Test de débordement de tableau

Thi Viet Nga NGUYEN
CRI - ENSMP
École des Mines de Paris

Séminaire INRIA/A3-CRI-LRI-VERSAILLES
29 Janvier 2001

Problèmes

Accès aux éléments de tableaux en dehors des dimensions déclarées

- Erreur fréquente de programmation
- Fautes de segmentation
- Résultats non fiables
- Trous de sécurité
- Comportements imprévisibles du programme

Problèmes (2)

- Tests systématiques (ex. Java)
- Avec tests de débordement
  - Temps d'exécution : **3** fois plus long
  - Temps de compilation : **3** fois plus long
  - Taille du code généré : **2** fois plus grande
- Développement avec tests (l'option -C , ex. Fortran, C, Pascal), Production sans test : non fiable

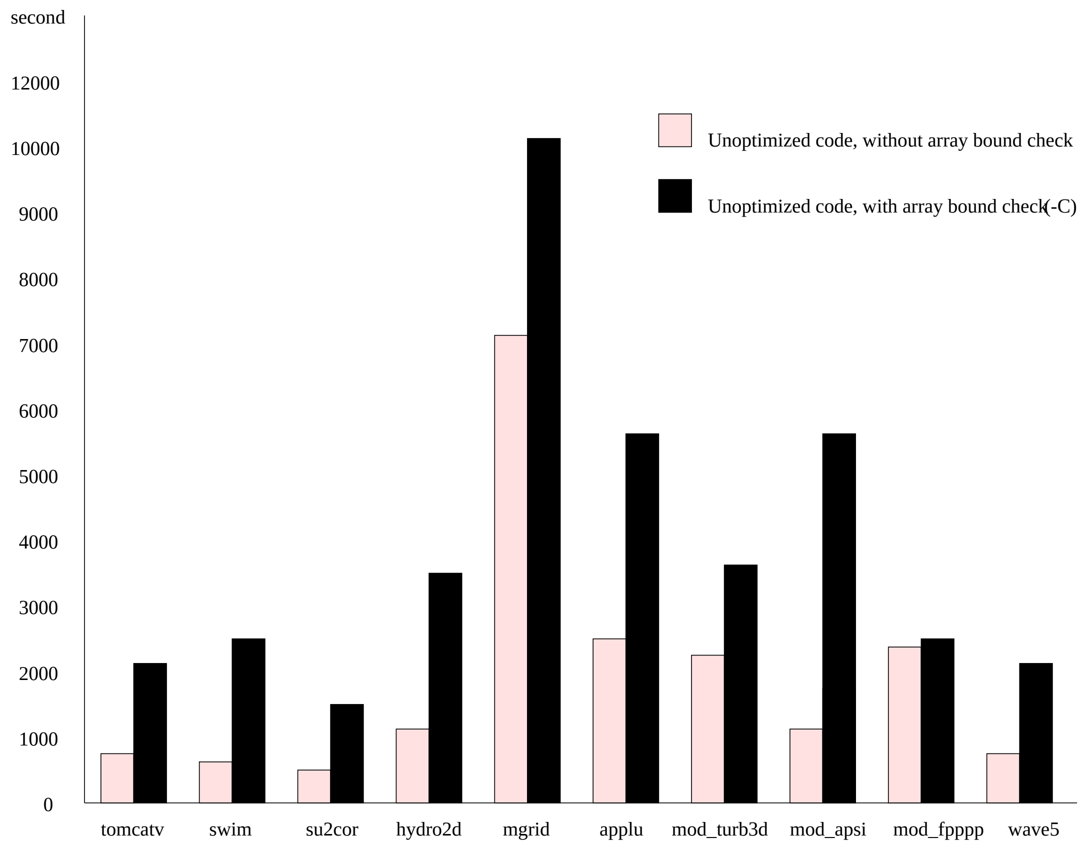


Figure 1: Execution times with and without array bound checking (SUN F77 5.0, Ultra SPARC 360MHz, Solaris 7) - Unoptimized code for development

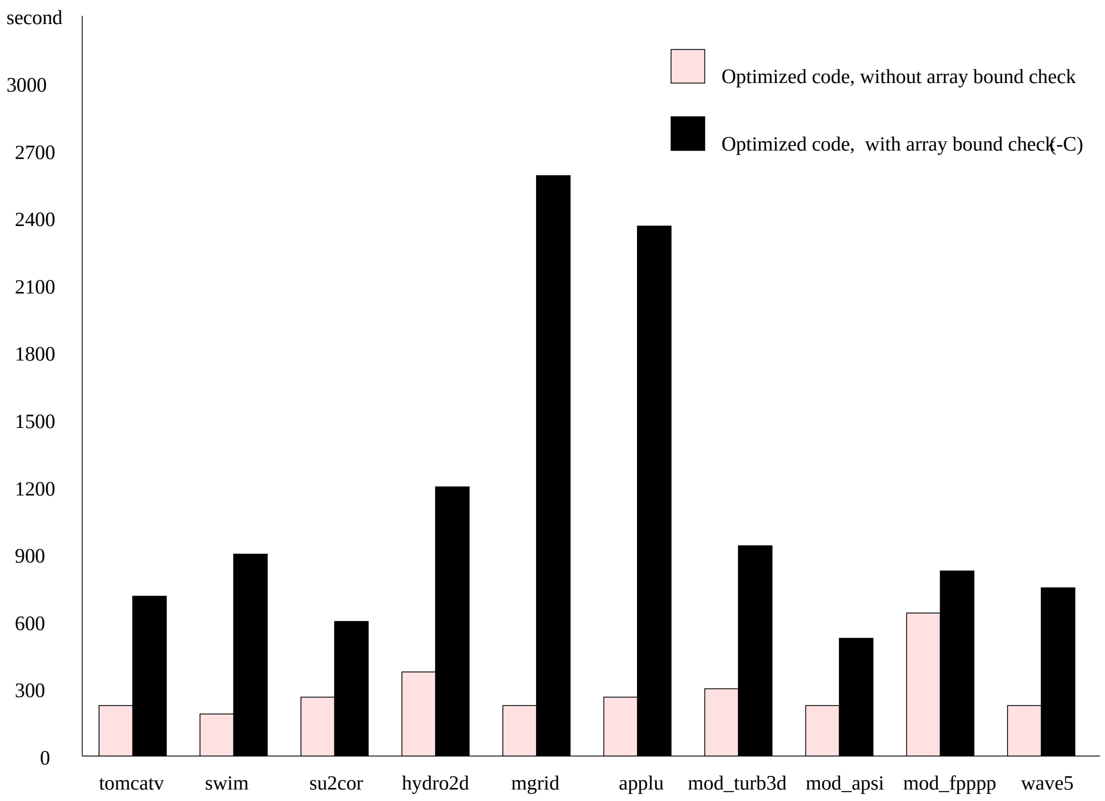


Figure 2: Execution times with and without array bound checking (SUN F77 5.0, Ultra SPARC 360MHz, Solaris 7) - Optimized code for production (f77 -fast -xarch=v8plusa -fsimple=2 -xprefetch)

	File size		Compile time	
Program	**Wo-C**	**W-C**	**Wo-C**	**W-C**
tomcatv	100336	128744	0:03	0:13
swim	112004	174792	0:04	0:20
su2cor	221752	536656	0:33	2:23
hydro2d	229076	386840	0:32	1:03
mgrid	114852	225324	0:08	0:37
applu	212912	999092	0:57	11:12
modturb3d	149364	260712	0:17	0:44
modapsi	361508	791028	1:16	3:15
modfpppp	201640	255820	0:54	1:22
wave5	552984	1459740	2:04	6:20

Table 1: Executable file size (in bytes) and compilation time (in minutes and seconds) with and without array bound checking (SUN F77 5.0, Ultra SPARC 360MHz, Solaris 7) - Optimized code for production (f77 -fast -xarch=v8plusa -fsimple=2 -xprefetch)

Motivations

- Optimisation des tests
- Coût raisonnable de la compilation
- Réutilisation des techniques automatiques d'optimisation et de parallélisation déjà implantées dans les compilateurs commerciaux
- Vérifications intra- et inter- procédurales

Plan

- Problèmes
- Motivations
- Travaux antérieurs
- Élimination des tests redondants
- Insertion des tests inévitables
- Résultats expérimentaux
- Conclusion

Travaux antérieurs

Suzuki, Ishihata (77) :

- assertions avant les accès aux tableaux
- vérification de l'absence de débordement par un démonstrateur de théorèmes

Cousot (78) :

- vérification automatique par interprétation abstraite
- preuve de l'absence de débordement en utilisant l'analyse statique de flot de données

- Coût élevé de la compilation
- Pas de déplacement de test pour créer des tests redondants
- Tests redondants partiels à évaluer à l'exécution

Travaux antérieurs (2)

Markstein, Coke (82), Asuru (92), Gupta (92,93), Kolte, Wolfe (95) :

- Déplacement des tests de débordement par analyse de flot de données
- Élimination des tests redondants
- Techniques : déplacement de code, élimination des redondances partielles, test disponible, test occupé et test anticipable,...

- Peu d'expérimentations
- Kolte, Wolfe (95) : bons résultats mais pas de comparaison de temps d'exécution

Travaux antérieurs (3)

Midkiff, Moreira, Snir, Gupta (98,99) :

- Réplication de code
- Partitionnement de boucles en multiples régions avec ou sans tests

- Pour Java seulement
- Problème de la taille du code

Travaux antérieurs (4)

Deux approches :

1. Mettre des tests de débordement partout et puis éliminer les tests redondants

2. Mettre des tests de débordement seulement là où on ne peut pas prouver qu'ils sont inutiles.

$\longrightarrow$ Implantation de chaque méthode à des fins de comparaison

Élimination des tests redondants

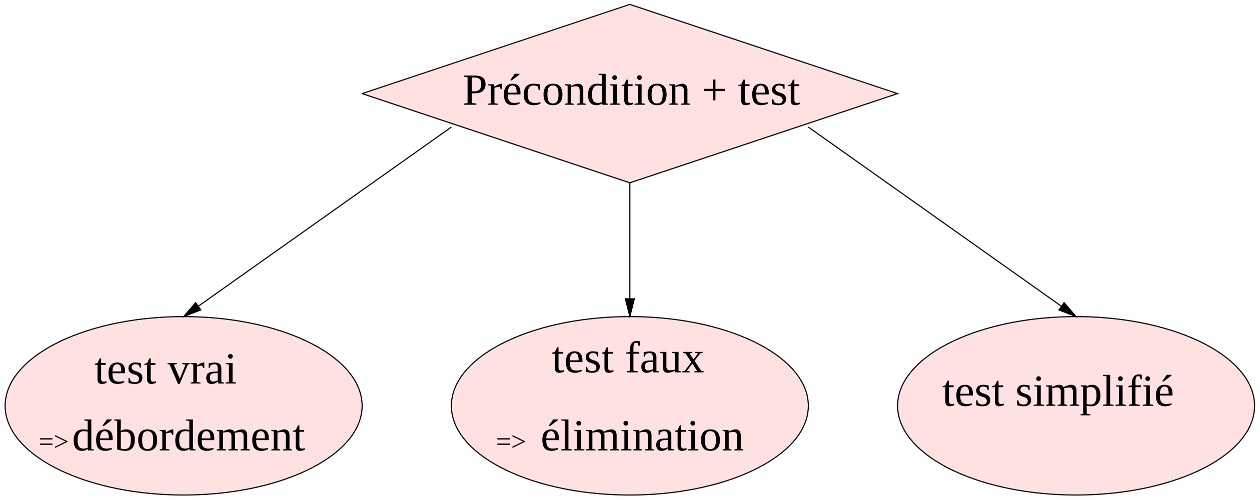
1. Génération de deux tests de débordement pour chaque dimension de chaque référence à un tableau
2. Élimination des tests :
   - triviaux : $\mathrm{N} - 1 \leq \mathrm{N}$
   - redondants par rapport à ceux de l'instruction courante
3. Élimination des redondances partielles

Élimination des tests redondants (2)

- **Transformer** : représente les effets d'une instruction sur les variables (scalaires et entières) du programme en donnant la relation entre leurs valeurs avant et après l'exécution de cette instruction
- **Précondition** : est un prédicat affine sur les variables (scalaires et entières). Il est vrai avant l'exécution de l'instruction correspondante
- Précondition $\xrightarrow{transformer}$ postcondition

Élimination des tests redondants (3)

- p_1, p_2 : deux points dans le graphe de contrôle ; $p_1 \longrightarrow p_2$
- test e est déjà évalué à p_1 et va être evalué à p_2
- postcondition(p_1) $\longrightarrow$ précondition(p_2) $\longrightarrow$ la deuxième évaluation est-elle redondante ?



```
   PROGRAM ELIMINATION
   REAL U(15), V(20)
   DO 10 I =1,N,2
     V(I+1) = U(I+1) + U(I)
10 CONTINUE
   END
```

1) *Code initial*

```
   PROGRAM ELIMINATION
   REAL U(15), V(20)
   DO 10 I =1,N,2
     IF (I+1.LT.1.OR.I+1.GT.15) STOP
     IF (I.LT.1.OR.I.GT.15) STOP
     IF (I+1.LT.1.OR.I+1.GT.20) STOP
     V(I+1) = U(I+1) + U(I)
10 CONTINUE
   END
```

2) *Après la génération des tests*

```
   PROGRAM ELIMINATION
   REAL U(15), V(20)
C  P() {}
   DO 10 I =1,N,2
C  P(I) {1≤I, I≤N}
     IF (I+1.LT.1.OR.I+1.GT.15) STOP
C  P(I) {1≤I, I≤14, I≤N}
     IF (I.LT.1.OR.I.GT.15) STOP
C  P(I) {1≤I, I≤14, I≤N}
     IF (I+1.LT.1.OR.I+1.GT.20) STOP
C  P(I) {1≤I, I≤14, I≤N}
     V(I+1) = U(I+1) + U(I)
C  P(I) {1≤I, I≤14, I≤N}
10 CONTINUE
   END
```

3) *Code avec tests et préconditions*

```
   PROGRAM ELIMINATION
   REAL U(15), V(20)
   DO 10 I =1,N,2
     IF (I.GT.14) STOP
     V(I+1) = U(I+1) + U(I)
10 CONTINUE
   END
```

4)*Après l'élimination des redondances partielles*

Program	Static checks	Dynamic checks
tomcatv	100.00%	100.00%
swim	98.45%	99.99%
su2cor	96.59%	97.54%
hydro2d	96.13%	94.14%
mgrid	94.92%	99.60%
applu	99.62%	97.09%
turb3d	93.09%	99.99%
modturb3d	97.62%	59.85%
apsi	95.87%	99.99%
modapsi	98.02%	99.90%
fpppp	92.81%	99.99%
modfpppp	94.61%	97.02%
wave5	94.66%	86.86%

Table 2: Percentage of static and dynamic checks eliminated: Elimination of Redundant Tests

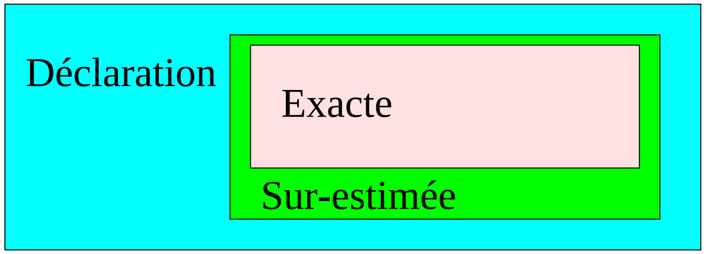
Insertion des tests inévitables

- Région de tableau : ensemble d'éléments d'un tableau représenté par un polyèdre convexe défini par des égalités et des inégalités
- Contraintes des paramètres des dimensions de tableau et des variables du programme
- Région sous-estimée, sur-estimée, exacte
- Exemple :

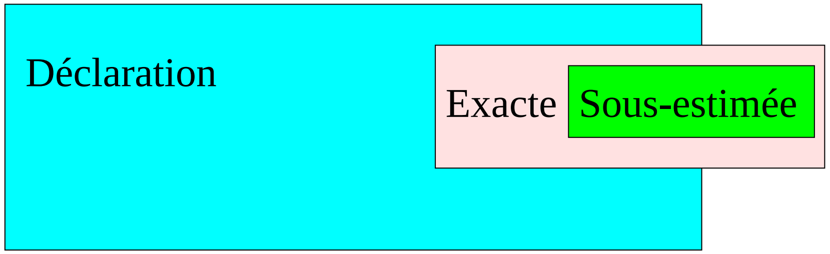
```
C <A(PHI1)-MAY-{1<=PHI1, PHI1<=N}>
    DO 10 I = 1,N,2
C <A(PHI1)-EXACT-{PHI1==I}>
        A(I) = 0.0
10  CONTINUE
```

Insertion des tests inévitables (2)

2 propriétés :



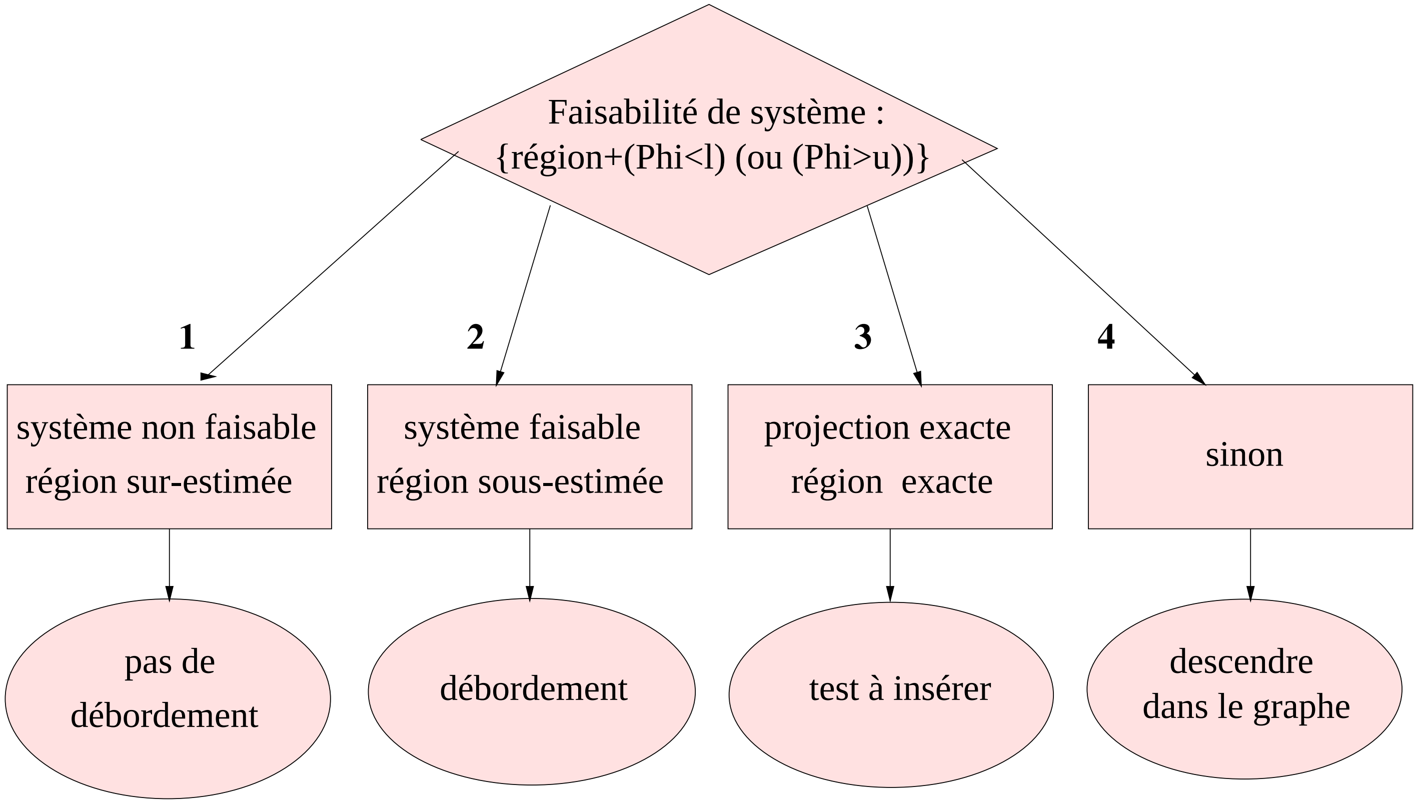
=> Pas de débordement



=> Débordement

Insertion des tests inévitables (3)

Algorithme (top-down) : commence par le nœud le plus haut dans le graphe de contrôle hiérarchique



```
      PROGRAM INSERTION1
      REAL U(15)
      N = 10
C <U(PHI1)-MAY-{N==10, 1≤PHI1, PHI1≤9}>
      DO 10 I = 1,N,2
C <U(PHI1)-EXACT-{PHI1==I, N==10, 1≤I, I≤10}>
         U(I) = I
10    CONTINUE
      END
```

1)*Région sur-estimée, pas de débordement*

```
      PROGRAM INSERTION2
      REAL U(15)
C <U(PHI1)-MUST-{PHI1==16}>
      U(16) = 0.
      DO 10 I = 1,N
        U(I) = I
10    CONTINUE
      END
```

2)*Région sous-estimée, débordement*

```
      PROGRAM INSERTION3
      REAL U(15)
C <U(PHI1)-EXACT-{1≤PHI1, PHI1≤N}>
      DO 10 I = 1,N
C <U(PHI1)-EXACT-{PHI1==I, 1≤I, I≤N}>
        U(I) = I
10 CONTINUE
      END
```

```
      PROGRAM INSERTION3
      REAL U(15)
      IF (16.LE.N) STOP
      DO 10 I = 1,N
        U(I) = I
10 CONTINUE
      END
```

3) *Région exacte, test à insérer*

```
      PROGRAM INSERTION4
      REAL U(15)
C <U(PHI1)-MAY-{1≤PHI1, PHI1≤N}>
      DO 10 I = 1,N,2
C <U(PHI1)-EXACT-{PHI1==I, 1≤I, I≤N}>
         U(I) = I
10    CONTINUE
      END
```

```
      PROGRAM INSERTION4
      REAL U(15)
      DO 10 I = 1,N,2
         IF (16.LE.I) STOP
         U(I) = I
10    CONTINUE
      END
```

4)*Descendre dans le graphe*

Program	**Static checks**	**Dynamic checks**
tomcatv	100.00%	100.00%
swim	84.72%	99.99%
su2cor	94.48%	97.66%
hydro2d	93.85%	99.46%
mgrid	97.93%	99.66%
applu	96.41%	99.87%
turb3d	94.33%	99.99%
modturb3d	98.97%	85.57%
apsi	96.42%	99.99%
modapsi	99.79%	99.99%
fpppp	92.81%	99.99%
modfpppp	95.82%	97.40%
wave5	94.01%	91.29%

Table 3: Percentage of static and dynamic checks eliminated: Insertion of Unavoidable Tests

Résultats expérimentaux

SPEC CFP95 benchmark

- Pourcentage de tests éliminés :
  1. Élimination des tests redondants
     - statique (à la compilation) : **96,34%**
     - dynamique (à l'exécution): **95,40%**
  2. Insertion des tests inévitables
     - statique (à la compilation) : **95,35%**
     - dynamique (à l'exécution): **97,08%**
- Débordement : *125.turb3d, 141.apsi, 145.fpppp* (REAL A(1))

Résultats expérimentaux (2)

Temps de compilation :

- Parcourir, analyser, optimiser, générer le code
- Avec 31605 lignes de code et 374 sous-routines : 8mn 55s pour Élimination des tests redondants 29mn 38s pour Insertion des tests inévitables
- Élimination des tests redondants : 66,07 lignes/seconde
- Insertion des tests inévitables : 17,24 lignes/seconde
- Test de faisabilité pour calculer les régions de tableaux (*107.mgrid, 141.apsi, 145.fpppp, 146.wave5*)

	Elimination			Insertion			SUN F77	
Program	PIPS	SUN	**Total**	PIPS	SUN	**Total**	**Wo -C**	**W -C**
tomcatv	0:02	0:02	**0:04**	0:05	0:02	**0:07**	**0:03**	**0:13**
swim	0:04	0:02	**0:06**	0:12	0:05	**0:17**	**0:04**	**0:20**
su2cor	0:40	0:25	**1:05**	2:28	0:33	**3:01**	**0:33**	**2:23**
hydro2d	0:16	0:19	**0:35**	0:43	0:32	**1:15**	**0:32**	**1:03**
mgrid	0:04	0:07	**0:11**	0:38	0:06	**0:44**	**0:08**	**0:37**
applu	0:33	0:25	**0:58**	2:02	0:54	**2:56**	**0:57**	**11:12**
modturb3d	0:15	0:16	**0:31**	1:05	0:17	**1:22**	**0:17**	**0:44**
modapsi	1:16	1:02	**2:18**	7:42	1:03	**8:45**	**1:16**	**3:15**
modfpppp	0:42	0:45	**1:27**	4:01	0:56	**4:57**	**0:54**	**1:22**
wave5	5:03	1:25	**6:28**	12:04	1:50	**13:54**	**2:04**	**6:20**

Table 4: Temps de compilation

Résultats expérimentaux (3)

Temps d'exécution : compilateurs et plate-formes différents

- Élimination des tests redondants
  - 3,94 fois plus vite que SUN avec -C
  - 1,88 fois plus vite que SGI avec -C
  - 1,03 fois plus vite que IBM avec -C
  - 19,29% plus long que SUN sans -C
  - 7,05% plus long que SGI sans -C
  - 16,59% plus long que IBM sans -C

Résultats expérimentaux (4)

Temps d'exécution :

- Insertion des tests inévitables
  - 4,37 fois plus vite que SUN avec -C
  - 2,02 fois plus vite que SGI avec -C
  - 1,07 fois plus vite que IBM avec -C
  - 5,33% plus long que SUN sans -C
  - 0,61% plus long que SGI sans -C
  - 10,62% plus long que IBM sans -C

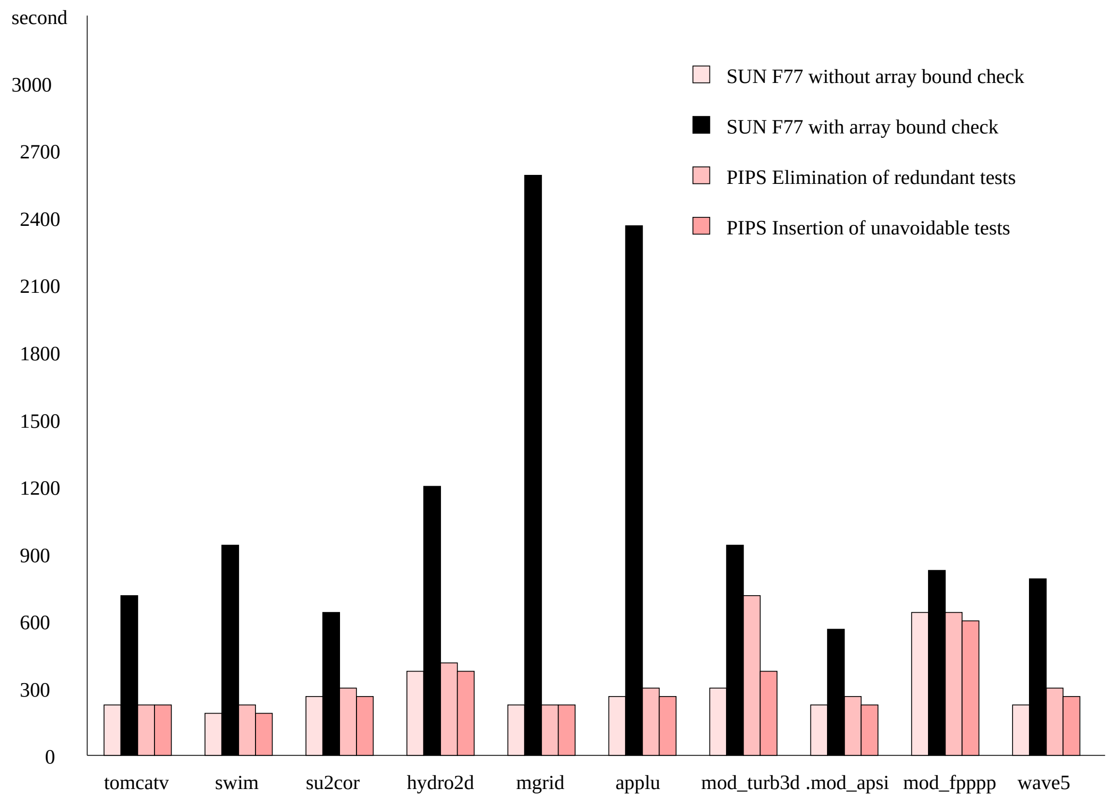


Figure 3: Execution time: SUN F77 without array bound checking, SUN F77 with array bound checking, PIPS Elimination of Redundant Tests, PIPS Insertion of Unavoidable Tests (SUN Workshop F77 5.0, Ultra SPARC 360MHz, Solaris 7) - Optimized code

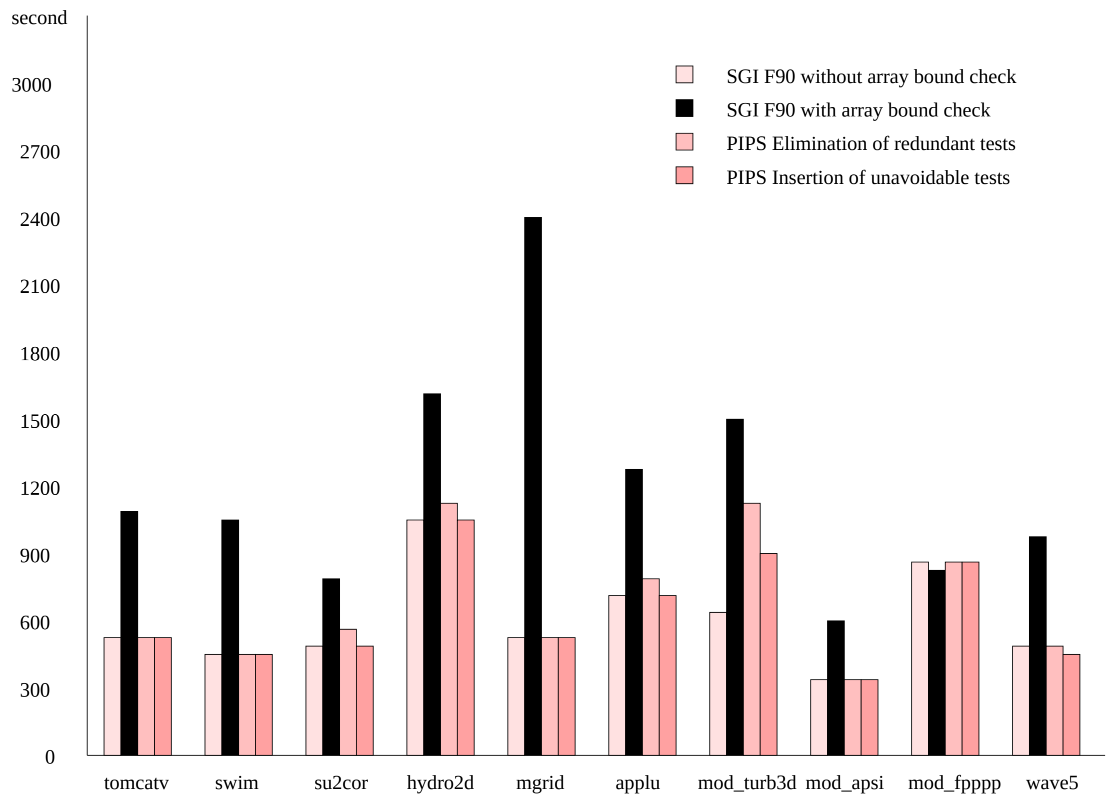


Figure 4: Execution time: SGI F90 without array bound checking, SGI F90 with array bound checking, PIPS Elimination of Redundant Tests, PIPS Insertion of Unavoidable Tests (SGI MIPSpro F90 7.3, O2 R5000 195MHz, IRIX 6.3) - Optimized code (f90 -Ofast=ip32_5k)

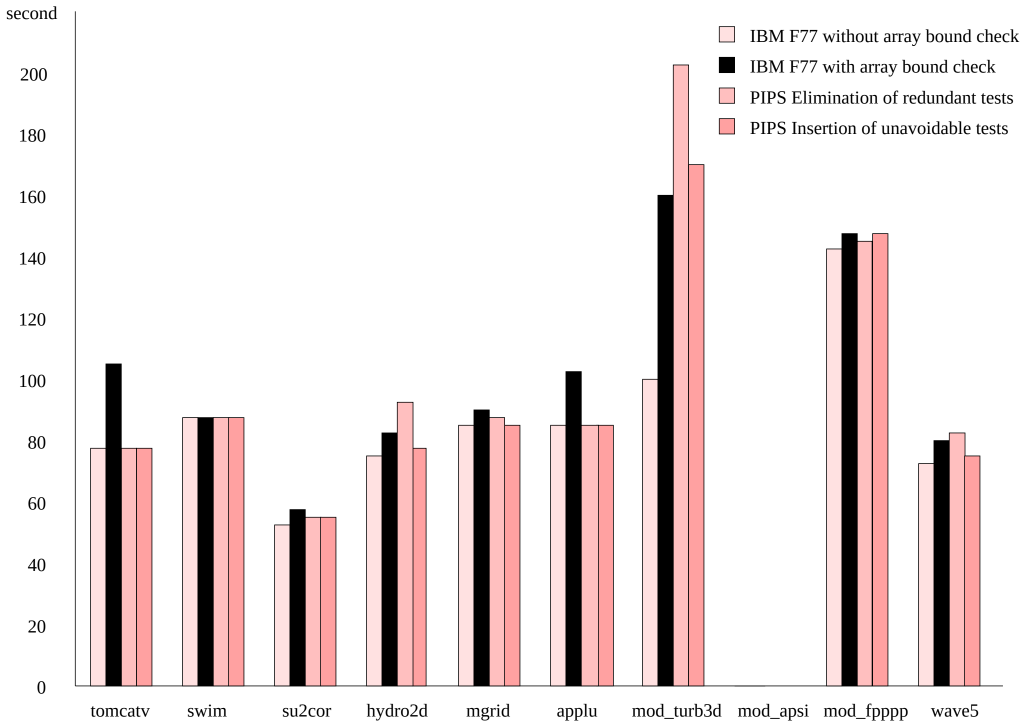


Figure 5: Execution time: IBM F77 without array bound checking, IBM F77 with array bound checking, PIPS Elimination of Redundant Tests, PIPS Insertion of Unavoidable Tests (IBM XL F77 7.1, RS/6000 44P-270 375MHz 4 CPU, AIX 4.3) - Optimized code (f77 -O3 -lmass)

Conclusion

- Analyse de flot de données : point fixe, précondition, région, ...
- Élimination des tests redondants : pas de déplacement de code
- Insertion des tests inévitables : conclure le plus tôt possible s'il y a débordement de tableau ou pas
- Différences : ex. 5,32% pour *104.hydro2d*
- Qualité du code : déclaration propre, test de paramètres,...

Conclusion (2)

- Techniques supplémentaires :
  - restructuration de code
  - clonage des procédures (module ADLEN de *104.hydro2d* : 94,14% $\rightarrow$ 99,50%)
  - test de borne de paramètres lus (*110.applu* : 97,09% $\rightarrow$ 100,00%)
  - linéarisation des bornes de boucles
  - modification de déclaration de tableau
- Améliorations :
  1. Déplacement de code
  2. Réplication de code pour région sur-estimée, région gardée, liste de régions, traitement des régions dimension par dimension, ...

Conclusion (3)

- Nombre de tests éliminés et temps d'exécution : pas proportionnel
- 3000 lignes de code C dans PIPS pour la vérification intra-procédurale efficace des accès aux tableaux
- Vérification inter-procédurale :
  - analyse inter-procédurale
  - redimensionnement de tableau
  - analyse d'alias de tableau

```
   PROGRAM INTERVERIF
   COMMON /FOO/ Z1
   DIMENSION Z1(10)
   CALL ZERO(Z1,20)
   PRINT *, Z1
   END

   SUBROUTINE ZERO(X, N)
   REAL X(N)
   DO 100 I=1,N
     X(I) = 0.
10 CONTINUE
   END
```

Vérification interprocédurale

```
      PROGRAM MAIN
      REAL A(201,200),B(200),X(200)
      INTEGER IPVT(200)
      LDA = 201
      N = 100
      CALL DGESL(A,LDA,N,IPVT,B)
      END

      SUBROUTINE DGESL(A,LDA,N,IPVT,B)
      INTEGER LDA,N,IPVT(1)
      REAL A(LDA,1),B(1)
      DO 20 K = 1, N-1
        CALL DAXPY(N-K,T,A(K+1,K),B(K+1))
   20 CONTINUE
      END

      SUBROUTINE DAXPY(N,DA,DX,DY)
      REAL DX(*),DY(*),DA
      DO 30 I = 1,N
        DY(I) = DY(I) + DA*DX(I)
   30 CONTINUE
      END
```

Redimensionnement de tableau

```
      PROGRAM INTERVERIF
      INTEGER IJKM
      PARAMETER (IJKM=100, A=5, B=6)
      REAL X,Y,Z
      COMMON /TOTAL/ X, Y, Z
      DIMENSION X(IJKM), Y(IJKM), Z(IJKM)
      DO 10 I = 1,IJKM
        X(I)= I
   10 CONTINUE
      ILONG = 3 * IJKM
      CALL CUMUL(X, X, A,B, ILONG)
      DO 20 I=1,ILONG
        PRINT Y(I)
   20 CONTINUE
      END

      SUBROUTINE CUMUL(X,Y,A,B,N)
      REAL X(N),Y(N)
      DO I =1,N
        Y(I) = X(I) + A*B
      ENDDO
      END
```

Vérification interprocédurale et analyse d'alias