Automatic Source-to-Source Code Generation for Vector Hardware Accelerators

Serge Guelton, serge.guelton@telecom-bretagne.eu

GDR SOC-SIP – 9-10-11 June 2010

TELECOM Bretagne

From C or FORTRAN legacy code...

- Widely spread in scientific community
- High level languages
- Easy to read and to maintain

...to hardware-accelerated assembly

- Parallel vector processing
- Real-time throughput
- Lower energy consumption

One Challenging Target: the Ter@pix Processor

- Limited Memory Size
- Fixed-size transfers
- 2D memory layout
- Integers only
- No division
- No constant values
- Hard-coded loops
- Limited control flow
- ROM & RAM access
- Vector loops
- VLIW instructions
- No masked execution

Source-to-source code transformations

0 Initial Code

```
void alphablending(short src0[512][512], short src1
  [512][512], short result[512][512])
{
  unsigned int i, j;
  for(i = 0; i <= 511; i += 1)
    for(j = 0; j <= 511; j += 1)
      result[i][j] = (40*src0[i][j]+60*src1[i][j])/100;
}
```

- instruction selection
- atomizer, unrolling, pointer conversion ...
- THALES compactor

1 Derived Host Call

```
for(i_t=0;i<64;i+=1)
 for(j_t=0;j_t<4;j_t+=1) {
  int src00[128][8],src10[...],CST0[4],result0[128][8];
  load_image(src10,src1,...);load_image(CST0,CST,...);
  load_image(src00,src0,...);
  launcher_0(i_t,CST10,result0,src00,src10);
  store_image(result,result0,...);}
```

- division removal
- tiling
- statement isolation
- constant grouping

2 Derived Vector Loop Emulator

```
void launcher_0(int I_0, int CST0[4], short result0
  [128][8], short src00[128][8], short src10[128][8])
{
  //PIPS generated variable
  unsigned int i, j;
  for(i = 1; i <= 128; i += 1)
    launcher_0_microcode(I_0, CST0, &result0[-1+i], &
      src00[-1+i], &src10[-1+i]);
}
```

- parallelism detection
- outlining
- loop expansion
- constant region
- outlining

3 Derived Accelerator Code

```
void launcher_0_microcode(int I_0, int *CST0, short **
  result00, short **src000, short **src100)
{
   //PIPS generated variable
   unsigned int j;
   for(j = 0; j <= -1+I_0; j += 1)
      *(*result00+j+1-1) = (CST0[2]**(*src000+j+1-1)+
        CST0[3]**(*src100+j+1-1))*(CST0[0]<<CST0[1])>>
        CST0[1];
}
```

4 Derived Assembly

```
sub launcher_microcode_0
 im,im7=FIFO2 ||  ||                          ||       ||
 im,im5=FIFO1 ||  ||                          ||       ||
...
              ||  ||                          || do_N1 ||
              ||  || P,re(21)=re(21)*re(30) ||       ||
...
endsub
```

PIPS4U

Implemented in the PIPS Infrastructure

python **compiler assembler**

PIPS4U inter-procedural Analysis

array region analysis

Open Source Software from CRI/MINES ParisTech

pipsdev@cri.mines-paristech.fr

memory effect computations

array privatization

Funded by the ANR FREIA

THALES Code Parallelizer

Serge Guelton [Télécom Bretagne] – François Irigoin [Mines ParisTech] – Ronan Keryell [HPC Project]