Dynamic SIMD Scheduling

Florian Wende

SC15 MIC Tuning BoF

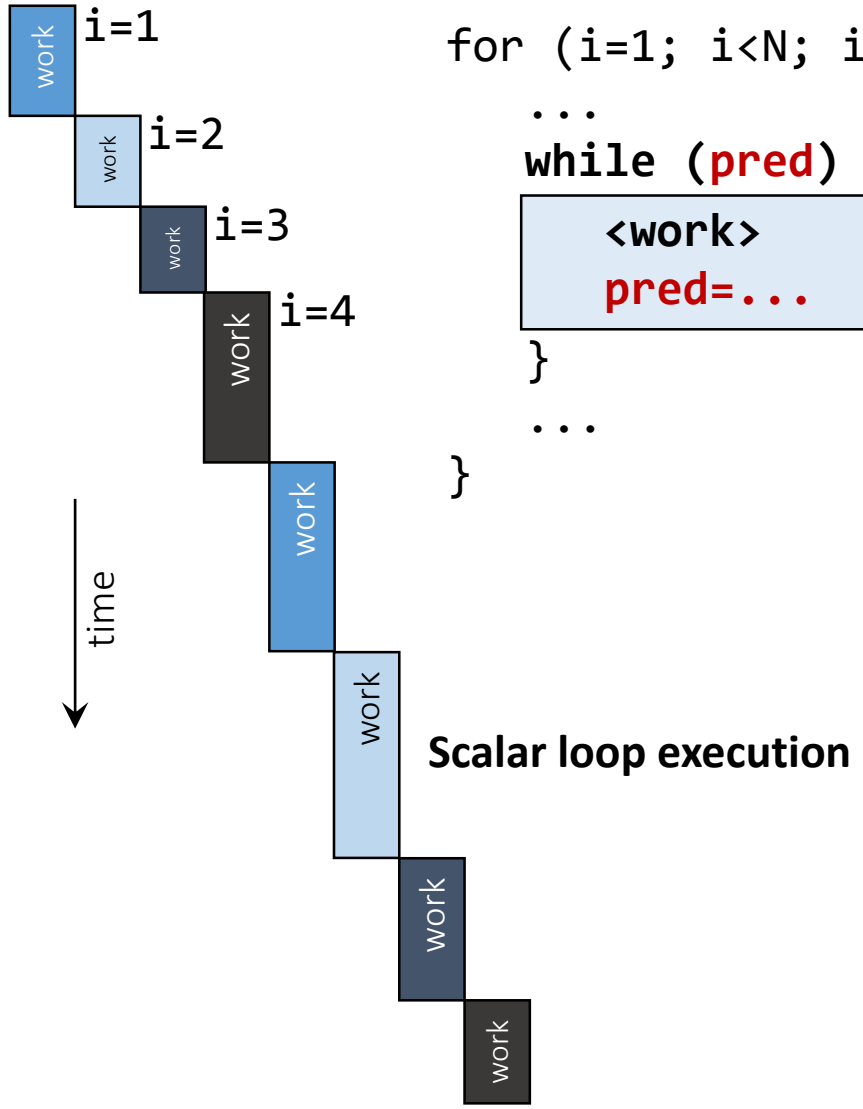
November 18th, 2015

Zuse Institute Berlin

Dynamic Work Assignment: The Idea

Irregular SIMD execution

- Caused by branching: control flow varies across SIMD lanes
- SIMD lanes run out of work even though there is still work to do (next iteration)

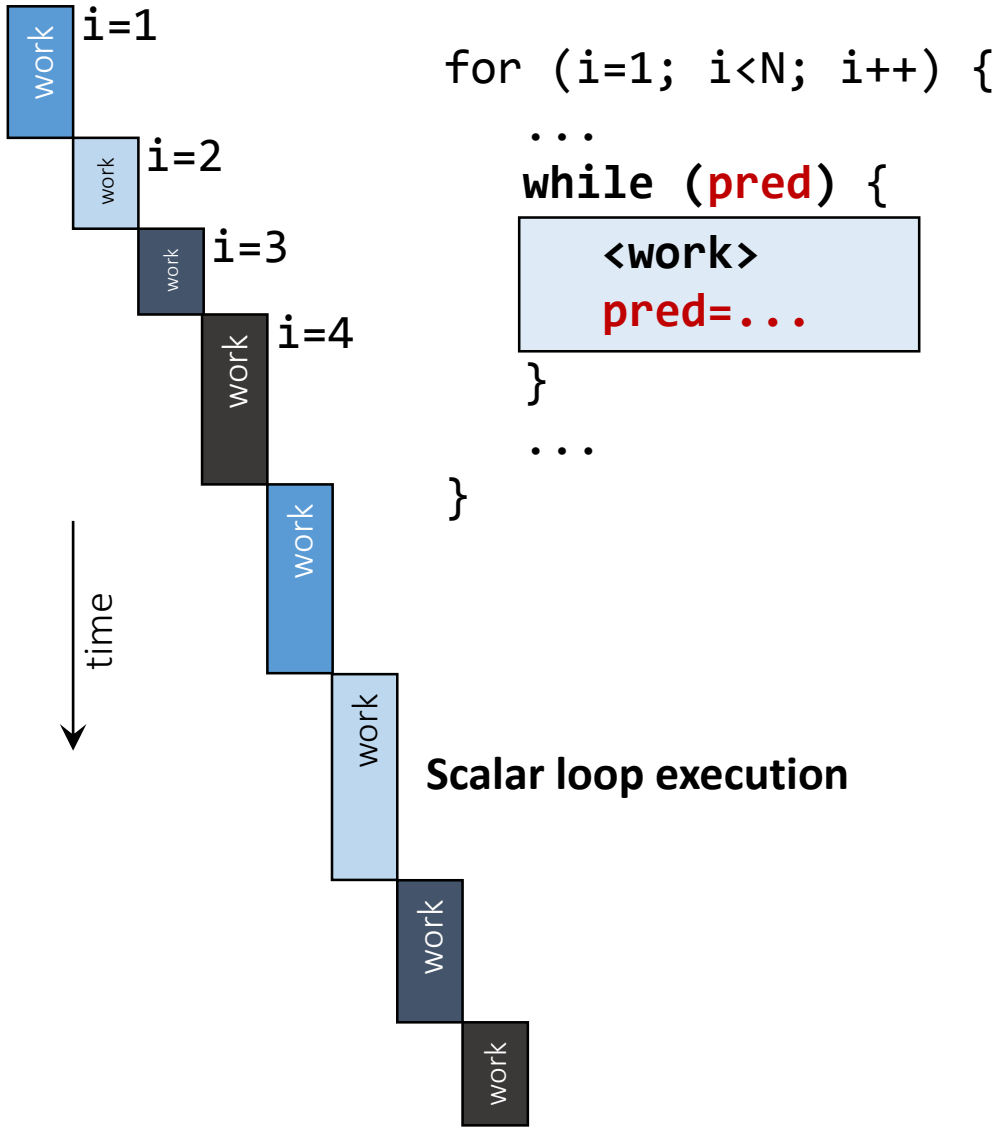


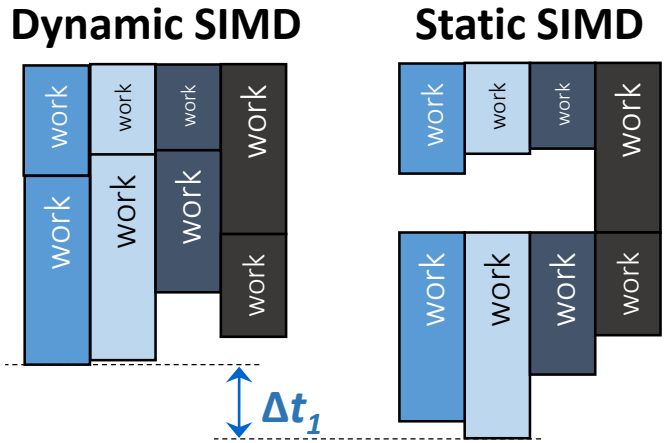

```
for (i=1; i<N; i++) {
    ...
    while (pred) {
        <work>
        pred=...
    }
    ...
}
```

Dynamic Work Assignment: The Idea

Irregular SIMD execution

- Caused by branching: control flow varies across SIMD lanes
- SIMD lanes run out of work even though there is still work to do (next iteration)
- Can we pre-schedule successive iterations on idle SIMD lanes (dynamically)?





a) **Even partitioning** of all work + pre-scheduling within partitions

b) **Greedy schedule** (not shown here)

Dynamic Work Assignment: Implementation

Dynamic SIMD

```
lane_alife_any=true, acquire_work_any=true
for (ii=0; ii<VL; ii++) // VL: Vector Length
   lane_alife[ii]=true, lane_acquire_work[ii]=true, i[ii]=ii

while (lane_alife_any)
   lane_alife_any=false
   if (acquire_work_any)
```

Initialization

Acquire work

```
   for (ii=0; ii<VL; ii++)
      if (lane_alife[ii])
```

Do work

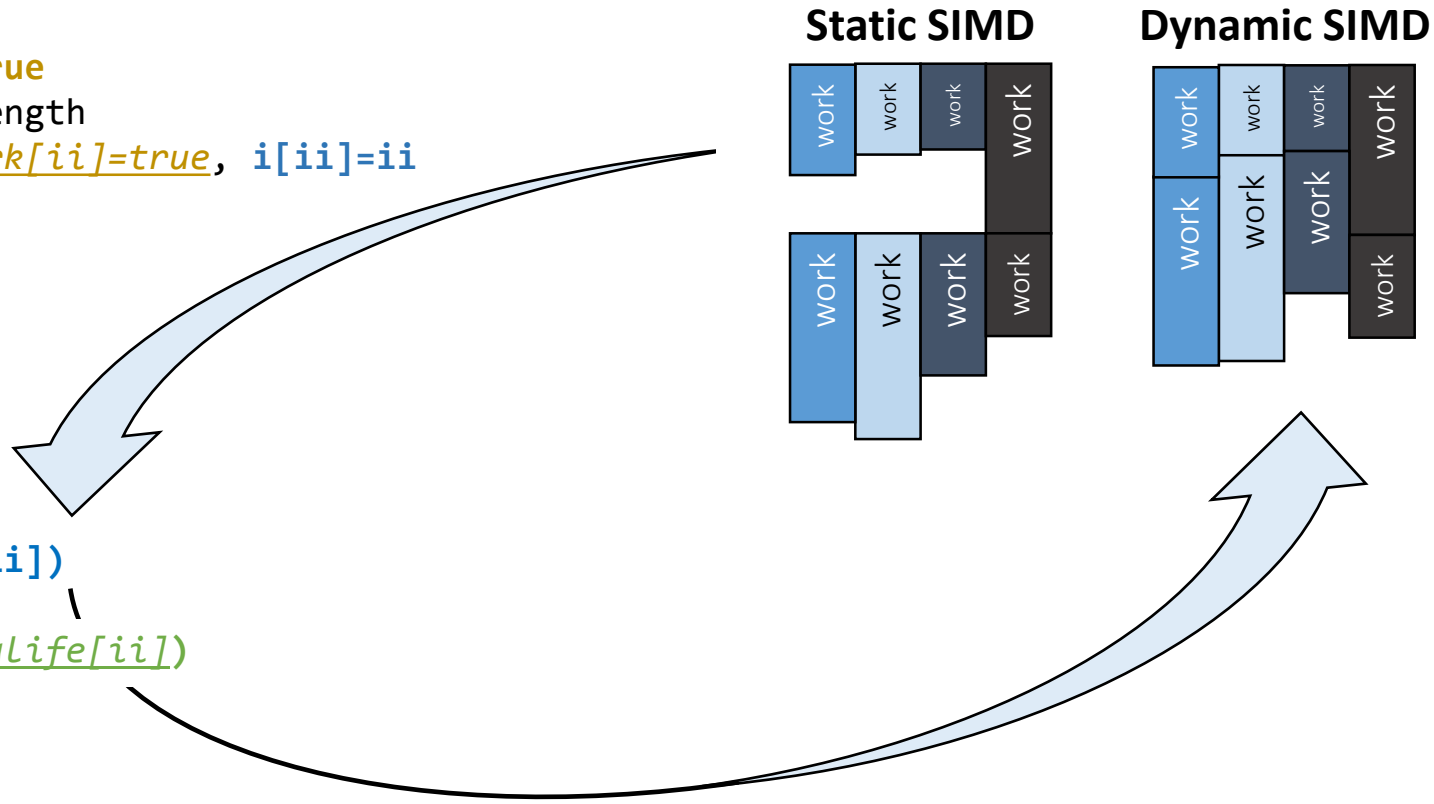
Dynamic Work Assignment: Implementation

Dynamic SIMD

```
lane_alife_any=true, acquire_work_any=true
for (ii=0; ii<VL; ii++) // VL: Vector Length
   lane_alife[ii]=true, lane_acquire_work[ii]=true, i[ii]=ii

while (lane_alife_any)
   lane_alife_any=false
   if (acquire_work_any)
      acquire_work_any=false
      for (ii=0; ii<VL; ii++)
         if (lane_acquire_work[ii])
            lane_acquire_work[ii]=false
            if (i[ii] < N) load_data(i[ii])
            else lane_alife[ii]=false       \
         lane_alife_any=reduce_or(lane_alife[ii])

   for (ii=0; ii<VL; ii++)
      if (lane_alife[ii])
         Do work
```



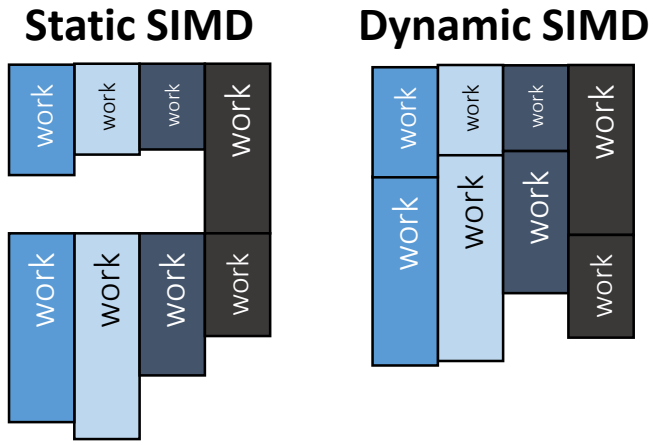
Dynamic Work Assignment: Implementation

Dynamic SIMD

```
lane_alife_any=true, acquire_work_any=true
for (ii=0; ii<VL; ii++) // VL: Vector Length
   lane_alife[ii]=true, lane_acquire_work[ii]=true, i[ii]=ii

while (lane_alife_any)
   lane_alife_any=false
   if (acquire_work_any)
      acquire_work_any=false
      for (ii=0; ii<VL; ii++)
         if (lane_acquire_work[ii])
            lane_acquire_work[ii]=false
            if (i[ii] < N) load_data(i[ii])
            else lane_alife[ii]=false
         lane_alife_any=reduce_or(lane_alife[ii])

   for (ii=0; ii<VL; ii++)
      if (lane_alife[ii])
         <work(i[ii])>
         pred=...
         if (!pred) store_data(i[ii]), i[ii]+=VL, lane_acquire_work[ii]=true
      acquire_work_any=reduce_or(lane_acquire_work[ii])
```



Dynamic Work Assignment: Test Sample

The kernel (synthetic)

Input: `x1[]` uniform over [0.0,2.0]
`x2[]` uniform over [0.0,1.0]
Output: `y[]`

```
#define N (8 * 1024 * 1024)
#define D (50)

for (i=0; i<N; i++)
   int k=0
   int kmax=(int)(D * x2[i])
   double temp_y=0.0
   double temp_x1=x1[i]
   while (k < kmax)
      temp_y=sqrt(temp_x1 + temp_y)
      if (temp_y > 1.0) temp_y=log(temp_y)
      k++
   y[i]=temp_y
```

Number of loop iterations varies from 0 to 50

Dynamic Work Assignment: Test Sample, Performance

Platforms / Software

- ❑ Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- ❑ Xeon Phi 5120D
- ❑ Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

Runtimes for the loop execution in seconds (initial performance)

	Haswell	**Xeon Phi**
*D=50, N=8*1024*1024, x1[] and x2[] at random*	VL=4	VL=8
SIMD Static, Auto-Vectorization	6.75 s	36.0 s
SIMD Static, Explicit Vectorization	4.63 s	11.7 s
SIMD $\text{Dynamic}_{\text{EvenPartitioning}}$, Explicit Vect.	3.90 s	10.1 s

Dynamic Work Assignment: Optimizations

Gaining performance over initial implementation

- Local (private) copies in case of repeated memory accesses
- Replace masked call to `log()` (maybe SVML in general) by unmasked call + conditional assignment

```
if (pred) y=log(...)  ⟶  double temp_y=log(...)
                         if (pred) y=temp_y
```

- `#pragma vector aligned` + `#pragma omp simd` to tell the compiler all memory accesses are aligned: OpenMP 4.0 aligned clause has some limitations
- `safelen(n)` clause with values n > native vector length

Dynamic Work Assignment: Test Sample, Performance

Platforms / Software

- ❑ Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- ❑ Xeon Phi 5120D
- ❑ Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

Runtimes for the loop execution in seconds (final performance)

	Haswell		**Xeon Phi**	
*D=50, N=8*1024*1024, x1[] and x2[] at random*	VL=4	VL=8	VL=8	VL=16
SIMD Static, Auto-Vectorization	6.75 s	-	36.0 s	-
SIMD Static, Explicit Vectorization	**4.63 s**	-	**11.7 s**	-
SIMD Dynamic$_{\text{EvenPartitioning}}$, Explicit Vect.	3.08 s (+26%)	**2.68 s** (+45%)	9.83 s (+3%)	**9.67 s** (+5%)

Fastest version on KNC with Intrinsics: ~8 s

Dynamic Work Assignment: Test Sample, Performance

Platforms / Software

- ❑ Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- ❑ Xeon Phi 5120D
- ❑ Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

Runtimes for the loop execution in seconds (final performance)

	Haswell		**Xeon Phi**	
*D=50, N=8*1024*1024, x1[] and x2[] at random*	VL=4	VL=8	VL=8	VL=16
SIMD Static, Auto-Vectorization	6.75 s	-	36.0 s	-
SIMD Static, Explicit Vectorization	**4.63 s**	-	**11.7 s**	-
SIMD Dynamic$_{\text{EvenPartitioning}}$, Explicit Vect.	3.08 s (+26%)	**2.68 s** (+45%)	9.83 s (+3%)	**9.67 s** (+5%)

static

Fastest version on KNC with Intrinsics: ~8 s

Dynamic Work Assignment: Test Sample, Performance

Platforms / Software

- ❑ Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- ❑ Xeon Phi 5120D
- ❑ Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

Runtimes for the loop execution in seconds (final performance)

	Haswell		Xeon Phi	
*D=50, N=8*1024*1024, x1[] and x2[] at random*	VL=4	VL=8	VL=8	VL=16
SIMD Static, Auto-Vectorization	6.75 s	-	36.0 s	-
SIMD Static, Explicit Vectorization	**4.63 s**	-	**11.7 s**	-
SIMD Dynamic$_{\text{EvenPartitioning}}$, Explicit Vect.	3.08 s (+26%)	**2.68 s** (+45%)	9.83 s (+3%)	**9.67 s** (+5%)

static

dynamic

Fastest version on KNC with Intrinsics: ~8 s

Final Notes

Dynamic SIMD scheduling can help gaining the overall application performance

- ❑ **Speedup over static SIMD scheduling: 1.5x** and more for our sample
- ❑ Can be implemented with high-level language constructs (also in Fortran)
- ❑ Kind of optimization not carried out by the compiler yet

Where to use?

- ❑ Iteration schemes with varying convergence criteria
- ❑ Dynamic work creation

The End

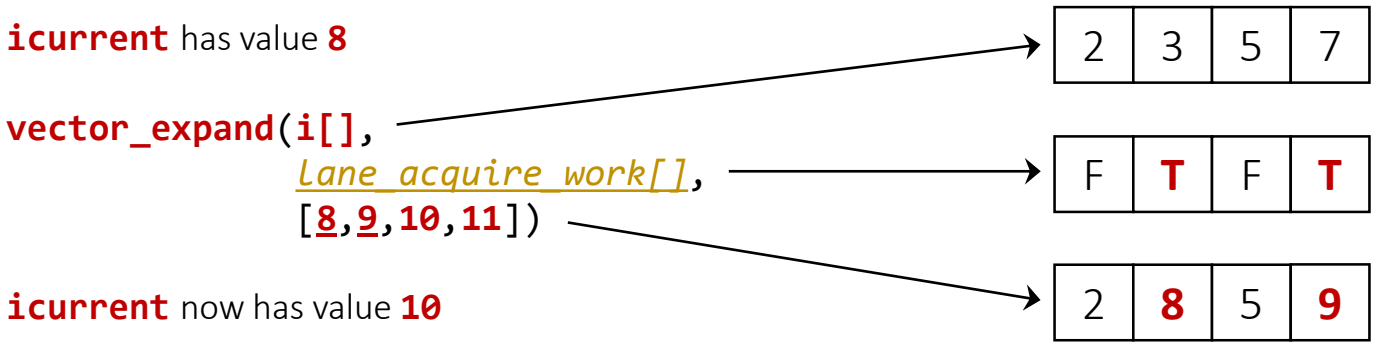
Backup

Dynamic Work Assignment: Vector Expand Operation

Dynamic SIMD (Greedy schedule)

```
for (ii=0; ii<VL; ii++)
   if (lane_acquire_work[ii]) i[ii]=icurrent++
```

SIMD execution requires vector expand operation!



Vector expand (and compress) operations will be available with AVX512!
Currently, we mimic this instruction in software

Dynamic Work Assignment: Test Sample, Performance

Platforms / Software

- ❑ Xeon E5-2680v3 @ 1.9GHz (AVX base frequency)
- ❑ Xeon Phi 5120D
- ❑ Intel C++ compiler 16.0, explicit vectorization with OpenMP 4.0

Runtimes for the loop execution in seconds (final performance)

	Haswell		**Xeon Phi**	
*D=50, N=8*1024*1024, x1[] and x2[] at random*	VL=4	VL=8	VL=8	VL=16
SIMD Static, Auto-Vectorization	6.75 s	-	36.0 s	-
SIMD Static, Explicit Vectorization	4.63 s	-	11.7 s	-
SIMD $Dynamic_{EvenPartitioning}$, Explicit Vect.	3.08 s (+26%)	2.68 s (+45%)	9.83 s (+3%)	9.67 s (+5%)
SIMD $Dynamic_{Greedy}$, Explicit Vect.	3.44 s (+18%)	3.07 s (+32%)	10.1 s (+2%)	11.0 s (-7%)
SIMD $Dynamic_{EvenPartitioning}$, Explicit Vect.	2.81 s (+38%)	-	7.97 s (+26%)	-
SIMD $Dynamic_{Greedy}$, Explicit Vect.	2.93 s (+38%)	-	8.08 s (+27%)	-