruction ;
instruction = sequence + test + loop + goto:statement
          + call + unstructured ;
sequence = statements:statement* ;
```

```
static bool dead_statement_filter(statement s)
{
  instruction i = statement_instruction(s);

  if (!statement_weakly_feasible_p(s))
    /* empty precondition */
    remove_dead_statement(s, i);
  else {
    switch (instruction_tag(i)) {
    case is_instruction_loop:
      loop l = instruction_loop(i);
      ...
    case is_instruction_test:
      test t = instruction_test(i);
      ...
```

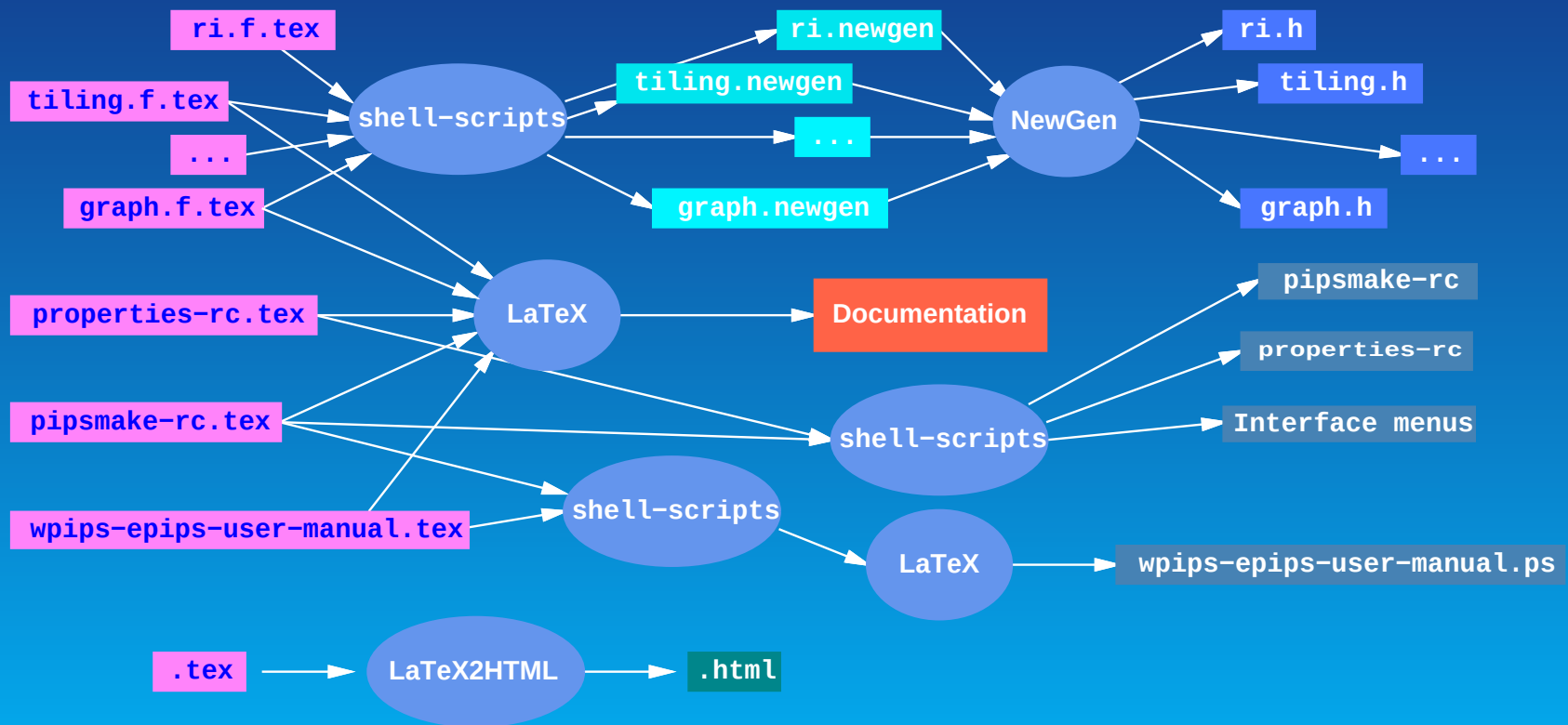
- Construction de types complexes à partir de listes, tables associatives, unions, produits, ensembles,... à la IDL
- Accesseurs, constructeurs, destructeurs, lecteurs, écrivains...
- Vérificateur de consistance
- Itérateurs génériques optimisés
- méthodes multi-languages (C & Common-Lisp pour l'instant)
- Support pour la persistance (PIPSdbm)
- Construction possible de passes indépendantes (sans PIPSmake) : EOLE

PIPS est basé sur l'algèbre linéaire :

- Vecteurs, matrices (e.g. forme normale de Hermite)
- Constraintes linéaires, systèmes (e.g. faisabilité via Fourier-Motzkin)
- Systèmes générateurs, enveloppe convexe (Chernikova)
- simplex, PIP

```
bool statement_weakly_feasible_p(statement s)
{
  /* load the precondition system sc associated to s */
  transformer p = load_statement_precondition(s);
  Psysteme sc = predicate_system(transformer_relation(p));
  /* return whether there may be a solution */
  return !sc_empty_p(sc);
}
```

La consistance des sources de PIPS est conservée automatiquement :



Chaque fichier est sous SubVersioN (SVN) :
versions de développement (branches), production (tronc) et

distribution (tag, release)

```
    \subsubsection{Dead Code Elimination}
    %%@UserManualDocumentation: suppress_dead_code
    Function \verb+suppress_dead_code+ is used to delete non-executed code,
    ...
    %%ÿUserManualDocumentation
\begin{verbatim}
alias suppress_dead_code 'Dead Code Elimination'
suppress_dead_code                > MODULE . code
        < PROGRAM . entities
        < MODULE . code
        < MODULE . proper_effects
        < MODULE . cumulated_effects
        < MODULE . preconditions
\end{verbatim}
\begin{PipsMenu}{Transformations}
...
   suppress_dead_code
...
\end{PipsMenu}
```

Pas moins de 5 interfaces suite à la longue vie de PIPS...

Shell : tests rapides, debug et tests de non-régression

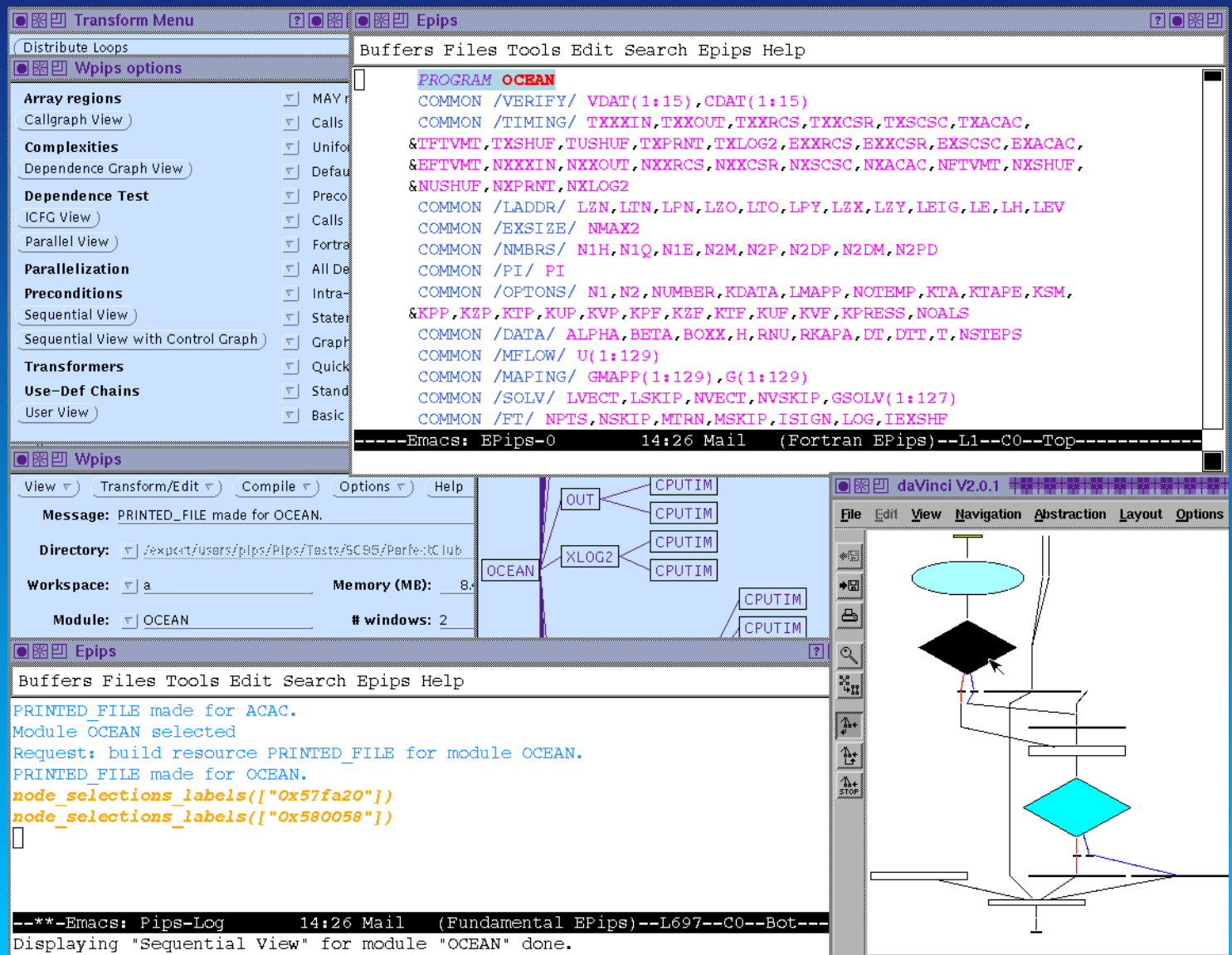
TTY : expérimentations, debug et tests de non-régression

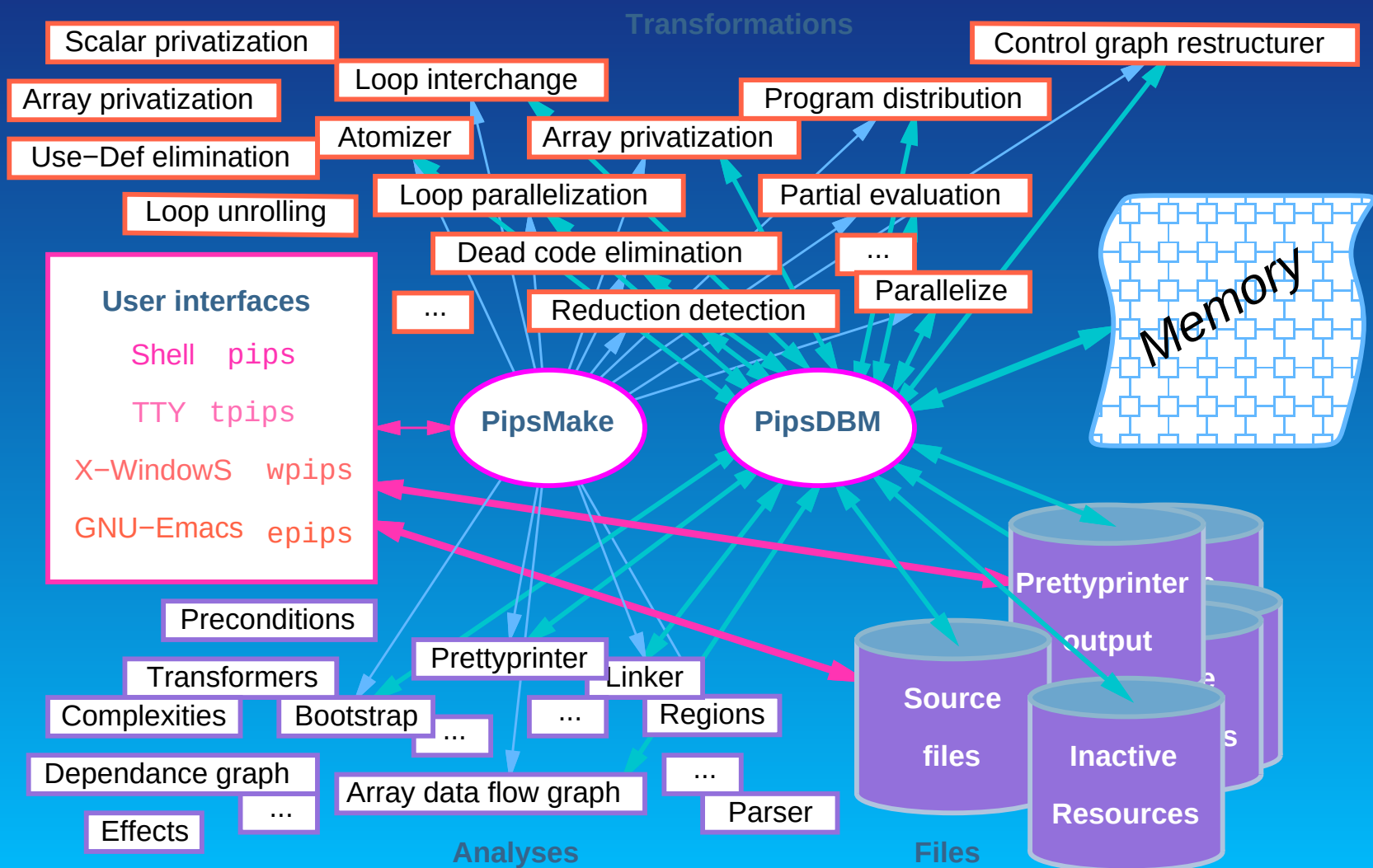
X11/XView : Interface avec menus

Java Swing : Interface avec menus à la mode et portable

GNU-Emacs : La totale avec hypertexte, graphes avec *daVinci*,...

`logfile_to_tpips` pour tests de non-régression


Transform Menu
Distribute Loops
Wpips options
Array regions
Callgraph View
Complexities
Dependence Graph View
Dependence Test
ICFG View
Parallel View
Parallelization
Preconditions
Sequential View
Sequential View with Control Graph
Transformers
Use-Def Chains
User View
Epips
Buffers Files Tools Edit Search Epips Help
PROGRAM OCEAN
COMMON /VERIFY/ VDAT(1:15),CDAT(1:15)
COMMON /TIMING/ TXXXIN,TXXOUT,TXXRCS,TXXCSR,TXSCSC,TXACAC,
&TFTVMT,TXSHUF,TUSHUF,TXPRNT,TXLOG2,EXXRCS,EXXCSR,EXSCSC,EXACAC,
&EFTVMT,NXXXIN,NXXOUT,NXXRCS,NXXCSR,NXSCSC,NXACAC,NFTVMT,NXSHUF,
&NUSHUF,NXPRNT,NXLOG2
COMMON /LADDR/ LZN,LTN,LPN,LZO,LTO,LPY,LZX,LZY,LEIG,LE,LH,LEV
COMMON /EXSIZE/ NMAX2
COMMON /NMBRS/ N1H,N1Q,N1E,N2M,N2P,N2DP,N2DM,N2PD
COMMON /PI/ PI
COMMON /OPTONS/ N1,N2,NUMBER,KDATA,LMAPP,NOTEMP,KTA,KTAPE,KSM,
&KPP,KZP,KTP,KUP,KVP,KPF,KZF,KTF,KUF,KVF,KPRESS,NOALS
COMMON /DATA/ ALPHA,BETA,BOXX,H,RNU,RKAPA,DT,DTT,T,NSTEPS
COMMON /MFLOW/ U(1:129)
COMMON /MAPING/ GMAPP(1:129),G(1:129)
COMMON /SOLV/ LVECT,LSKIP,NVECT,NVSKIP,GSOLV(1:127)
COMMON /FT/ NPTS,NSKIP,MTRN,MSKIP,ISIGN,LOG,IEXSHF
-----Emacs: EPips-0 14:26 Mail (Fortran EPips)--L1--C0--Top----------
Wpips
View
Transform/Edit
Compile
Options
Help
Message: PRINTED_FILE made for OCEAN.
Directory:
Workspace: a
Memory (MB):
Module: OCEAN
windows: 2
OCEAN
OUT
XLOG2
CPUTIM
Epips
Buffers Files Tools Edit Search Epips Help
PRINTED_FILE made for ACAC.
Module OCEAN selected
Request: build resource PRINTED_FILE for module OCEAN.
PRINTED_FILE made for OCEAN.
node_selections_labels(["0x57fa20"])
node_selections_labels(["0x580058"])
--**-Emacs: Pips-Log 14:26 Mail (Fundamental EPips)--L697--C0--Bot---
Displaying "Sequential View" for module "OCEAN" done.
daVinci V2.0.1
File
Edit
View
Navigation
Abstraction
Layout
Options

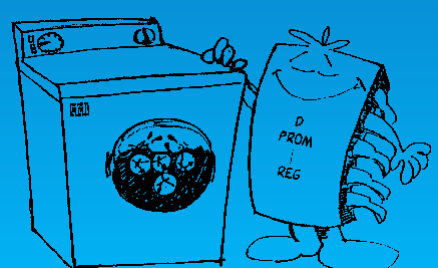

Transformations
Scalar privatization
Array privatization
Use−Def elimination
Loop unrolling
Loop interchange
Atomizer
Loop parallelization
Dead code elimination
...
Reduction detection
Array privatization
Program distribution
Partial evaluation
...
Parallelize
Control graph restructurer
Memory
User interfaces
Shell pips
TTY tpips
X−WindowS wpips
GNU−Emacs epips
PipsMake
PipsDBM
Preconditions
Transformers
Complexities
Bootstrap
...
Dependance graph
...
Effects
Prettyprinter
Linker
...
Regions
Array data flow graph
...
Parser
Analyses
Prettyprinter output
Source files
Inactive Resources
Files

La petite *cuisine* :

Cassolette de PIPS aux phases

- Créer des objets NewGen si nécessaire
- Écrire votre code (!) pour traiter un module
- Compiler et tester avec Makefile inspiré d'autres bibliothèques/phases
- Déclarer la nouvelle phase dans `pipsmake-rc.tex`

  ⇝ Nouvelle analyse/transformation interprocédurale avec interface graphique



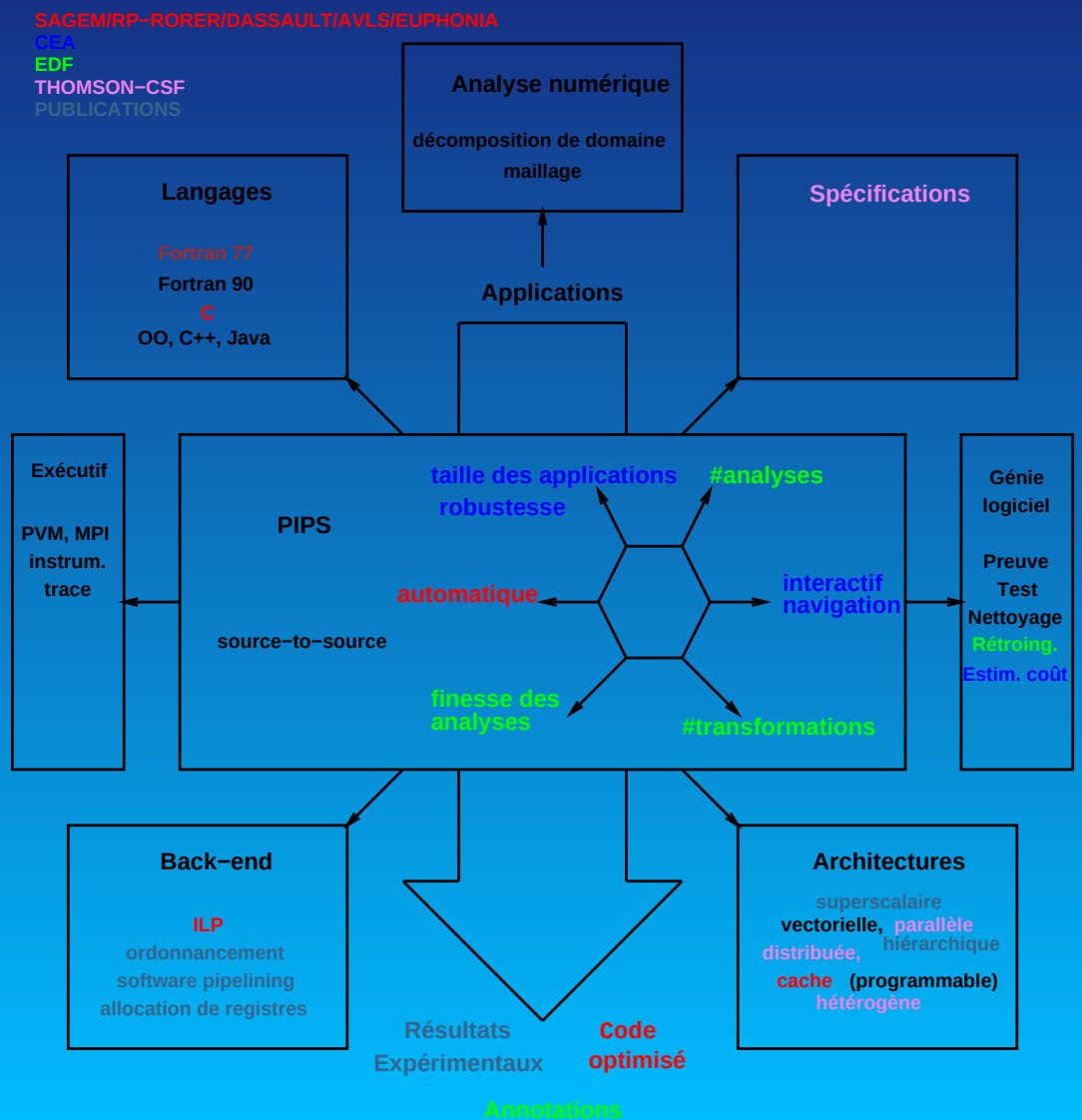
Autres établis de compilation :

- SUIF (Monica Lam, Stanford)
  - Plutôt orienté C, génération de code de + bas niveau
  - Pas d'interprocéduralité dans la version distribuée
  - Ne gère par l'`EQUIVALENCE` Fortran
  - Pas d'équivalent de PIPSmake pour garantir automatiquement la cohérence et gérer l'interprocéduralité
  - Pas de gestionnaire d'objets à la NewGen ⇝ écrire ses propres classes C++

- PARAFRASE 2 (Constantin Polychronopoulos, UIUC)
  - Pas d'équivalent de PIPSmake (- interprocéduralité)
  - Pas de NewGen (C++ brut de fonderie)
- POLARIS (David Padua, UIUC)
  - + vérificateur de sémantique Fortran
  - Pas d'équivalent de PIPSmake (- interprocéduralité)
  - Pas de NewGen (C++)

- Crayettes, MPP (réseaux de Transputers), réseau neuronal, algorithme génétique,...
- Parallélisation interprocédurale : 1982–1996/97–????
- Tiling : 1985–????–1997/98 (HP puis SGI et MIPS)
- Synthèse de code et de communication : 1987–1991–????
- Exploitation de l'investissement fait dans PIPS *robustesse de la bibliothèque linéaire* Évolution avec APRON

Évolutions du workbench PIPS



- Connexité (automne 1993)
- Simulog (septembre 97 et mars 1998)
  - Réécriture
  - Analyses trop compliquées (Dassault) ou pas encore assez fines (EDF)
  - Tentative de coopération dans le cadre post-Ariane 4
- CEA-DAM : propriété des sources ou co-développement
  Intégration de la méthode polyédrale dans PIPS, thèses d'A. Platonoff et d'A. Leservot
- EDF : co-développement si sources dans le domaine public
  Phase de calcul d'alias, CIFRE ENS-Cachan/EDF

- Évolution des thèmes de recherche largement au-delà du cadre *parallélisation automatique*
  ⇝ *optimisations en coût et en temps de réponse, génie logiciel,...*
- Sans perdre de vue les objectifs économiques ni sacrifier la qualité de la recherche
- Les expériences ne sont pas oubliées
  théorie et pratique
- Les contacts industriels non plus
  vitesse contre précision, implémentation contre papiers
- Et donc le financement contractuel

- Importance du *source-à-source* (vs projet SUIF de Stanford)
- Réduction des coûts en rétro-ingénierie
  ⇝ *travail supplémentaire dans PIPS...*
- Réduction des coûts de maintenance
  pour du code optimisé
- Réduction du coût de développement
  par la compilation de spécifications

- PIPS = interprocéduralité + sémantique + programmation linéaire
- Aborde des programmes réels
- Beaucoup d'autres choses (encore)
- PIPS $\equiv$ établi + boîte à outils, environnement ouvert
- Environnement de génie logiciel ouvert (NewGen, PIPSmake, PIPSdbm,...)
- $\rightsquigarrow$ rajout simplifié de nouvelles phases
- Bonne plateforme pour étudier l'interprocéduralité
- Disponible gratuitement

`http ://www.cri.ensmp.fr/pips`

- Durcir le parser C (pour le traitement du signal...)
- Java pour le calcul scientifique (tableaux)... part de très bas en performances
  - Rajout d'un parser Java
  - Élimination des vérifications d'accès aux tableau
  - Spécialisation de code
  - Élimination déréférencements méthodes
  - Couplage JVM avec un JIT : optimisation code 3 adresses, détection et optimisation de boucle

List of Slides

Table des matières