Register allocation for iVMX architecture

Mircea Namolaru
IBM Haifa Research Labs
namolaru@il.ibm.com

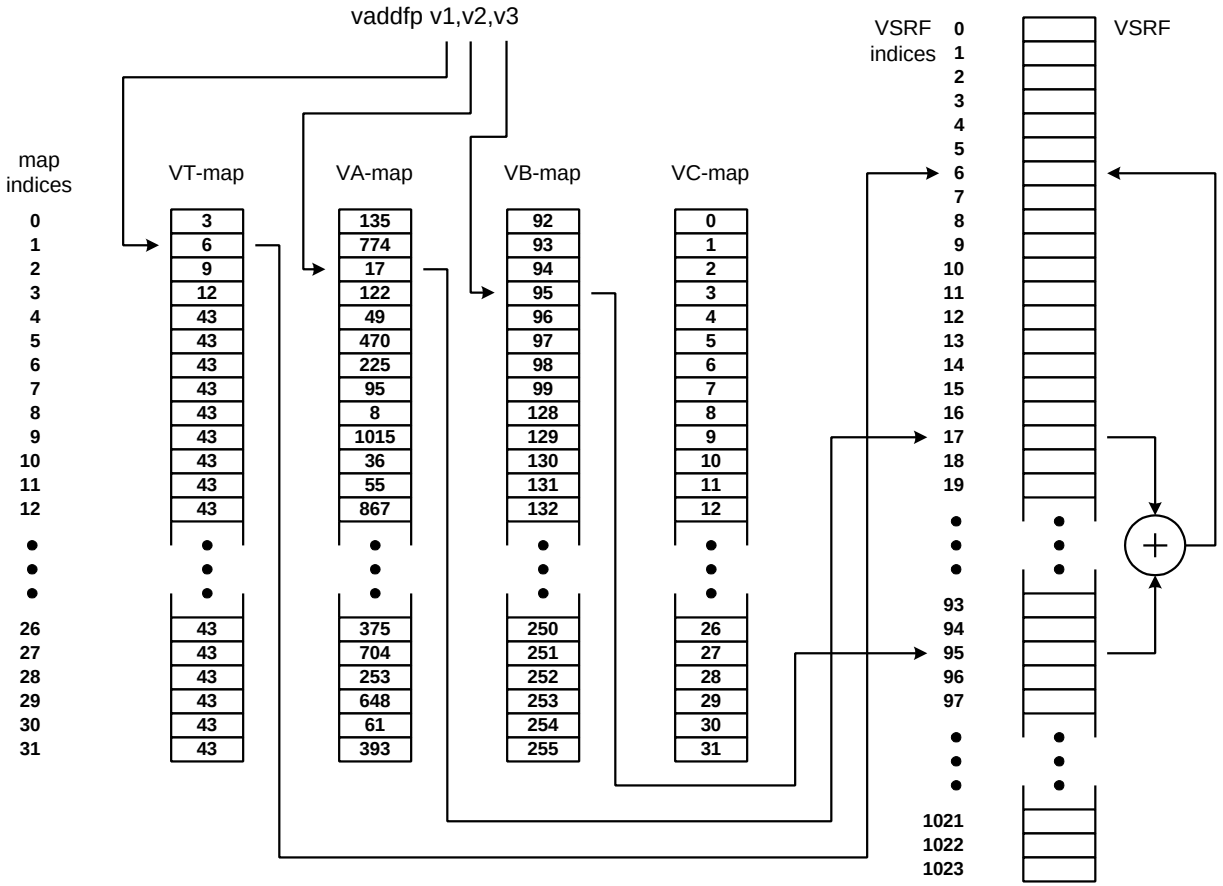
Talk Layout

- iVMX architecture
- Two techniques for register mapping
- Special subsets of live ranges and register allocation and mapping for them
- Conclusions and future work

iVMX architecture

- J. H. Derby, R. K. Montoye, J. Moreira: VICTORIA: VMX indirect compute technology oriented towards in-line acceleration. Conf. Computing Frontiers 2006.
- Extension to VMX architecture – compatible with it
- Very large register file - 1024 vector registers (16 byte each)
- Mapping mechanism
  - Each operand has a type (T, A, B, C)
  - An operand is an index in the range [0, 31] that is mapped to the register range [0, 1023]
  - For each operand type, the mapping is done via four 16 byte mapping register (16 byte each), each register being responsible for mapping eight indexes
  - Default mapping
- Mapping instructions
  - Set map from register - mtmr mr, vreg
  - Set map immediate - setmr mr, const

iVMX Indirection Mechanism – An Example



Why GCC ?

- Develop register allocation and mapping for iVMX architecture
- GCC compiler
  - Retargetable
  - Support for VMX architecture
  - Sources are available
  - Automatic vectorization
  - Good infrastructure available for register allocation

FIR kernel

```
for (iiv=0; iiv<M; iiv+=1){
   vy0 = vzero; vx0 = vx[iiv];
   for (jv=0; jv<N; jv++){
          vcj = vc[jv];
          vx1 = vx[iiv+jv+1];
          vy0 = vec_madd(vec_splat(vcj,0), vx0,vy0);
          vy0 = vec_madd(vec_splat(vcj,1), vec_sld(vx0,vx1,4),vy0);
          vy0 = vec_madd(vec_splat(vcj,2), vec_sld(vx0,vx1,8),vy0);
          vy0 = vec_madd(vec_splat(vcj,3), vec_sld(vx0,vx1,12),vy0);
          vx0 = vx1;}
          vy[iiv] = vy0;}
   }
```

M = 16, N = 10, completely unrolled – 1650 spill instructions

First technique

- Basic block mapping register allocation
- Use the regular register allocation with 1024 registers available
- Additional pass for register mapping invoked after register allocation
- Mapping done for each operand type separately
- Basic block divided in regions – in each region there are occurrences of 32 different registers (except the last region that have as boundary the end of the basic block)
- At the start of each region load the 32 registers in the four map registers available
- lazy, wait until current maps are insufficient, then swap it with new maps for next 32 needed registers

First techniques - Results

- Usually the registers in the maps found are not consecutive and set map instruction should be used to load them in the map registers. This requires the loading of the maps (vector constants) in vector registers.
- FIR: 450 set map instructions (+ 900 instructions for loading the maps in vector registers)

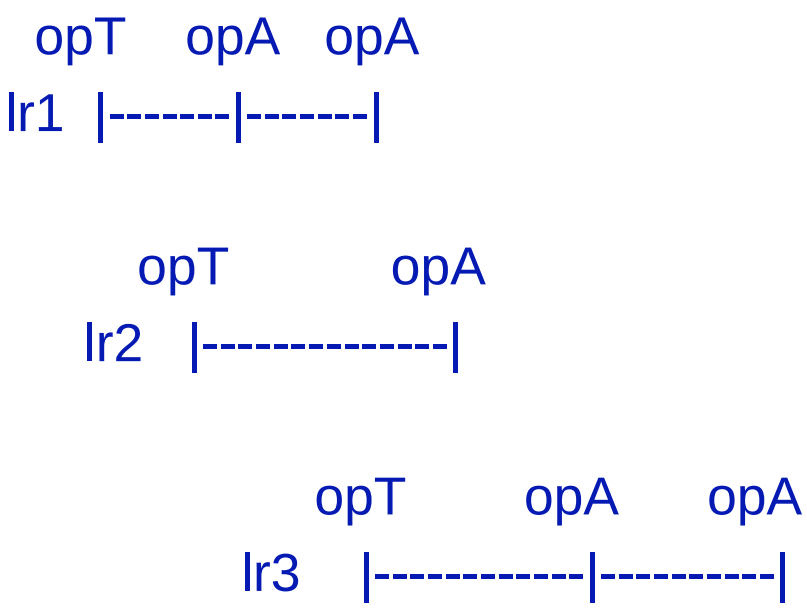
Second technique

- Use the regular register allocation with 1024 registers available
- For an occurrence of register I, the map [I - (I mod 8), I - (I mod 8)+7] is used
- If I is not in a map register, find a map register where to load the map that includes I
- Lazy load of maps
- If not register map available, we need to decide what register map to reset:
  - Always reset the first map register
  - Lookahead and reset the map register with the furthest uses
- Allow the use of set map immediate instructions
- A map loaded in a register may have unused entries
- FIR: 400 instructions (second heuristic), 900 (first heuristic)

Chains

- Compute the live ranges – we may undo the decision made by the regular GCC register allocation and perform our own register allocation.
- Identify special subsets of live ranges that allows efficient register allocation and mapping
- Chains - sequence of live ranges {lr1, lr2 …., lrn} such as for every two consecutive live ranges lri and lrj:
  - lri and lrj infer
  - lir and lrj have the same operands type
  - the occurrences of lri as operand of a given type, occur before the occurrences of lrj as operand of the same type
- Less restrictive definitions possible (not needed for FIR)

Chain example



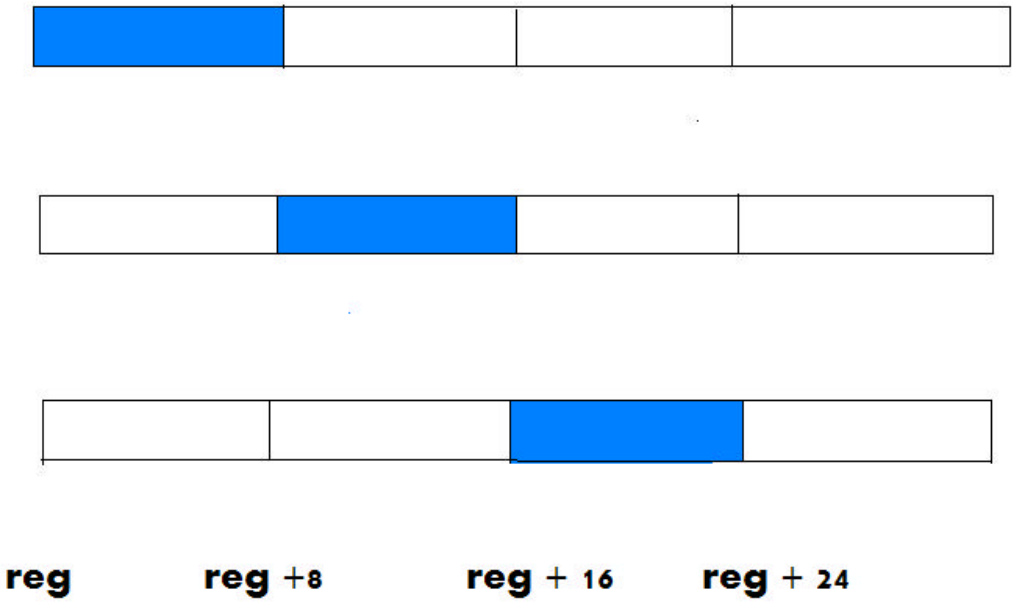
Chain – code example

```
vspltw  r1,r20,1
vspltw  r2,r26,1
vspltw  r3,r21,1
……..
vmaddfp r17,r1,r21,r8
vmaddfp r19,r1,r24,r9
……..
vmaddfp r18,r2,r21,r8
……..
vmaddfp r19,r3,r6,r8
vmaddfp r19,r3,r4,r9
```

Register allocation for a chain

- For a chain {lr1, lr2, ….., lrn} allocate a block of consecutive registers starting at register reg
- The i-entry live range in the chain receives the register reg + i
- Consider the maps mi = [reg + 8 * i, reg + 8 *i + 7]. For a chain of length n, we need n div 8 (+ 1 if n mod 8 != 0) such maps to cover the register in the block allocated for the chain
- When a new map is required, the previous map is no longer needed, so the map register could be used
- The loading of maps could be done by using set map immediate instructions
- If a map register is available for the chain, any map is loaded only once. For a chain of length n, the loading of the maps requires n div 8 (+1 if n mod 8 != 0) set map immediate instructions
- Efficient mechanism for register allocation in regions where the high level register pressure is due mainly to inferences of ranges from the same chains.

Register allocation for chains (2)



FIR

- For FIR completely unrolled there are 16 chains of length 40 and 16 chains of length 30.
- In the point of maximal register pressure there are 52 inferring ranges, where 24 ranges are part of the first chain, and 13 ranges part of the second chain
- Consecutive block of register allocated for chains, and a register map for each operand type.
- The number of set map immediate instructions comes down to less then 300, when the dedicated blocks of registers and map registers for chains are used
- Default mapping is not assumed

FIR – register mapping

range [0, 15]:not in chains
- type T: use map registers mT1. mT2
- type A: use map registers mA1. mA2
- type B: use map registers mB1, mB2
- type C: use map registers mC1, mC2

range [240, 280]:chain1
- type T: use map registers mT3
- type A: use map register mA3

range [320, 350]:chain2
- type T: use map register mT4
- type B: use map register mB4

Conclusions

- The register allocation for iVMX is challenging
- We moved from the paradigm of allocating a register to a live range, to the paradigm of allocating blocks of registers to subsets of live ranges (chains)
- We identified patterns of live ranges (chains) typical for code where the register pressure it due to overlapping of instructions from different iterations (as after modulo-scheduling, unrolling + scheduling etc).