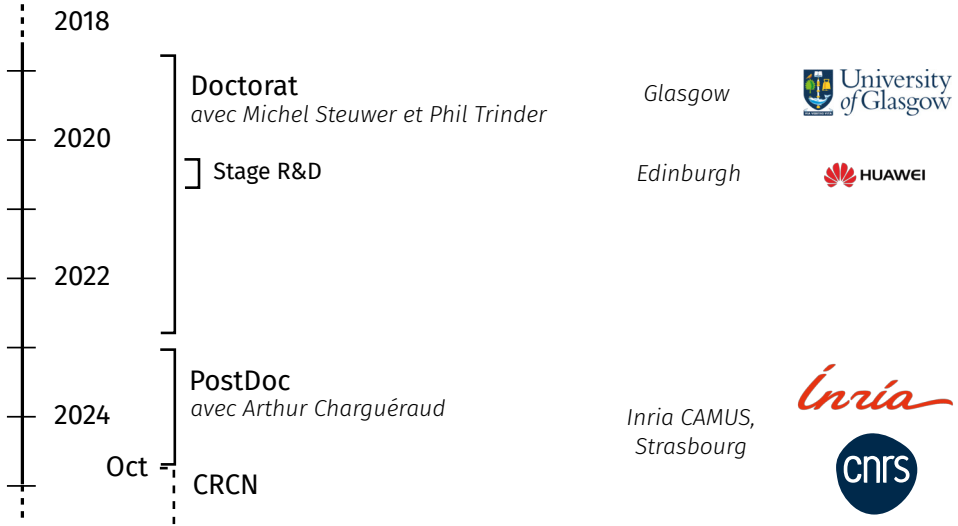
Optimisation de Programmes Interactive et Holistique

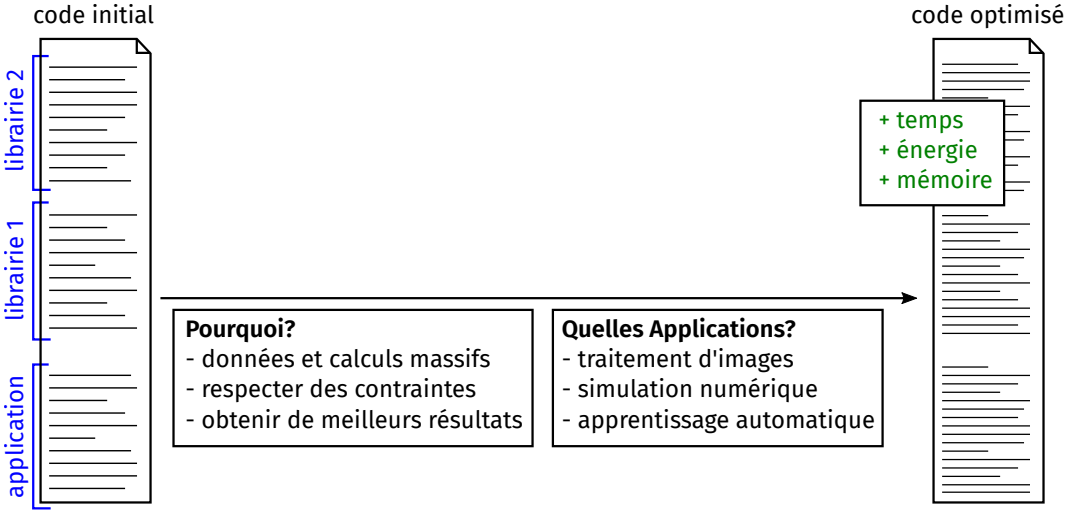
Thomas Kœhler thok.eu

Laboratoire Méthodes Formelles — Juillet 2024

Mon parcours



Optimisation de programmes



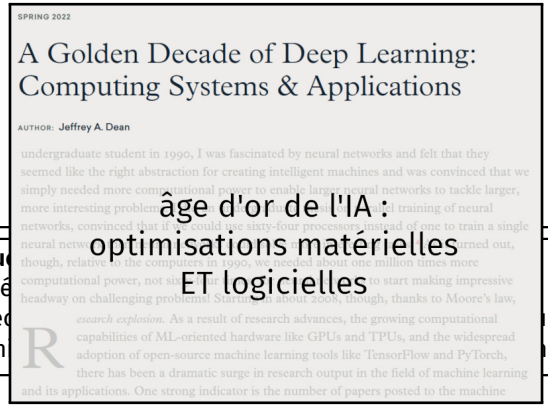
Optimisation de programmes

code initial

librairie 2

librairie 1

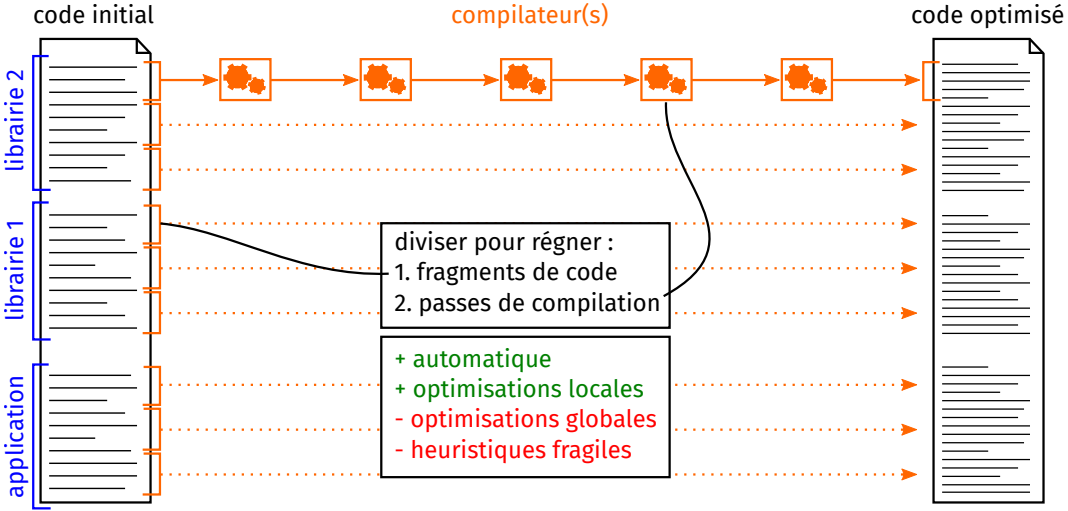
application



code optimisé

+ temps
+ énergie
+ mémoire

Optimisation de programmes automatique



Optimisation de programmes automatique

code initial

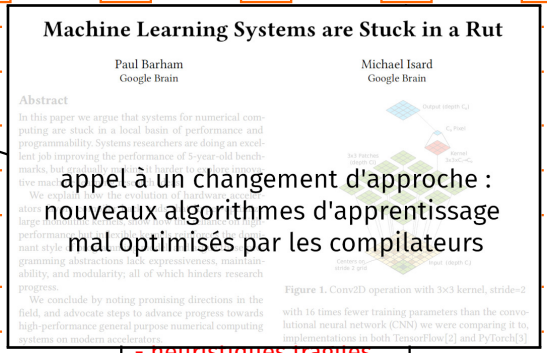
compilateur(s)

code optimisé

librairie 2

librairie 1

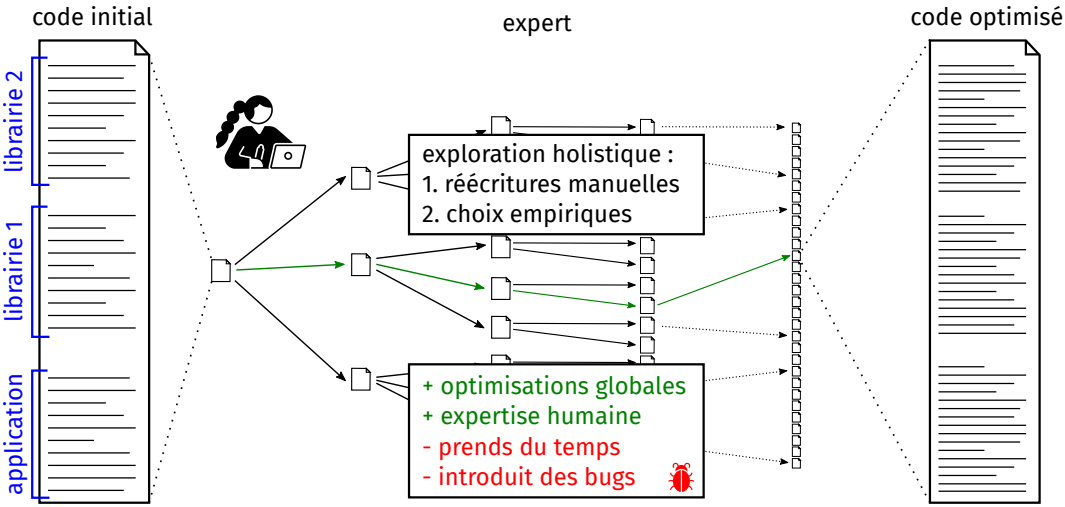
application



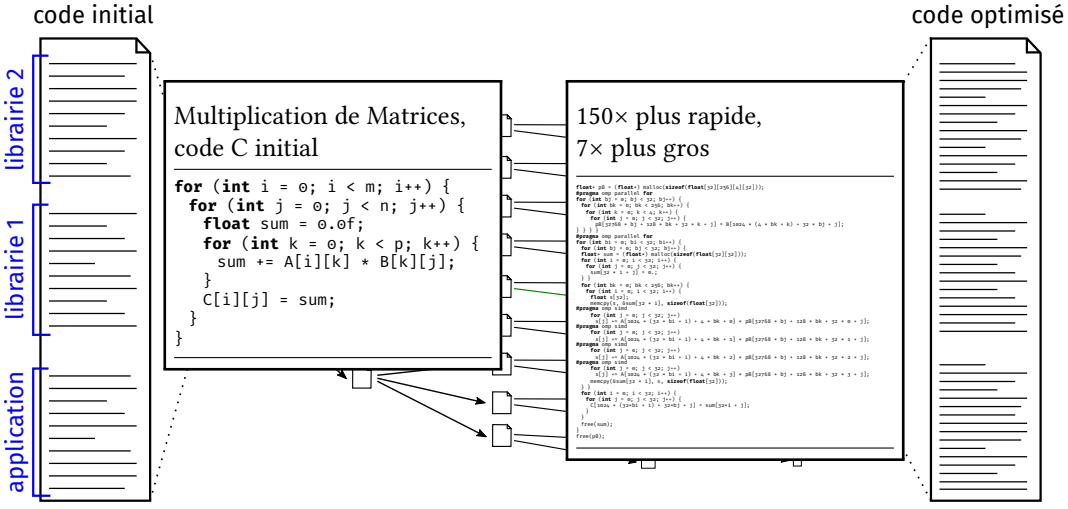
appel à un changement d'approche :
nouveaux algorithmes d'apprentissage
mal optimisés par les compilateurs

- heuristiques fragiles

Optimisation de programmes manuelle



Optimisation de programmes manuelle

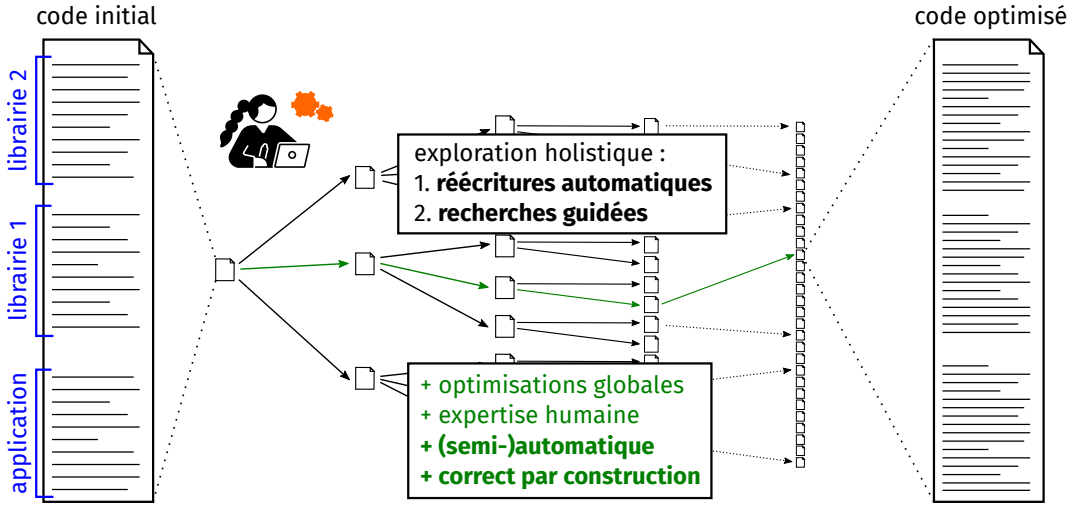


Optimisation de programmes : quel futur ?

- les compilateurs ne répondent ni aux besoins actuels ni futurs d'optimisation
- les algorithmes et architectures matérielles évoluent plus vite que les compilateurs
- le repli sur l'optimisation manuelle **ralenti le progrès**

Présentation à PLDI 2024: The Future of Fast Code: Giving Hardware What It Wants

Optimisation de programmes interactive et holistique

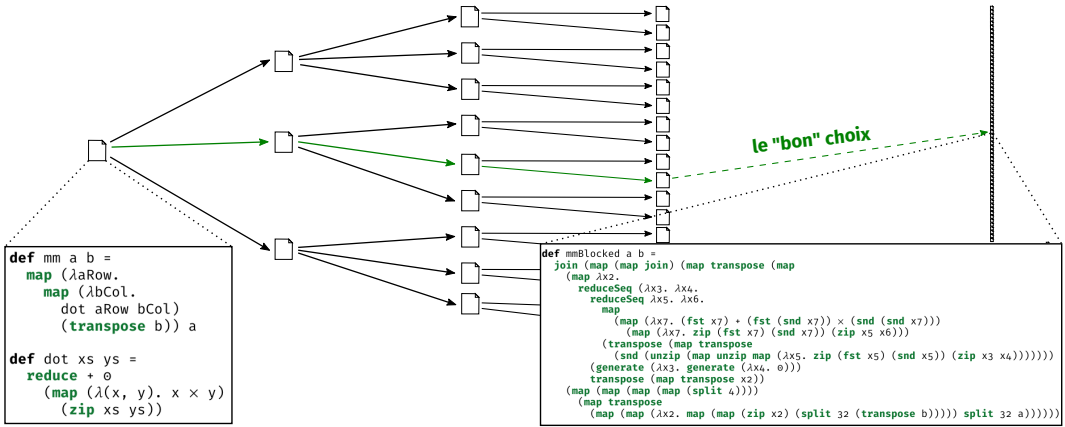


Doctorat : réécriture de programmes fonctionnels

règles de réécritures

```
(map f) . (map g) = map (f . g)
```

espace de réécriture combinatoire, correct et extensible



Doctorat : optimisations expertes, par composition

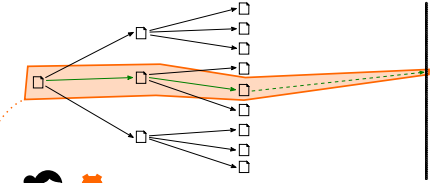
6 optimisations d'experts
⟶ décomposées en **74 règles**

Harris: détection de coins et de contours

16x plus rapide que OpenCV,
1.4x plus rapide que Halide! **[CGO'21** *A***]**

compilateur spécialisé de qualité industrielle

espace de réécriture immense, programmes 10x plus gros qu'avant,
des milliers d'étapes de réécriture



stratégies de réécriture :
[ICFP'20 *A**, **CACM'23** 🏆 **]**

```
baseline ;
tile(32,32)      @ outermost(mapNest(2))   ;;
fissionReduceMap @ outermost(appliedReduce);;
split(4)         @ innermost(appliedReduce);;
reorder(List(1,2,5,6,3,4))
```

Doctorat : recherche automatique et expertise humaine

saturation d'égalité guidée [POPL'24 *A]**

=

recherche de réécriture automatique, partage des sous-termes équivalents

+

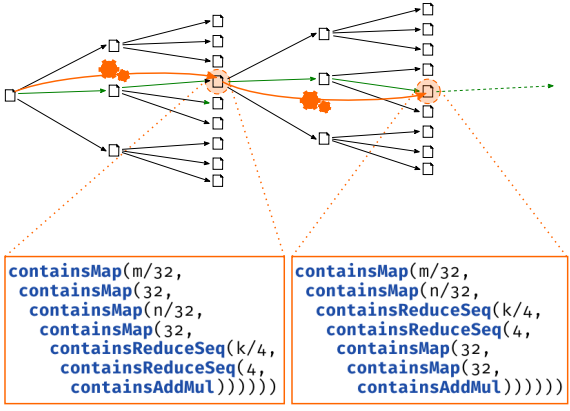
spécification de guides sous forme d'esquisses de programmes

MIT PL Review : 🏆

"significant potential to shape the future direction of PL research and/or industry practice"

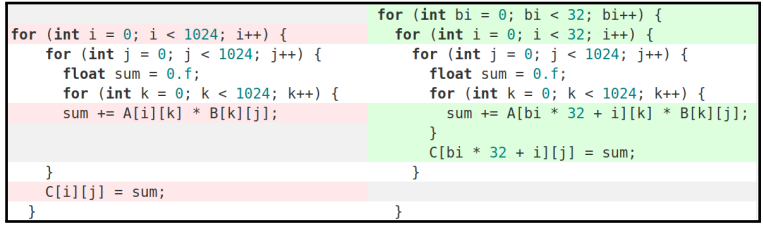
collaboration sur la preuve de théorèmes :
Andrés Goens, University of Amsterdam

recherche guidée :
582x plus rapide, 116x moins de mémoire



PostDoc : transformation de programmes impératifs

- rajout des **choix impératifs** dans l'espace d'optimisation *réutilisation de mémoire*
- **langage généraliste** plus expressif qu'un langage fonctionnel tableau *C, C++*



```
for (int i = 0; i < 1024; i++) {
    for (int j = 0; j < 1024; j++) {
      float sum = 0.f;
      for (int k = 0; k < 1024; k++) {
      sum += A[i][k] * B[k][j];
    }
    C[i][j] = sum;
  }
```

```
for (int bi = 0; bi < 32; bi++) {
  for (int i = 0; i < 32; i++) {
    for (int j = 0; j < 1024; j++) {
      float sum = 0.f;
      for (int k = 0; k < 1024; k++) {
        sum += A[bi * 32 + i][k] * B[k][j];
      }
      C[bi * 32 + i][j] = sum;
    }

  }
```

Le projet OptiTrust :

- transformations source-à-source guidées par l'utilisateur
- validées par une analyse statique des ressources *basée sur la logique de séparation*
- en interne : λ-calcul impératif

PostDoc : optimisations inédites sur du code impératif généraliste

Etudes de cas réalisées :

- Multiplication de Matrices
  algèbre linéaire, référence : compilateur spécialisé TVM
- Filtre Boîte (sur lignes) : floutage et débruitage
  traitement d'image, référence : optimisations manuelles de la librairie OpenCV

Travail en cours :

- Harris : détection de coins et de contours
  traitement d'image, référence : compilateur spécialisé Halide
- Particle-In-Cell : simulation de plasmas
  simulation physique, référence : optimisations manuelles d'expert

Optimisation de MatMul avec OptiTrust

Quoi calculer : code généraliste

```
void mm(float* C, float* A, float* B,
        int m, int n, int p) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
      float sum = 0.0f;
      for (int k = 0; k < p; k++)
        sum += A[i][k] * B[k][j];
      C[i][j] = sum;
    }
  }
}
```

Comment optimiser :
script de transformation

```
Function.inline_def [cFunDef "mm"];
let tile (id, tile_size) = Loop.tile (int tile_size)
  ~index:("b" ^ id) ~bound:TileDivides [cFor id] in
List.iter tile [("i", 32); ("j", 32); ("k", 4)];
Loop.reorder_at
  ~order:["bi"; "bj"; "bk"; "i"; "k"; "j"] [cPlusEq ()];
Loop.hoist_expr ~dest:[tBefore; cFor "bi"] "pB"
  ~indep:["bi"; "i"] [cArrayRead "B"];
Matrix.stack_copy ~var:"sum" ~copy_var:"s"
  ~copy_dims:1 [cFor ~body:[cPlusEq ()] "k"];
Omp.simd [cFor ~body:[cPlusEq ()] "j"];
Omp.parallel_for [cFunBody "mm1024"; cStrict; cFor ""];
Loop.unroll [cFor ~body:[cPlusEq ()] "k"];
```

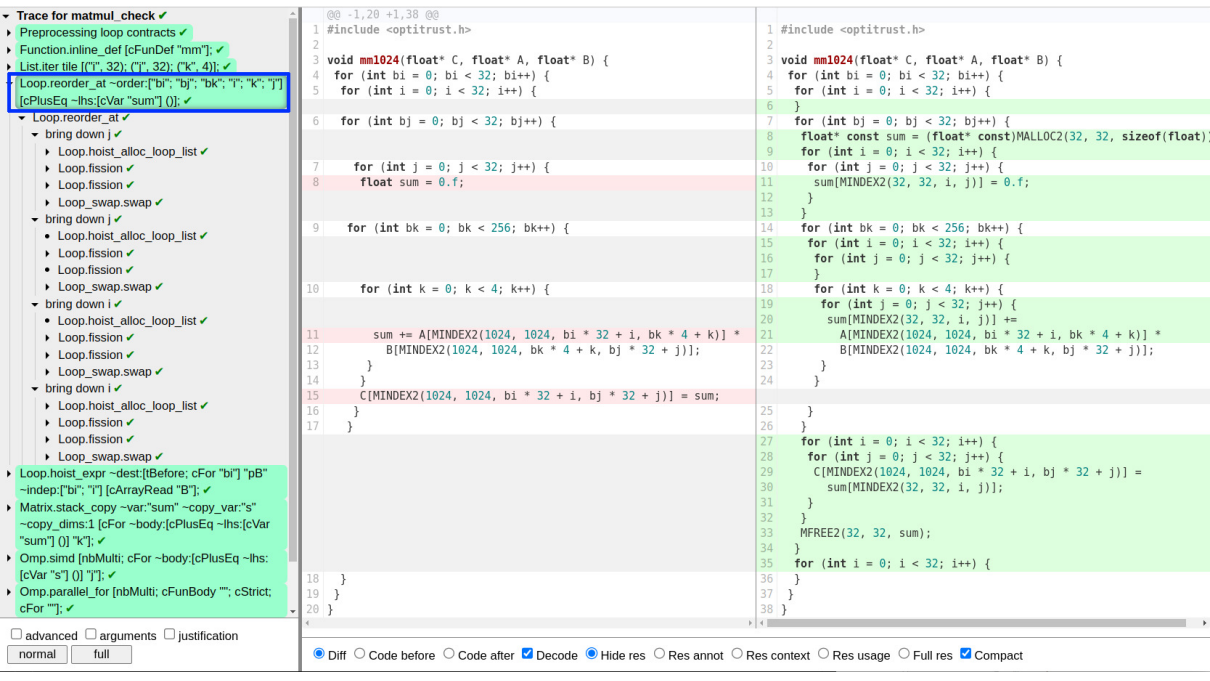
Optimisation de MatMul avec OptiTrust

Quoi calculer : code généraliste

```
void mm(float* C, float* A, float* B,
        int m, int n, int p) {
  __reads("A ~> Matrix2(m, p)");
  __reads("B ~> Matrix2(p, n)");
  __modifies("C ~> Matrix2(m, n)");
  for (int i = 0; i < m; i++) {
    __xmodifies("for j in 0..n→  &C[i][j]~> Cell")
        ;
    for (int j = 0; j < n; j++) {
      __xmodifies("&C[i][j] ~> Cell");
      float sum = 0.0f;
      for (int k = 0; k < p; k++)
        sum += A[i][k] * B[k][j];
      C[i][j] = sum;
    }
  }
}
```

Comment optimiser :
script de transformation

```
Function.inline_def [cFunDef "mm"];
let tile (id, tile_size) = Loop.tile (int tile_size)
  ~index:("b" ^ id) ~bound:TileDivides [cFor id] in
List.iter tile [("i", 32); ("j", 32); ("k", 4)];
Loop.reorder_at
  ~order:["bi"; "bj"; "bk"; "i"; "k"; "j"] [cPlusEq ()];
Loop.hoist_expr ~dest:[tBefore; cFor "bi"] "pB"
  ~indep:["bi"; "i"] [cArrayRead "B"];
Matrix.stack_copy ~var:"sum" ~copy_var:"s"
  ~copy_dims:1 [cFor ~body:[cPlusEq ()] "k"];
Omp.simd [cFor ~body:[cPlusEq ()] "j"];
Omp.parallel_for [cFunBody "mm1024"; cStrict; cFor ""];
Loop.unroll [cFor ~body:[cPlusEq ()] "k"];
```


Trace for matmul_check ✔
Preprocessing loop contracts ✔
Function.inline_def [cFunDef "mm"]; ✔
List.iter tile [("i", 32); ("j", 32); ("k", 4)]; ✔
Loop.reorder_at ~order:["bi"; "bj"; "bk"; "i"; "k"; "j"] [cPlusEq ~lhs:[cVar "sum"] ()]; ✔
Loop.reorder_at ✔
bring down j ✔
Loop.hoist_alloc_loop_list ✔
Loop.fission ✔
Loop.fission ✔
Loop_swap.swap ✔
bring down j ✔
Loop.hoist_alloc_loop_list ✔
Loop.fission ✔
Loop.fission ✔
Loop_swap.swap ✔
bring down i ✔
Loop.hoist_alloc_loop_list ✔
Loop.fission ✔
Loop.fission ✔
Loop_swap.swap ✔
bring down i ✔
Loop.hoist_alloc_loop_list ✔
Loop.fission ✔
Loop.fission ✔
Loop_swap.swap ✔
Loop.hoist_expr ~dest:[tBefore; cFor "bi"] "pB" ~indep:["bi"; "i"] [cArrayRead "B"]; ✔
Matrix.stack_copy ~var:"sum" ~copy_var:"s" ~copy_dims:1 [cFor ~body:[cPlusEq ~lhs:[cVar "sum"] ()] "k"]; ✔
Omp.simd [nbMulti; cFor ~body:[cPlusEq ~lhs:[cVar "s"] ()] "j"]; ✔
Omp.parallel_for [nbMulti; cFunBody ""; cStrict; cFor ""]; ✔
advanced
arguments
justification
normal
full
@@ -1,20 +1,38 @@
#include <optitrust.h>
void mm1024(float* C, float* A, float* B) {
for (int bi = 0; bi < 32; bi++) {
for (int i = 0; i < 32; i++) {
for (int bj = 0; bj < 32; bj++) {
for (int j = 0; j < 32; j++) {
float sum = 0.f;
for (int bk = 0; bk < 256; bk++) {
for (int k = 0; k < 4; k++) {
sum += A[MINDEX2(1024, 1024, bi * 32 + i, bk * 4 + k)] *
B[MINDEX2(1024, 1024, bk * 4 + k, bj * 32 + j)];
}
}
C[MINDEX2(1024, 1024, bi * 32 + i, bj * 32 + j)] = sum;
}
}
}
}
#include <optitrust.h>
void mm1024(float* C, float* A, float* B) {
for (int bi = 0; bi < 32; bi++) {
for (int i = 0; i < 32; i++) {
}
for (int bj = 0; bj < 32; bj++) {
float* const sum = (float* const)MALLOC2(32, 32, sizeof(float));
for (int i = 0; i < 32; i++) {
for (int j = 0; j < 32; j++) {
sum[MINDEX2(32, 32, i, j)] = 0.f;
}
}
for (int bk = 0; bk < 256; bk++) {
for (int i = 0; i < 32; i++) {
for (int j = 0; j < 32; j++) {
}
for (int k = 0; k < 4; k++) {
for (int j = 0; j < 32; j++) {
sum[MINDEX2(32, 32, i, j)] +=
A[MINDEX2(1024, 1024, bi * 32 + i, bk * 4 + k)] *
B[MINDEX2(1024, 1024, bk * 4 + k, bj * 32 + j)];
}
}
}
}
for (int i = 0; i < 32; i++) {
for (int j = 0; j < 32; j++) {
C[MINDEX2(1024, 1024, bi * 32 + i, bj * 32 + j)] =
sum[MINDEX2(32, 32, i, j)];
}
}
MFREE2(32, 32, sum);
}
for (int i = 0; i < 32; i++) {
}
}
}
Diff
Code before
Code after
Decode
Hide res
Res annot
Res context
Res usage
Full res
Compact

Décomposition des étapes de MatMul par OptiTrust

Transformations combinées :

- composition de transformations basiques
- validées par transitivité

Transformations basiques :

- modification directe de l'AST
  λ-calcul impératif
- vérification de conditions suffisantes pour la correction

8 étapes MatMul :

- 55 transformations basiques
- 61 transformations "ghost"

- `Instr` : `move`, `delete`, `insert`, `duplicate`
- `Function` : `inline`
- `Variable` : `inline`, `bind`, `to_const`, `init_detach`, `local_name`
- `Loop` : `fusion`, `fission`, `swap`, `hoist`, `move_out`, `tile`, `collapse`, `unroll`
- `Matrix` : `intro_malloc0`,
- `Omp` : `parallel_for`, `simd`
- `Arith` : `simpl`
- …

Condition suffisante d'OptiTrust pour `Instr.move`

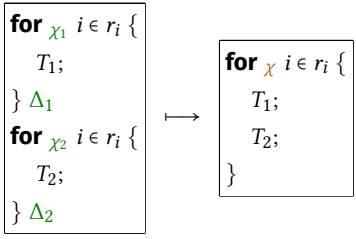
$$\boxed{E\left[\begin{array}{l} T_1; \Delta_1 \\ T_2; \Delta_2 \end{array}\right]} \longmapsto \boxed{E\left[\begin{array}{l} T_2; \\ T_1; \end{array}\right]}$$

correct if :

$$\begin{cases} \Delta_1.\mathsf{notRO} \cap \Delta_2 = \varnothing \\ \Delta_2.\mathsf{notRO} \cap \Delta_1 = \varnothing \end{cases}$$

- exploite les informations de ressources calculées par notre analyse (Δ_1)
- correct si les ressources sont exclusivement partagées en lecture seule

Condition suffisante d'OptiTrust pour `Loop.fusion`



correct if :

$$\begin{cases} i \text{ not free in } \chi_1.\mathsf{shrd} \text{ and } \chi_2.\mathsf{shrd} \\ \chi_1.\mathsf{shrd} \vdash (\Delta_1.\mathsf{notRO} \cap \Delta_2) = \varnothing \\ \chi_2.\mathsf{shrd} \vdash (\Delta_2.\mathsf{notRO} \cap \Delta_1) = \varnothing \end{cases}$$

with :

$$\_, \_, Q_1, P_2 \equiv \mathsf{PartialSub}(\chi_1.\mathsf{excl.post}, \chi_2.\mathsf{excl.pre})$$

$$\chi \equiv \begin{cases} \mathsf{vars} \equiv \chi_1.\mathsf{vars}, \chi_2.\mathsf{vars} \\ \mathsf{shrd} \equiv \chi_1.\mathsf{shrd} \star \chi_2.\mathsf{shrd} \\ \mathsf{excl.pre} \equiv \chi_1.\mathsf{excl.pre} \star P_2 \\ \mathsf{excl.post} \equiv \chi_2.\mathsf{excl.post} \star Q_1 \end{cases}$$

Plan : optimisation de programmes interactive et holistique

images

apprentissage

simulation

• • •

Objectif :
construire le permier
Assistant d'Optimisation

qui s'adapte aux évolutions des domaines et du matériel, évitant le repli sur l'optimisation manuelle

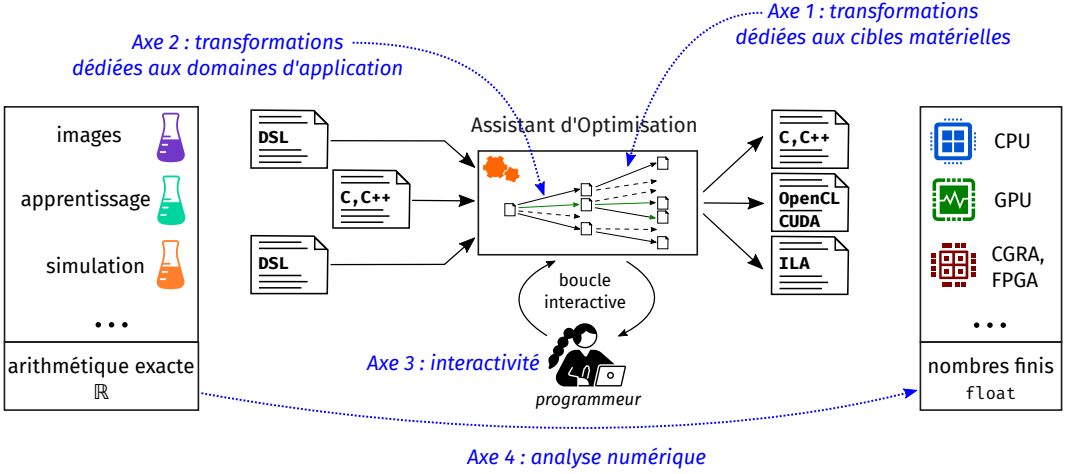
CPU

GPU

CGRA, FPGA

• • •

Plan : optimisation de programmes interactive et holistique

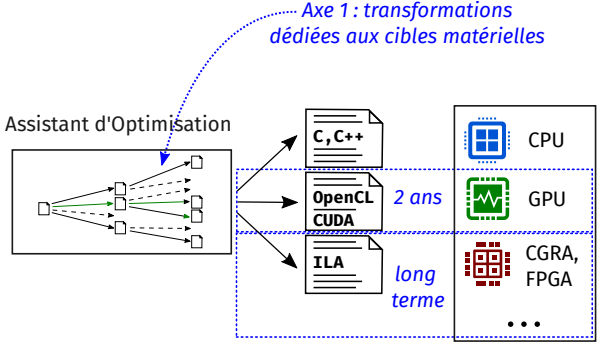


Axe 1 : transformations correctes dédiées au matériel

Objectif : créer un espace d'optimisation
qui évolue avec le matériel
par combinaison de transformations

Approche :
- décomposer les optimisations expertes
  en des transformations plus simples

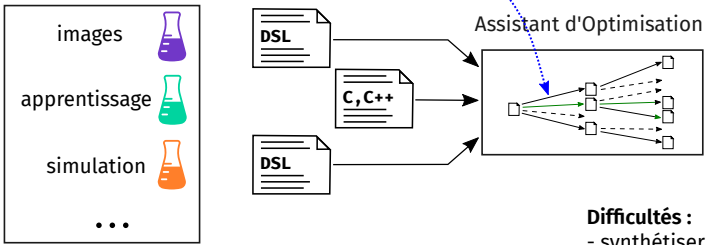
Difficulté :
- identifier et garantir la correction
  des transformations en prenant en
  compte les modèles de programmation



Collaboration Internationales :
- Bastian Köpcke, Universität Münster [**arXiv'22**, GPGPU'22, PLDI'24]
- Jackson Woodruff, University of Edinburgh [PLDI'22, **arXiv'23**, CC'23]

Axe 2 : transformation correctes dédiées aux domaines

Axe 2 : transformations dédiées aux domaines d'application



Objectif : créer un espace d'optimisation qui évolue avec les domaines et leurs langages dédiés *(DSLs)*

Plan :
- 2 ans : langage dédié type Halide
- 4 ans : synergie avec l'Axe 1 (GPUs)
- long terme : mélange de DSLs

Difficultés :
- synthétiser les annotations de code
- faciliter la définition de transformations dédiées

Contacts:
- Gabriel Radanne, Inria CASH, Lyon
- Roland Leissa, Universität Mannheim

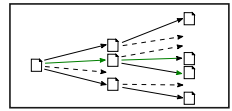
Axe 3 : interactivité entre programmeur et assistant

Objectif : développer une boucle interactive permettant
d'explorer l'espace d'optimisation de façon productive

Plan :
- 3 ans : saturation d'égalité interactive sur du code impératif
  recherche polyédrique et/ou par faisceau
- long-terme : suggestions d'optimisation et visualisations
  goulots d'étranglement, code chaud

Difficulté :
- exploration d'un espace d'optimisation complexe

Collaboration Interdisciplinaire :
- Géry Casiez (Inria LOKI, Interaction Homme-Machine)

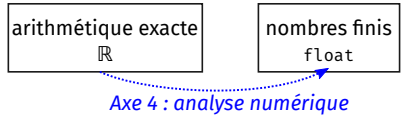


Axe 3 : interactivité

Axe 4 : précision numérique garantie pendant l'optimisation

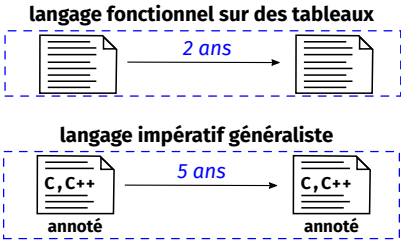
Objectif : garantir la précision numérique pendant l'optimisation, éviter les erreurs et permettre des optimisations

fonction trigonométrique approchée = 4x plus rapide
virgule fixe sur FPGA



Approche :
- analyse d'intervalles sur les valeurs et erreurs
  erreur bornée par `e(x o y) + d`

Difficulté :
- la plupart des compilateurs ne font pas d'analyses numériques
  'gcc': pas d'optimisations sur les flottants
  'gcc -ffast-math': traite les flottants comme des réels



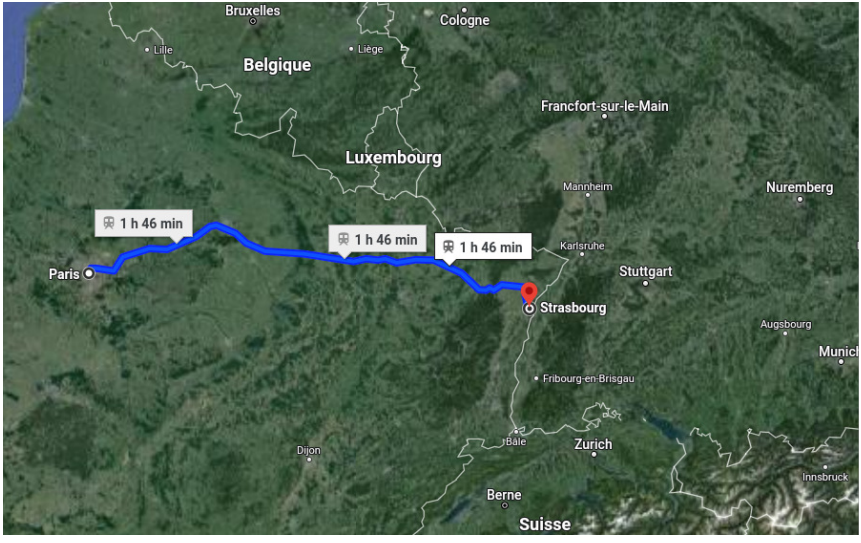
Collaboration Internationale :
- Eva Darulova, Uppsala University [TOPLAS'17, SAS'23]

Intéressé par un stage ?

- liberté de sujet au sein de mon projet
- vous donner les moyens d'être créatif, de devenir indépendant
- viser à publier dans les meilleures conférences internationales
- contacts et visites internationales
- possibilité de continuation en thèse

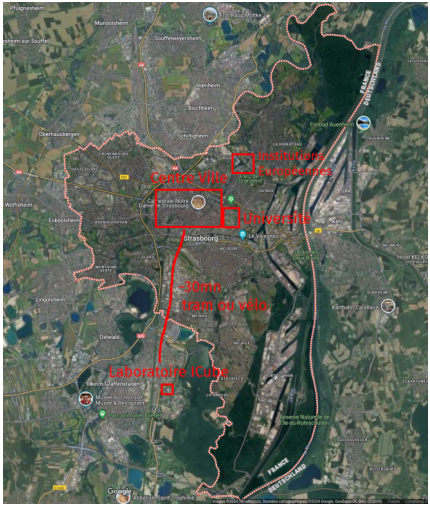
  plan : financement de thèse ANR JCJC

La ville de Strasbourg



La ville de Strasbourg

La ville de Strasbourg



- ~300 000 habitants
- 2ème meilleure ville étudiante de France *letudiant.fr*
  - >50 000 étudiants
- Cadre exceptionnel : architecture, culture, transport, sport, nature, ...
  - Centre pédestre et quartier Neustadt classés UNESCO
  - Trams et pistes cyclables
- Ville internationale
  - Institutions Européenes
  - Tram passe la frontière allemande
  - ~14% d'habitants internationaux

La ville de Strasbourg

L'équipe CAMUS/ICPS du Laboratoire ICube

Thématique : optimisation et compilation automatique, preuves et certification

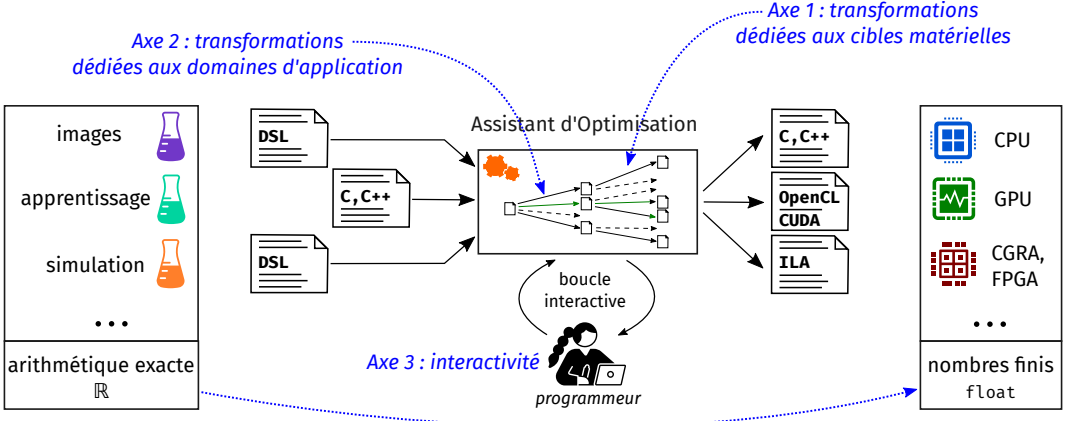
9 (enseigant-)chercheurs, 1 ingénieur, 3 postdocs, 9 doctorants

Doctorant sur OptiTrust : Guillaume Bertholon

Expertises :

- mathématiques des polyèdres *Philippe Clauss, Vincent Loechner, Alain Ketterlin*
- applications avancées *Vincent Loechner, Stéphane Genaud, Bérenger Bramas*
- annotations et raffinements dans le langage C *Jens Gustedt*
- logique de séparation et invariants de code *Arthur Charguéraud*

Optimisation de Programmes Interactive et Holistique

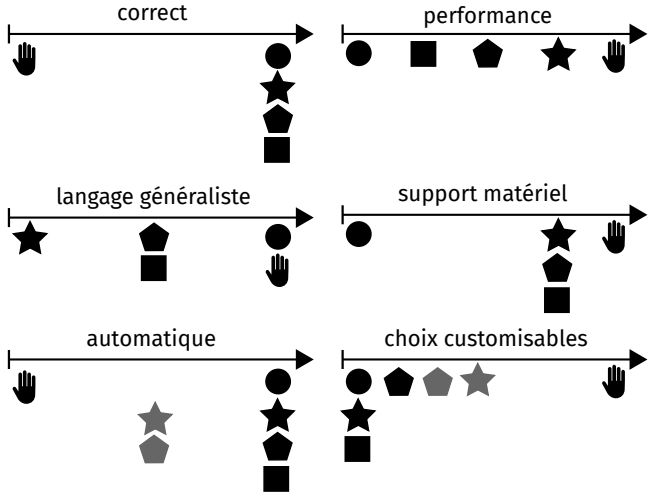


Thomas Kœhler thok.eu

Backup Slides

Les Compromis

- compilateurs généralistes
  gcc, clang
- optimisation manuelle
- compilateurs spécialisés
  Halide, TVM
- compilateurs polyédriques
  PLUTO, Polly
- compilateurs tableau
  Accelerate, SaC, Futhark



Les Compromis

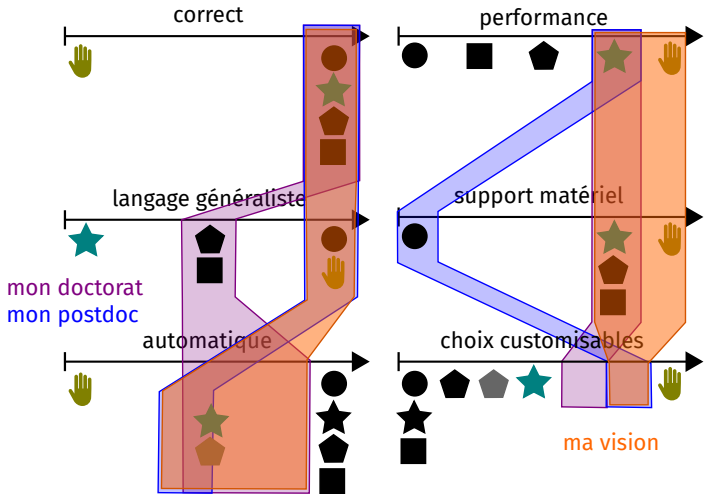
compilateurs généralistes
gcc, clang

optimisation manuelle

compilateurs spécialisés
Halide, TVM

compilateurs polyédriques
PLUTO, Polly

compilateurs tableau
Accelerate, SaC, Futhark

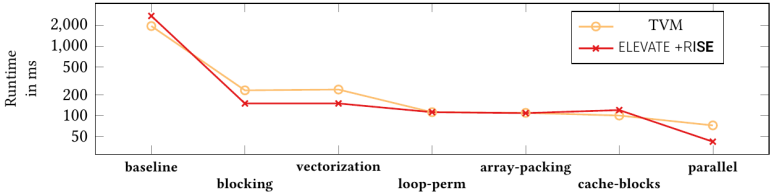


1st Benchmark : Matrix Multiplication on Intel CPU

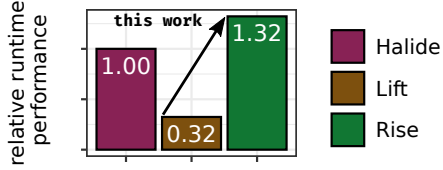
- 6 optimizations
  - transform loops *blocking, permutation, unrolling*
  - change data layout *array packing*
  - add parallelism *vectorization, multi-threading*

- performance is on par with reference schedules from TVM.

  https://tvm.apache.org/docs/how_to/optimize_operators/opt_gemm.html



2nd Benchmark : Corner Detection on ARM CPUs



- standard corner detection pipeline
- 6 well-known optimizations

  circular buffering, operator fusion, multi-threading, vectorization, convolution separation, register rotation

- extensibility + control

  $\Longrightarrow$

  faster code than Halide, with 2 additional optimizations

Matrix Multiplication Performance

- Intel(R) Core(TM) i7-8665U CPU, AVX2 (8 floats), 4 cores (8 hyperthreads)
- Relative speedup on 1024^3 input :

version	single-thread	multi-thread
unoptimized	1×	1×
optimized	46×	150×
TVM	46×	150×
numpy (Intel MKL) [1]	71×	183×

Both codes have 90th percentile runtime of 9.4ms over 200 benchmark runs, corresponding to a speedup of 150× compared to the 90th percentile of the naive code.

1. uses assembly code, explicit vectorization, custom thread library