REGIONALES HOCHSCHUL-
RECHENZENTRUM KAISERSLAUTERN

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

RHRK-Tutorial

Vectorization: AVX512 extensions

Course instructor: Gabriele Monaco
In charge: Dr. Josef Schüle, RHRK

AVX512 extensions: AVX-512CD

- **Conflict Detection: available in Skylake processors, allows the vectorization of codes with indirect memory accesses leading to write conflicts**
  - A block of values from B is loaded into a vector register (vmovups)
  - A conflict detection instruction (vpconflictd) identifies entries with identical values.
  - If there are no conflicts the code jumps to block B1.6.
  - Possible conflicts are handled in block B1.4.
  - Non conflicting values of C are gathered (vgatherdps) and the resulting vector A scattered to memory (vscatterdps)

```
float A[SIZE],C[SIZE];
int B[SIZE],i;
for(i=0;i<SIZE;i++)
    A[B[i]] += C[i];
```

```
..B1.2:
  vmovups    64000(%rsp,%rdx,4), %zmm6
  vpconflictd %zmm6, %zmm9
  vptestmd   .L_2il0floatpacket.1(%rip),
%zmm9, %k0
  je         ..B1.6
..B1.4:
  vpermps    %zmm8, %zmm3, %zmm9{%k2}
  vaddps     %zmm9, %zmm8, %zmm8{%k2}
  jne        ..B1.4
..B1.6:
  vgatherdps (%rax,%zmm3,4), %zmm9{%k2}
  vaddps     %zmm9, %zmm8, %zmm8
  vscatterdps %zmm8, (%rax,%zmm6,4){%k3}
```

AVX512 extensions: AVX-512F

- **Masking: sets a bitmask to vectorize loops with if statements**
  - B is loaded into a ZMM register afterwards A (first 4 lines)
  - With vcmpps the if statement is mapped in mask registers (k2 and k4) and their logical AND results (knotw)
  - B is copied with the AND-masks (movaps) and added to A (vaddps)
  - The other B values are multiplied with corresponding values according to the mask registers k2 and k4 (vmulps) before the results are stored completely (vmovups)
  - The loop is unrolled with a factor of 2, so all instructions appear twice.

```
float A[SIZE],B[SIZE];
int i;
for(i=0;i<SIZE;i++) {
  if(B[i] > 0.0)
     A[i] *= B[i];
  else
     A[i] += B[i];
}
```

```
vmovups    (%rsp,%rax,4), %zmm3
vmovups    64(%rsp,%rax,4), %zmm7
vmovups    32000(%rsp,%rax,4), %zmm2
vmovups    32064(%rsp,%rax,4), %zmm6
vcmpps     $14, %zmm0, %zmm3, %k2
vcmpps     $14, %zmm0, %zmm7, %k4
knotw      %k2, %k1
knotw      %k4, %k3
vmovaps    %zmm3, %zmm1{%k1}{z}
vmovaps    %zmm7, %zmm5{%k3}{z}
vaddps     %zmm1, %zmm2, %zmm4
vaddps     %zmm5, %zmm6, %zmm8
vmulps     %zmm3, %zmm2, %zmm4{%k2}
vmulps     %zmm7, %zmm6, %zmm8{%k4}
vmovups    %zmm4, 32000(%rsp,%rax,4)
vmovups    %zmm8, 32064(%rsp,%rax,4)
```

AVX512 extensions: AVX-512F

- **Compress/Expand: allows to compress and expand instructions to AVX2. vcompress is used in the following example**
  - B is loaded (vmovups) and compared into a mask register (k1)
  - Guided by this mask the valid values of B are compressed into another ZMM register (vcompressps) and the contents finally written to memory
  - The predicted speedup amounts to a factor of 18 and the loop will be unrolled to a factor of 4

```
float A[SIZE],B[SIZE];
int i,j=0;
for(i=0;i<SIZE;i++) {
  if(B[i] != 0.0)
     A[j++] =B[i];
}
```

```
vmovups      32000(%rsp,%rcx,4), %zmm2
vcmpps       $4, %zmm0, %zmm2, %k1
vcompressps  %zmm2, %zmm1{%k1}
vmovups      %zmm1, (%rsp,%rax,4){%k1}
```

AVX512 extensions: AVX-512F

- **Permutation or Shuffle: allows to rearrange elements in one or two source registers and write them to the destination register**
  - Can be used for transpositions like A[i][j] = B[j][i]
  - Not really effective, output is complex
- **Gather/Scatter: to vectorize non-unit strided access to arrays**
  - The compiler may decide whether to use it or not based on the value of INC (if known).
  - With low values (2) it may prefer to use a sequence of shuffle and expand
  - Allows the possibility to vectorize more complicated loops using few registers

```
for(i=0;i<SIZE/INC;i++)
    A[i*INC] += C[i];
```

AVX512 extensions: AVX-512F

- Puts 2 words of A onto one cache line
- Following a mask register 8 floats are read from cache into one ZMM registers half filled
- 2 half filled ZMM registers are combined (vpermt2ps) and the operands C are fetched from memory and added (vaddps)
- Half of the values are expanded into a new register (vexpandps) and the other half permuted into another register (ZMM8)
- Then both stored to 4 cache lines each
- Note the amount of registers required here: the vector masks ZMM1 and ZMM0 additional 5 registers are needed

```
vpermt2ps   %zmm2, %zmm1, %zmm4
vaddps      32000(%rsp,%rax,4), %zmm4, %zmm7
vexpandps   %zmm7, %zmm6{%k1}{z}
vpermps     %zmm7, %zmm0, %zmm8
```

- The data is first gathered directly from cache into one register (ZMM1)
- Then 16 32-bit floats added and according to the information in ZMM0, scattered to memory (vscatterdps)
- In this case a total of just 3 vector registers are involved, even though the compiler might prefer the other version

```
for(i=0;i<SIZE/INC;i++)
    A[i*INC] += C[i];
```

```
vgatherdps   (%rax,%zmm0,4), %zmm1{%k1}
vaddps       32000(%rsp,%rdx,4), %zmm1, %zmm3
vscatterdps  %zmm3, (%rax,%zmm0,4){%k3}
```

AVX512 extensions: AVX-512F

- **Embedded Broadcasting: useful for matrix multiplications**
  - The innermost loop performs an update of one line of A where B[i*SIZE+j] is constant for the whole loop
  - The constant B[i*SIZE+j] is loaded once and broadcasted to fill register ZMM0 before the registers ZMM1-4 are filled with values of A (vmovups)
  - Finally C is loaded, multiplied with B in ZMM0, added and stored to ZMM1-4 in a single fused multiply and add instruction (vfmadd213ps)

```
for(i=0;i<SIZE;i++)
  for(j=0;j<SIZE;j++)
    for(k=0;k<SIZE;k++)
      A[i*SIZE+k] +=
B[i*SIZE+j] * C[j*SIZE+k];
```

```
vbroadcastss (%rdi,%r8,4), %zmm0
vmovups      512000(%rsp,%r10,4), %zmm1
vmovups      512064(%rsp,%r10,4), %zmm2
vmovups      512128(%rsp,%r10,4), %zmm3
vmovups      512192(%rsp,%r10,4), %zmm4
vfmadd213ps  256000(%rsp,%r9,4), %zmm0, %zmm1
vfmadd213ps  256064(%rsp,%r9,4), %zmm0, %zmm2
vfmadd213ps  256128(%rsp,%r9,4), %zmm0, %zmm3
vfmadd213ps  256192(%rsp,%r9,4), %zmm0, %zmm4
```

AVX512 extensions: others

- **AVX-512DQ: Double and Quad Words**
  - Available in Skylake CPUs to convert integers to floating point numbers in 512-bit vector registers. Not available for Knights Landing.
- **AVX-512BW: Byte and Word Support**
  - Supports byte- or word-size data elements. Not available for Knights Landing.
- **AVX-512VL: Vector Length**
  - Allows 512-bit instructions to operate on 128-bit (XMM) and 256-bit (YMM). Useful if the amount of data parallelism cannot fill the 512-bit registers. Not available in Knights Landing.
- **AVX-512ER: Exponential and Reciprocal**
  - Available for Knights Landing only. Adds vector versions for base-2 exponential, reciprocal and reciprocal square root at high accuracy.
- **AVX-512PF: Prefetch for Gather/Scatter**
  - Available in Knights Landing only. Adds the possibility to prefetch up to 16 elements of scattered data in memory, it's useful for code using indirect addressing.

- **High Performance Computing on Elwetritsch**

Vielen Dank
Thank You