THE UNIVERSITY of EDINBURGH
informatics

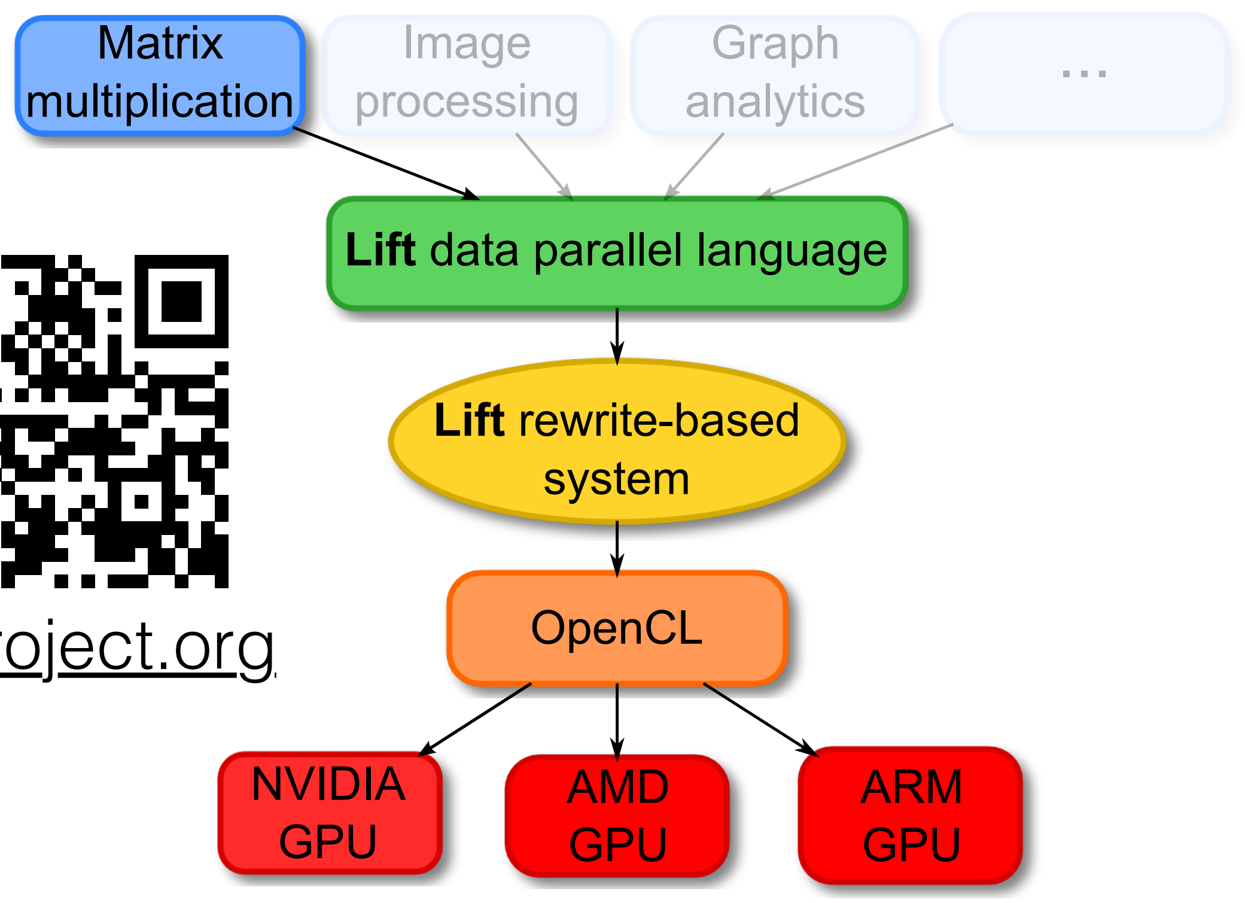
icsa | Institute for Computing Systems Architecture

Matrix Multiplication Beyond Auto-Tuning: Rewrite Based GPU Code Generation

Michel Steuwer, Toomas Remmelg, Christophe Dubach

{michel.steuwer, toomas.remmelg, christophe.dubach}@ed.ac.uk

Lift is a novel approach to achieve high performance on parallel accelerators



lift-project.org

Starting from a *high-level language*, *rewrite rules* are used to derive *optimised implementations.*

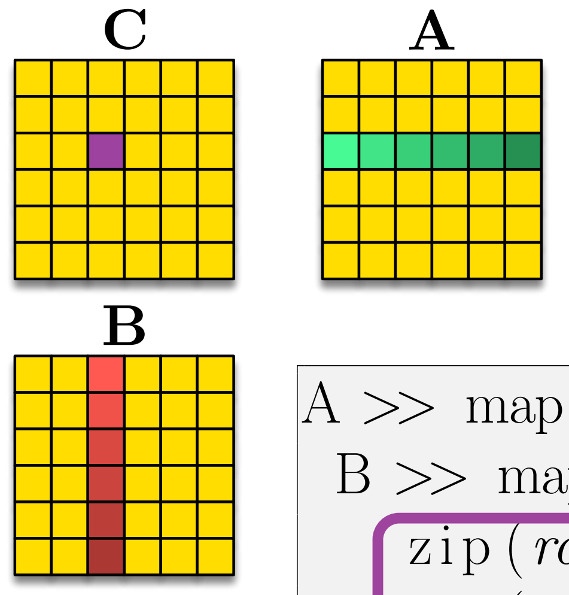
Goal: Achieving *Performance Portability*, *i.e.* high performance across different GPUs

Inherent Limitation of *Auto-Tuning*

Chooses between a fixed number of optimisations and tuning parameters

⇒ Falls short on new architectures! Performance portability cannot be achieved by only using auto-tuning

Matrix Multiplication in the **Lift** Data Parallel Language



Starting from a portable high-level program:

Functional Lift Code

```
A >> map(λ rowOfA ↦
  B >> map(λ colOfB ↦
    zip(rowOfA, colOfB) >>
    map(mult) >> reduce(0.0f,add) ))
```

Lift Rewrite-Based System:

Compilation and Optimisation Using Provably Correct Rewrite Rules

Algorithmic rewrite rules express optimisation choices

Split-join rule

```
map(f)  ⇒  split(k) >> map(map(f)) >> join
```

Functional Lift Code

```
A >> split(m) >> map(λ rowsOfA ↦
  rowsOfA >> map(λ rowOfA ↦
    B >> map(λ colOfB ↦
      zip(rowOfA, colOfB) >>
      map(mult) >> reduce(0.0f,add) ))
) >> join
```

Imperative OpenCL Code

```
for (int i = 0; i<M/2; i++) {
  for (int l = 0; l<2; l++) {
    for (int j = 0; j<N; j++) {
      for (int k = 0; k<K; k++) {
        temp[k + 2*K*N*i + K*N*l + K*j] =
          mult(A[k + K*l + 2*K*i], B[k + K*j]);
      }
      for (int k = 0;k<K;k++) {
        C[j + N*l + 2*N*i] +=
          temp[k + 2*K*N*i + K*N*l + K*j];
      }
    }
  }
}
```

Map-map interchange rule

```
X >> map(λ x ↦ Y >> map(λ y ↦ f))
                  ⇒
Y >> map(λ y ↦ X >> map(λ x ↦ f)) >> transpose
```

Functional Lift Code

```
A >> split(m) >> map(λ rowsOfA ↦
  B >> map(λ colOfB ↦
  rowsOfA >> map(λ rowOfA ↦
    zip(rowOfA, colOfB) >>
    map(mult) >> reduce(0.0f,add) )
  ) >> transpose
) >> join
```

Imperative OpenCL Code

```
for (int i = 0; i<M/2; i++) {
  for (int j = 0; j<N; j++) {
    for (int l = 0; l<2; l++) {
      for (int k = 0; k<K; k++) {
        temp[k + 2*K*N*i + K*N*l + K*j] =
          mult(A[k + K*l + 2*K*i], B[k + K*j]);
      }
      for (int k = 0;k<K;k++) {
        C[j + N*l + 2*N*i] +=
          temp[k + 2*K*N*i + K*N*l + K*j];
      }
    }
  }
}
```

More algorithmic rules

...

OpenCL specific rules express *mapping choices*

OpenCL specific rules

```
map(f)  ⇒  mapGlb{0,1,2}(f)
map(f)  ⇒  mapSeq(f)
f  ⇒  toGlobal(f)
f  ⇒  toPrivate(f)
map(f)  ⇒  asVector(n, b)
    >> map(vectorize(n, f)) >> asScalar
```

Functional Lift Code

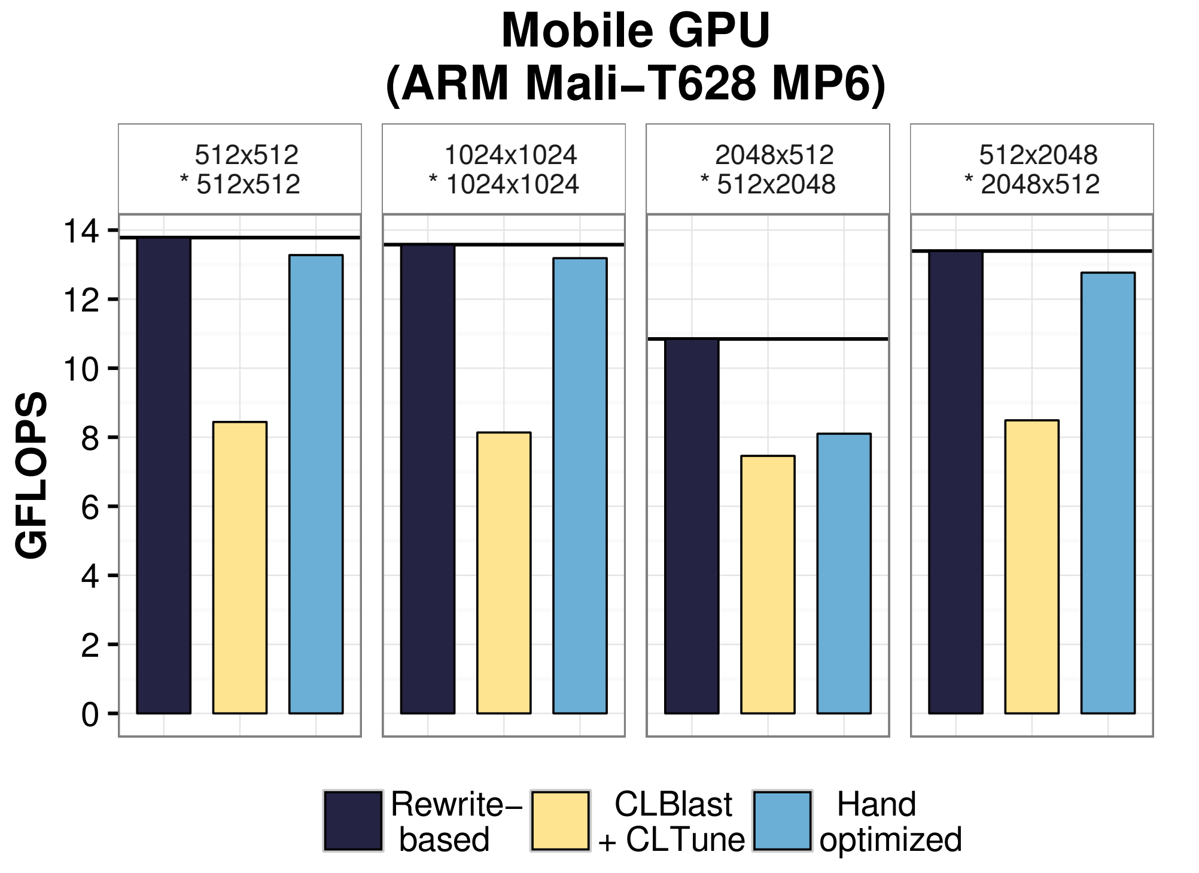
```
A >> split(m) >> mapGlb0(λ nRowsOfA ↦
 B >> split(n) >> mapGlb1(λ mColsOfB ↦
   zip( transpose(nRowsOfA) >> split(k),
        transpose(mColsOfB) >> split(k) ) >>
   reduceSeq(init = make2DArray(n,m, 0.0f) >>
                toPrivate(mapSeq(mapSeq(id))),
    λ (accTile, (tileOfA, tileOfB)) ↦
     zip(accTile, transpose(tileOfA)) >>
    mapSeq(λ (accRow, rowOfTileOfA) ↦
      zip(accRow, transpose(tileOfB)) >>
     mapSeq(λ (acc, colOfTileOfB) ↦
      zip(rowOfTileOfA >> asVector(k),
          colOfTileOfB >> asVector(k)) >>
          mapSeq(dot) >> reduceSeq(acc, add)
      ) >> join
     )
   ) >> toGlobal(mapSeq(mapSeq(mapSeq(id)))
   >> transpose() >>
   map(transpose) >> transpose
 ) >> join >> transpose
) >> join
```

Highly-Optimised Code for the Embedded ARM Mali GPU

Imperative OpenCL Code

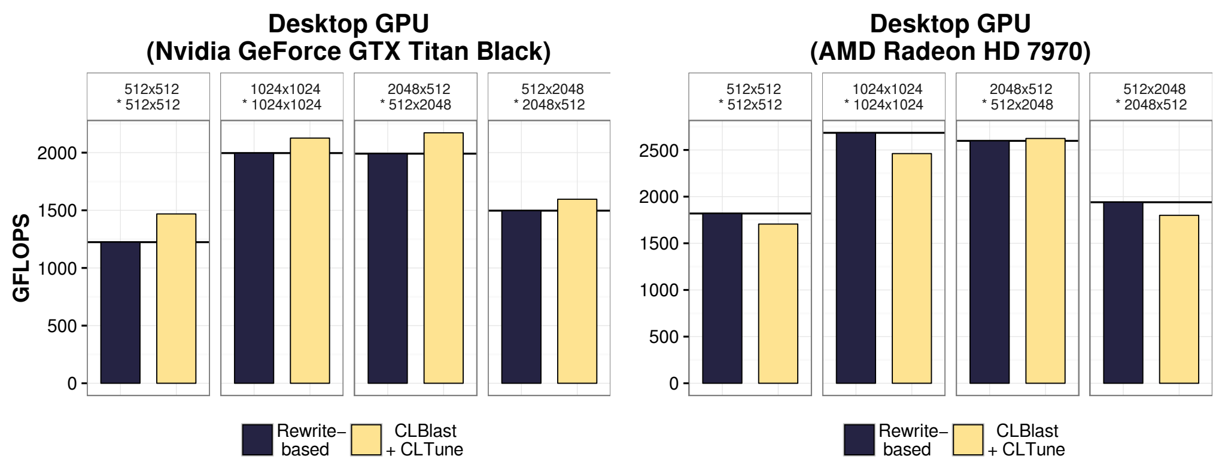
```
int i = get_global_id(0);
int j = get_global_id(1);

float temp_0; float temp_1;
float temp_2; float temp_3;
float acc_0;  float acc_1;
float acc_2;  float acc_3;

for (int k = 0;k<K/4; k++) {

  temp_0 = dot(vload4(k + K*i/2, A),
    vload4(k + K*j/2, B));
  acc_0 += temp_0;

  temp_1 = dot(vload4(k + K*i/2, A),
    vload4(k + K + 2*K*j/4, B));
  acc_1 += temp_1;

  temp_2 = dot(vload4(k + K + 2*K*i/4, A),
    vload4(k + K*j/2, B));
  acc_2 += temp_2;

  temp_3 = dot(vload4(k + K + 2*K*i/4, A),
    vload4(k + K + 2*K*j/4, B));
  acc_3 += temp_3;
}
C[2*N*i + 2*j] = id(acc_0);
C[1 + 2*N*i + 2*j] = id(acc_1);
C[N + 2*N*i + 2*j] = id(acc_2);
C[1 + N + 2*N*i + 2*j] = id(acc_3);
```

High Performance on Mali



Performance Portability

Automatic Exploration Using the **Lift** Rewrite-Based System



Supported by: Google