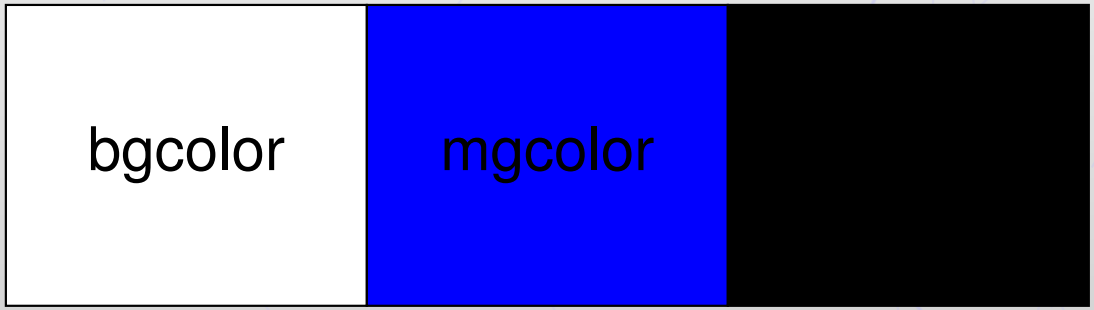
Sanity Check



Compilation pour cibles hétérogènes: automatisation des analyses, transformations et décisions nécessaires

Serge Guelton, François Irigoin et Ronan Keryell

Télécom Bretagne, Mines ParisTech, HPC Project

Renpar'20, S^{t} Malo, France

Besoins de performances ?

La quantité

La spécialisation

Dans la vraie vie . . .

- Contrainte d'énergie (Watt)
- Contrainte de place (mm^2)
- Coûts de fabrication (€)
- Coûts de refroidissement (Joule)
- `< votre chef peut insérer une contrainte ici >`

$\Rightarrow$ spécialisation sous contrainte

Dans la vraie vie . . .

- Contrainte d'énergie (Watt)
- Contrainte de place (mm^2)
- Coûts de fabrication (€)
- Coûts de refroidissement (Joule)
- `< votre chef peut insérer une contrainte ici >`

$\Rightarrow$ spécialisation sous contrainte

Problèmes liés à la spécialisation

Spécifique mais. . .

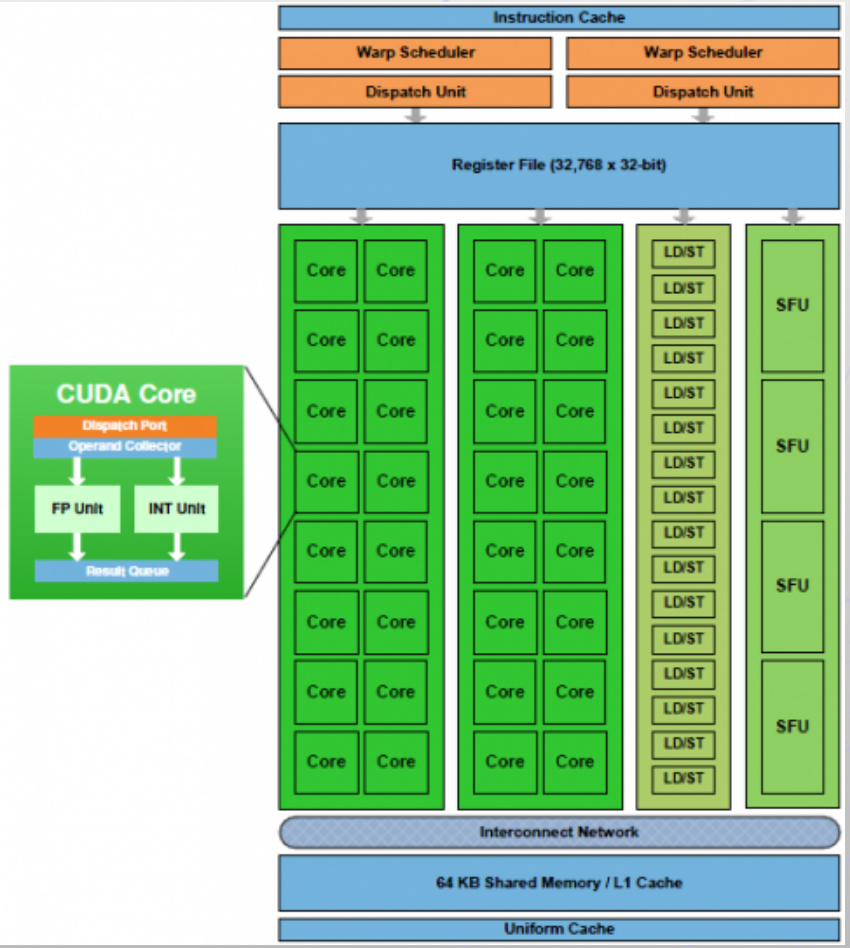
- Coût de conception (matériel)
- Coût de développement (logiciel)
- Peu d'effet de communauté pour les outils

Solutions

1. Bibliothèque applicative
2. Compilation

Exemple$_1$: carte graphique

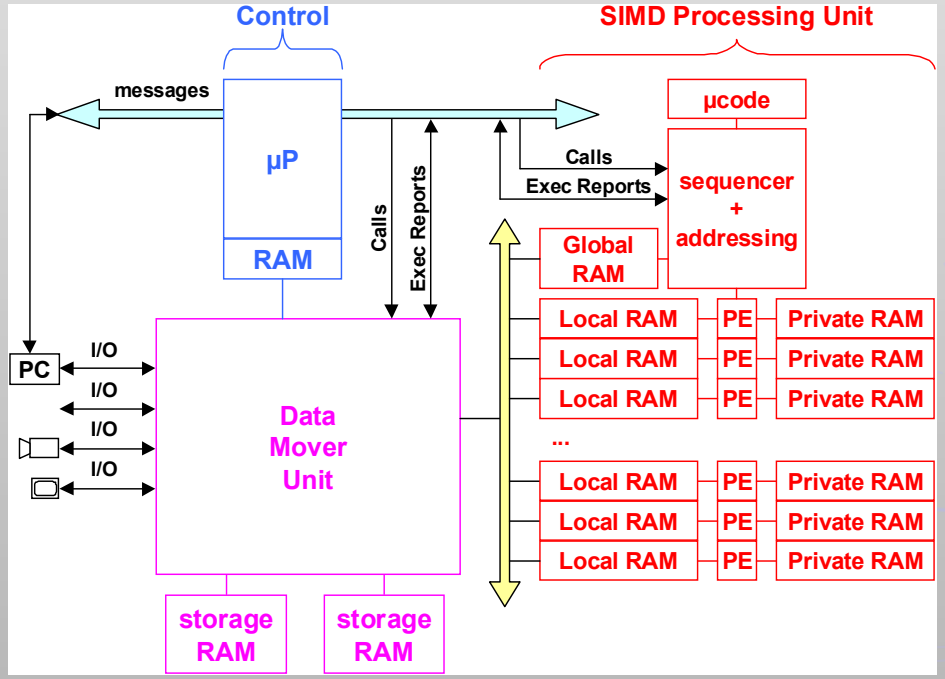
- ☺ Performance de crête
- ☺ Faible consommation



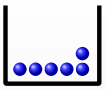
- ☹ Modèle d'exécution hybride SIMD/MIMD
- ☹ champs d'application
- ☹ coût de portage
- ☹ pérennité / évolution de l'architecture
- ☹ portabilité des performances
- ☹ coût de debug

Exemple$_2$: Terapix

- ☺ Bon rapport $\frac{\text{FLOPS}}{\text{Watt}}$
- ☺ Embarqué
- ☺ Spécialisé pour le traitement d'image
- ☺ À base de FPGA $\Rightarrow$ Indépendance



- ☹ Pas de compilateur
- ☹ Modèle d'exécution SIMD
- ☹ Pas d'exécution masquée
- ☹ Espaces d'adressage multiples
- ☹ Mémoire 2D
- ☹ Jeu d'instruction VLIW
- ☹ DMA 2D de taille fixe
- ☹ Mémoire limitée



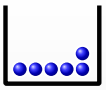
Problématique

Compilation pour accélérateurs matériels

Comment produire rapidement des compilateurs pour des accélérateurs spécifiques ?

Méthodologie

1. Identification des contraintes matérielles récurrentes
2. Identification/développement des transformations de code qui lèvent ces contraintes
3. Assemblage de compilateurs dédiés
4. Utilisation des compilateurs matériels (quand ils existent)



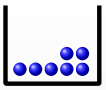
Problématique

Compilation pour accélérateurs matériels

Comment produire rapidement des compilateurs pour des accélérateurs spécifiques ?

Méthodologie

1. Identification des contraintes matérielles récurrentes
2. Identification/développement des transformations de code qui lèvent ces contraintes
3. Assemblage de compilateurs dédiés
4. Utilisation des compilateurs matériels (quand ils existent)



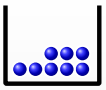
Contraintes matérielles I

Parallélisme

- SIMD ? MIMD ? vectoriel ? pipeline ?
- niveau nœud ? cœur ? instruction ?

Mémoire

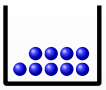
- Partagée ? distribuée ?
- Coût des transferts mémoires ?
- Flexibilité du DMA ?
- Contraintes d'alignement ? de dimension ?
- Taille mémoire ?
- Hiérarchie mémoire ?
- Cache ?
- Présence d'une ROM ?



Contraintes matérielles II

Jeu d'instructions

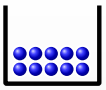
- Calcul flottant ?
- Nombres d'opérandes ?
- Taille des opérandes ?
- Fonctions mathématiques ? trigonométriques ?
- Fonctions spécifiques aux domaine ?
- Registres spécifiques ?
- Opérateurs de flot de contrôle ?
- RISC ? CISC ?
- ...



Transformations de code

Dans la littérature. . .

- Parallélisme
  - Vectorisation de code
  - Détection de parallélisme
  - Extraction de parallélisme
  - . . .
- Mémoire
  - Blocage
  - Extraction de procédure
  - Génération de transferts mémoire
  - . . .
- Jeu d'instructions
  - Détection de motifs
  - Recouvrement de graphe
  - Allocation de registres
  - . . .



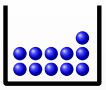
Pièces manquantes

Transformations

- Automatisation de la décision du déport de code sur accélérateur
- Génération optimisée des transferts mémoires
- Agrégation / recouvrement mémoire

Gestionnaire de passes

Enchaînement Programmable $>$ Séquentiel



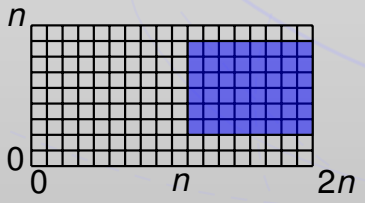
Analyse de régions de tableaux convexes

```
char isme(int n, float a[n][2*n],float b[1+n])
     {
 for(int i=2;i<n-1;i++)
  for(int j=1;j<n;j++)
   a[i][j+n] = j%2 ? 1+a[i][j+n] : b[j];
 return of_the_jedi;
}
```

Analyse de régions de tableaux convexes

```
char isme(int n, float a[n][2*n],float b[1+n])
     {
 for(int i=2;i<n-1;i++)
  for(int j=1;j<n;j++)
   a[i][j+n] = j%2 ? 1+a[i][j+n] : b[j];
 return of_the_jedi;
}
```

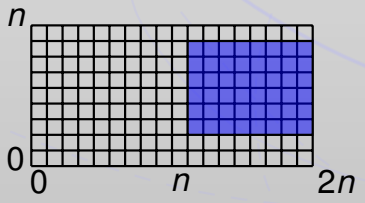
$$\mathcal{R}_w^{exact}(\sigma, a) = \{a[\phi_1][\phi_2] | 2 \leq \phi_1 < n-1, 1+n \leq \phi_2 < 2 \times n\}$$



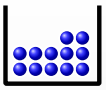
Analyse de régions de tableaux convexes

```
char isme(int n, float a[n][2*n],float b[1+n])
      {
 for(int i=2;i<n-1;i++)
  for(int j=1;j<n;j++)
   a[i][j+n] = j%2 ? 1+a[i][j+n] : b[j];
 return of_the_jedi;
}
```

$$\mathcal{R}_w^{exact}(\sigma, a) = \{a[\phi_1][\phi_2] | 2 \leq \phi_1 < n-1, 1+n \leq \phi_2 < 2 \times n\}$$



$$\mathcal{R}_r^{may}(\sigma, b) = \{b[\phi_1] | 1 \leq \phi_1 < n\}$$



Utilisation pour la génération de code sur machine hétérogènes

Concepts analogues

région $\simeq$ zone mémoire

tableau structure de données de base en HPC

convexe $\simeq$ zone contigüe

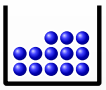
Application

Volume mémoire comptage de points !

Allocation mémoire $\bar{\cup}$ région lues et écrites

Copies vers l'accélérateur régions lues

Copies vers l'hôte régions écrites



Cas pratique: Érosion verticale sur Terapix

```
void runner(int n, int img_out[n-4][n], int
    img[n][n]) {
 for(int y=2;y<n-2;++y)
  for(int x=0;x<n;x++)
   img_out[y-2][x] = MIN(MIN(MIN(MIN(img[y-2][
       x], img[y-1][x]), img[y][x]), img[y+1][x
       ]), img[y+2][x]);
}
```

Ça vous parait facile ?

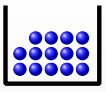
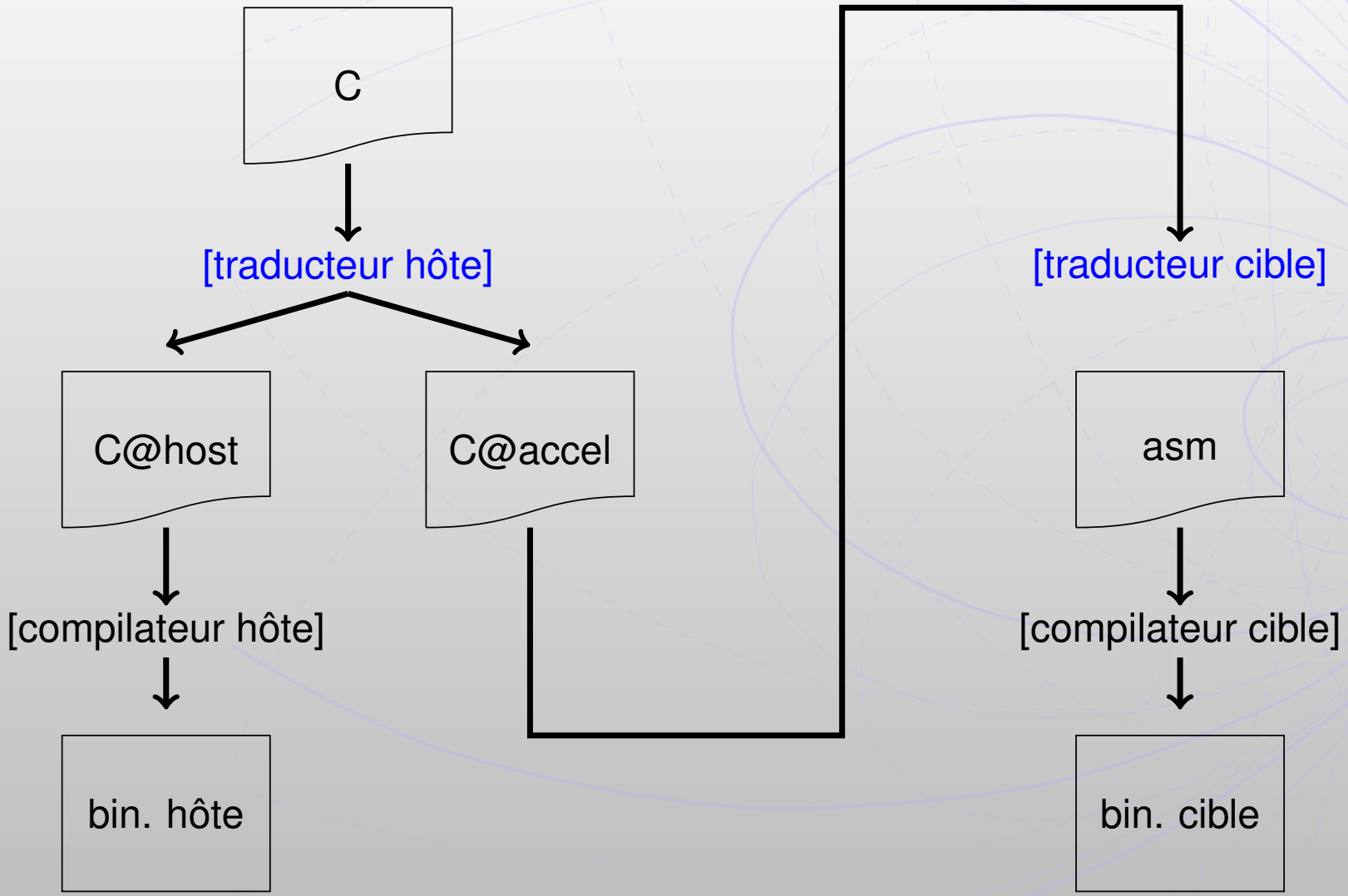


Schéma de compilation

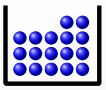
1. Détection du parallélisme
2. Pavage rectangulaire symbolique
3. Estimation de l'empreinte mémoire $\rightarrow$ pavage fixe
4. Régions de tableaux $\rightarrow$ génération de DMA
5. Extraction de procédure $\rightarrow$ génération de noyau
6. « mise aux normes » du noyau (for $\rightarrow$ while, code 2 adresses . . .)
7. Génération de code assembleur

Chaîne de compilation



Code hôte généré

```
for(yt=0;yt<=(n+119)/124-1;yt+=1){
 for(xt=0;xt<=(n+259)/260-1;xt+=1){
  //PIPS generated variable
  int (*img0)[132][260] = (int (*)[132][260]) 0, (*
     img_out0)[128][260] = (int (*)[128][260]) 0;
  P4A_accel_malloc(&img_out0, 33280*sizeof(int));
  P4A_accel_malloc(&img0, 34320*sizeof(int));
  P4A_copy_to_accel_2d(sizeof(int), MIN((131*n-131)
     /4, n+126)+1, MIN((263*n-263)/4, n+258)+1,
     132, 260, 124*yt, 260*xt, &img[0][0], *img0);
  launcher_0(259+1, *img0, *img_out0);
  P4A_copy_from_accel_2d(sizeof(int), n+123, n+259,
      128, 260, 124*yt, 260*xt, &img_out[0][0], *
     img_out0);
  P4A_accel_free(img_out0);
  P4A_accel_free(img0);
 }
}
```

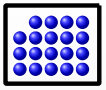


Code accélérateur généré

```
void launcher_0_microcode(int y, int l_3, int
    *img0, int *img_out00)
{
 //PIPS generated variable
 int x;
 //PIPS generated variable
 int *img_out000;
 img_out000 = img_out00;
 for(x = 0; x <= l_3-1; x += 1) {
  *img_out000 = MIN(MIN(MIN(MIN(*(img0+x+260*y
      ), *(img0+x+260*y+260)), *(img0+x+260*y
      +520)), *(img0+x+260*y+780)), *(img0+x
      +260*y+1040));
  img_out000 += 1;
 }
}
```

Code assembleur généré (non compacté...)

```
sub launcher_0_microcode
im7 = FIFO1       ||   ||        ||                    ||
im6 = FIFO0       ||   ||        ||                    ||
im5 = im7         ||   ||        ||                    ||
                  ||   ||        ||      do_N0         ||
                  ||   || re16 = 260 ...
                  ||   || re16 = re16*y ...
                  ||   || re15 = re17 ...
                  ||   || re15 = re15+re16 ...
im4 = im6         ||   ||
im4 = im4+re15    ||   ||
```



Conclusion

Rappel du problème

Comment assembler à moindre coût un compilateur pour un accélérateur ?

Méthodologie

Boîte à outils Une infrastructure de compilation pour le calcul hétérogène doit fournir des transformations à grain fin pour lever les contraintes matérielles

Analyses Polyédriques Les outils polyhédriques fournissent des analyses fines qui se prêtent bien à la génération de code pour accélérateurs matériels

Pragmatique Utiliser les outils existants / fournis par le vendeur