HERIOT WATT UNIVERSITY

Data-Layout Inference for Generating Vectorised Code

IFIP 2.11 Meeting

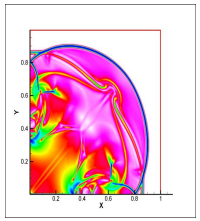
3-6. June. 2013, Minneapolis

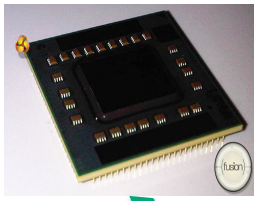
Artem Shinkarov, Sven-Bodo Scholz

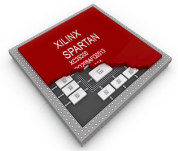
ADVANCE StatArch

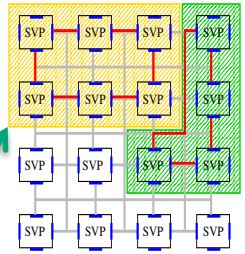
HP^3 Vision

HERIOT WATT UNIVERSITY









compiler

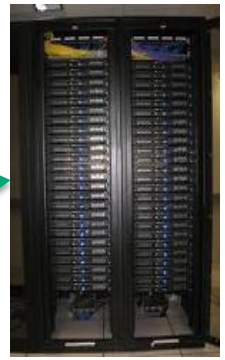
algorithm

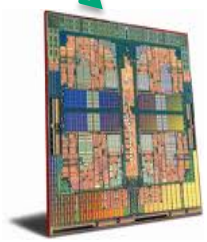
OpenCL

VHDL

µTC

MPI/OpenMP





High **P**roductivity

High **P**erformance

High **P**ortability

Vectorisation is soooo cool!

HERIOT WATT UNIVERSITY

- vectorisation is orthogonal to all other parallelisation efforts!
  - multithreaded SMP gains
  - distributed machines gain
  - GPUs gain

- register width tends to increase rather than decrease

=> a true must have !!!!

Auto-Vectorisation exists!

HERIOT WATT UNIVERSITY

- many techniques have been developed
- polyhedral model gives rather extensive coverage
- we cross-compile to C

=> smells like a free lunch!!!

sad verse:

HERIOT WATT UNIVERSITY

1	2	3
11	12	13
21	22	23
31	32	33
41	42	43
51	52	53
61	62	63
71	72	73

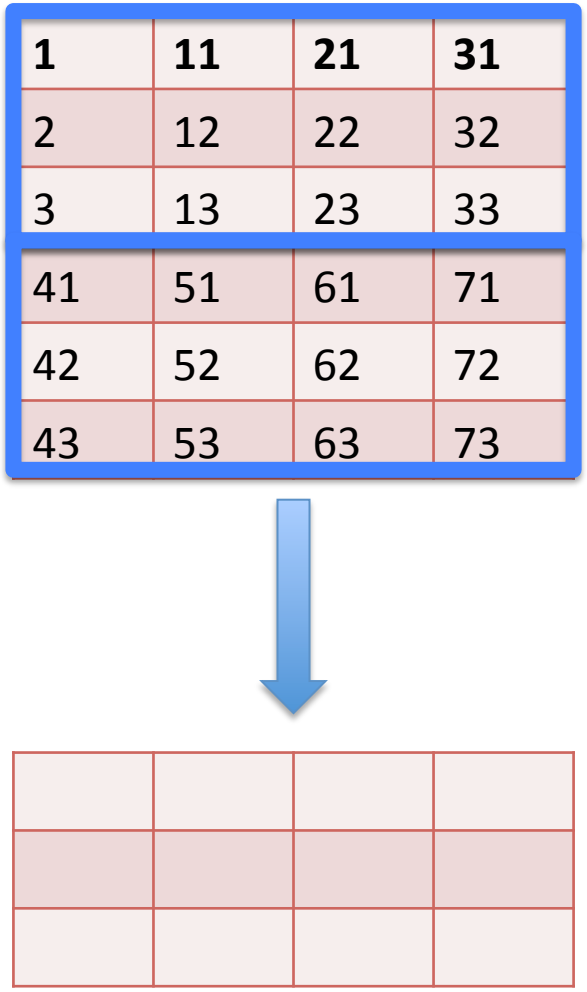

```
res = with {
        ([0]<=[i]<[N]) :a[i]*a[i];
      } : fold( +, [0,0,0]);
```

sad verse:

1	2	3
11	12	13
21	22	23
31	32	33
41	42	43
51	52	53
61	62	63
71	72	73

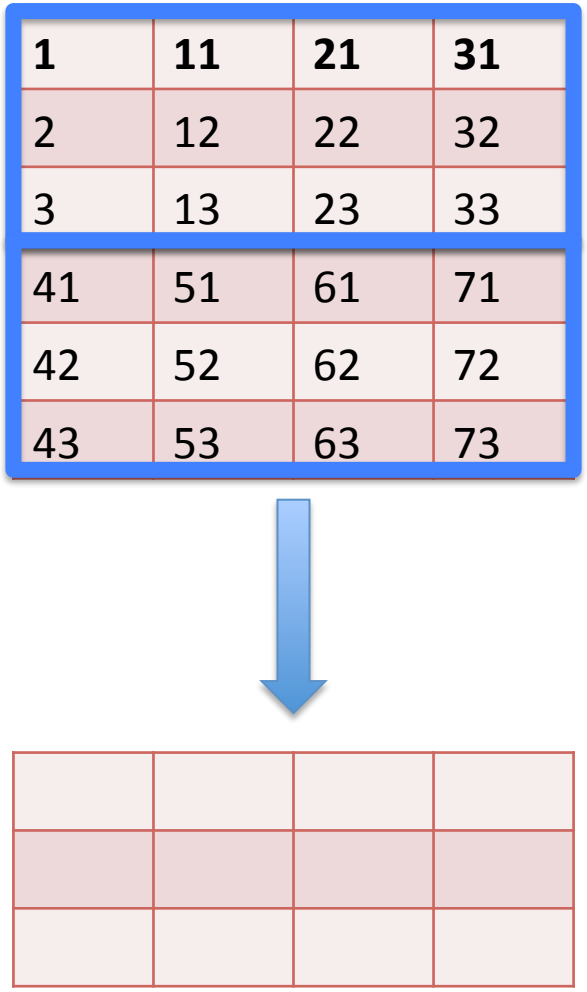

```
res = [0,0,0];
for( i=0; i<N; i++) {
   for( j=0; j<3; j++) {
        res[j] += a[i,j] * a[i,j];
   }
}
```

a new memory layout to the rescue:




```
res' = genarray([3,4], 0);
res' = with {
          ([0]<=[i]<[N/4]:a[i] * a[i];
       } : fold( +, res');
res = with {
          ([0]<=[i]<[3]) : sum(res'[i]);
       } : genarray( [3]);
```

a new memory layout to the rescue:

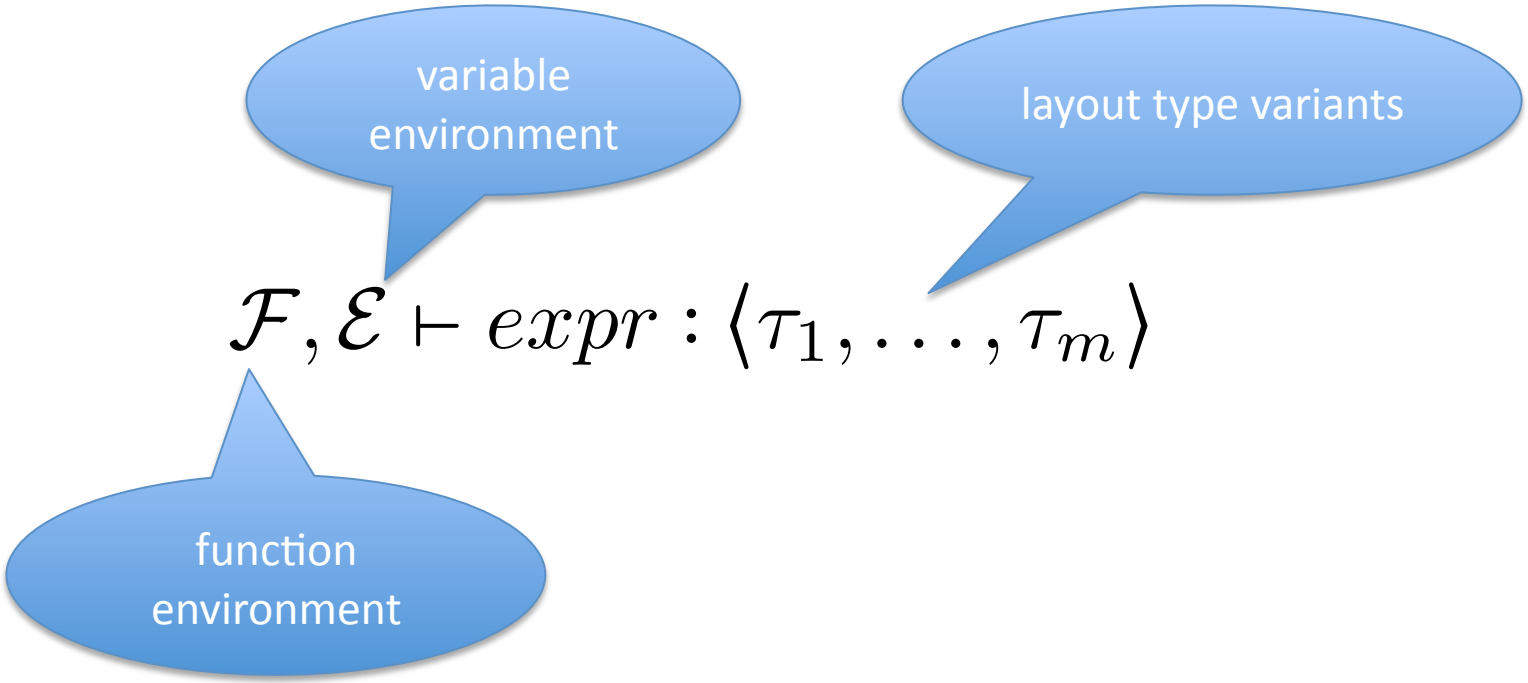



```
res' = genarray([3,4], 0);
for( i=0; i<N/4; i++) {
  for( j=0; j<3; j++) {
    for( k=0; k<4; k++) {
      res'[j,k] += a[i,j,k] * a[i,j,k];
  }
}
res = ...
```

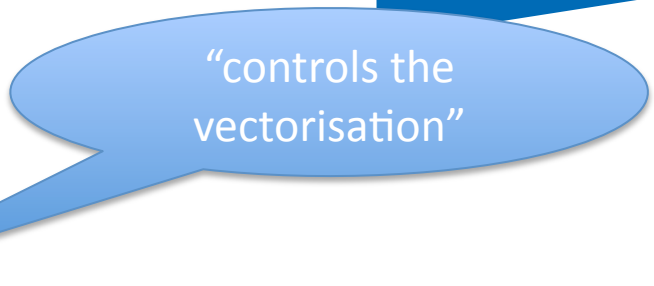
Layout-Types

shape	layout types	layout candidates
[m]	0 1	[m] [m/4, 4]
[m,n]	0 1 2	[m,n] [m/4,n,4] [m,n/4,4]
[m,n,o]	0 1 2 3	[m,n,o] [m/4,n,o,4] [m,n/4,o,4] [m,n,o/4,4]
...	...	...

Layout Typing

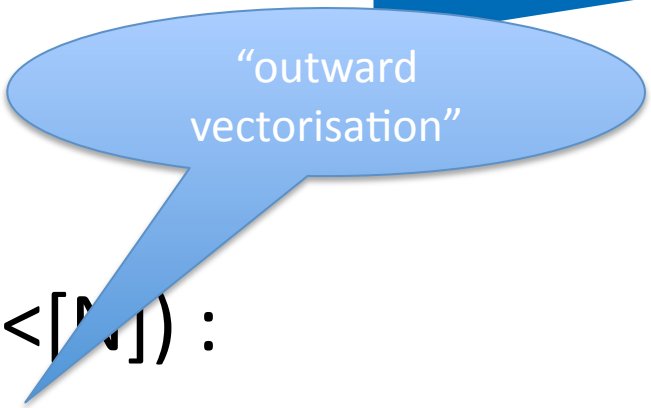


Example




```
res     = with {
            ([0]<=[(i::idx(1))]<[N]) :
                    (a::1)[i]     * (a::1)[i]          ;
          } : fold( +,  [0,0,0]     );
```

Example




```
res     = with {
            ([0]<=[(i::idx(1))]<[N]) :
                ((a::1)[i]::Δ)*((a::1)[i]::Δ)     ;
          } : fold( +,  [0,0,0]     );
```

$$\textsc{Sel}[\triangle]: \frac{\rho_i = \begin{cases} \triangle & \tau_i = idx(k) \wedge \sigma_i = k \wedge k \in \mathbb{Z}_+ \\ \triangle & \tau_i = 0 \wedge \sigma_i = \triangle \\ 0 & \tau_i = 0 \wedge \sigma_i \in \mathbb{N} \end{cases}}{\mathcal{F}, \mathcal{E} \vdash \text{sel } (j, a) : \langle \rho_1, \ldots, \rho_n \rangle}$$

Example

HERIOT WATT UNIVERSITY

"actual" vectorisation

```
res     = with {
            ([0]<=[(i::idx(1))]<[N]) :
                (((a::1)[i]::Δ)*((a::1)[i]::Δ)::Δ);
          } : fold( +,  [0,0,0]     );
```

$$\textsc{Prf}[\triangle]: \frac{\begin{array}{c}\mathcal{F},\mathcal{E} \vdash e_1 : \langle \tau_1, \ldots, \tau_n \rangle \\ \mathcal{F},\mathcal{E} \vdash e_2 : \langle \tau'_1, \ldots, \tau'_n \rangle \\ \forall 1 \leq i \leq n \\ \sigma_i = \begin{cases} 0 & (\tau_i = 0) \wedge (\tau'_i = 0) \\ \triangle & (\tau_i = 0) \wedge (\tau'_i = \triangle) \\ \triangle & (\tau_i = \triangle) \wedge (\tau'_i = 0) \\ \triangle & (\tau_i = \triangle) \wedge (\tau'_i = \triangle) \end{cases}\end{array}}{\mathcal{F},\mathcal{E} \vdash +(e_1, e_2) : \langle \sigma_1, \ldots, \sigma_n \rangle}$$

Example

```
(res::0) = with {
        ([0]<=[(i::idx(1))]<[N]) :
            (((a::1)[i]::Δ)*((a::1)[i]::Δ)::Δ);
    } : fold( +, ([0,0,0]::Δ));
```

$$\sigma_k^{bin} \equiv (\sigma_k, \sigma_k) \to \sigma_k$$
$$\mathcal{F}, \mathcal{F}(f) \vdash f : \langle \sigma_1^{bin}, \ldots, \sigma_n^{bin} \rangle$$
$$\mathcal{F}, \mathcal{E} \oplus (j, \langle \tau_1, \ldots, \tau_n \rangle) \vdash e : \langle \sigma_1, \ldots, \sigma_n \rangle$$
$$\forall 1 \le i \le n$$
$$\rho_i = \begin{cases} 0 & \tau_i = idx(k) \wedge \sigma_i = \Delta \\ \Delta & \tau_i = 0 \wedge \sigma_i = \Delta \\ \sigma_i & \tau_i = 0 \wedge \sigma_i \in \mathbb{N} \end{cases}$$

$$\textsc{Red}[\Delta] : \frac{}{\mathcal{F}, \mathcal{E} \vdash \text{reduce } j < u \ f \ e : \langle \rho_1, \ldots, \rho_n \rangle}$$

N-Body: the core

HERIOT WATT UNIVERSITY

effect of pos2 on pos 1

```
double[3] accelaration( double[3] pos1, double[3] pos2, double mass2)
{
  distance = nonzerol2norm( pos2 - pos1);
  return( ( pos2 - pos1) * ( mass2 / ( distance * distance * distance)) );
}
```

accumulated effect of all N planets in pos2s on pos 1

```
double[3] acceleration( double[3] pos1, double[N,3] pos2s, double[N] masss)
{
  res = [0d,0d,0d];
  for( i=0; i< shape(masss)[[0]]; i++) {
    res += accelaration( pos1, pos2s[[i]], masss[[i]]);
  }
  return( res);
}
```

Full N-Body

```
double[N,3], double[N,3]  advance( double [N,3] planet_pos,
                                   double [N,3] planet_vel,
                                     double [N] planet_mass,
                                       double dt)
{
  accs = with {
         (.<= [i] <=.) : acceleration( planet_pos[[i]], planet_pos, planet_mass );
       } : genarray( shape( planet_mass), [0d,0d,0d]);
  planet_vel = planet_vel + accs * dt;
  planet_pos = planet_pos + planet_vel * dt;
  return( planet_pos, planet_vel);
}
```

Vectorisation Attempt I

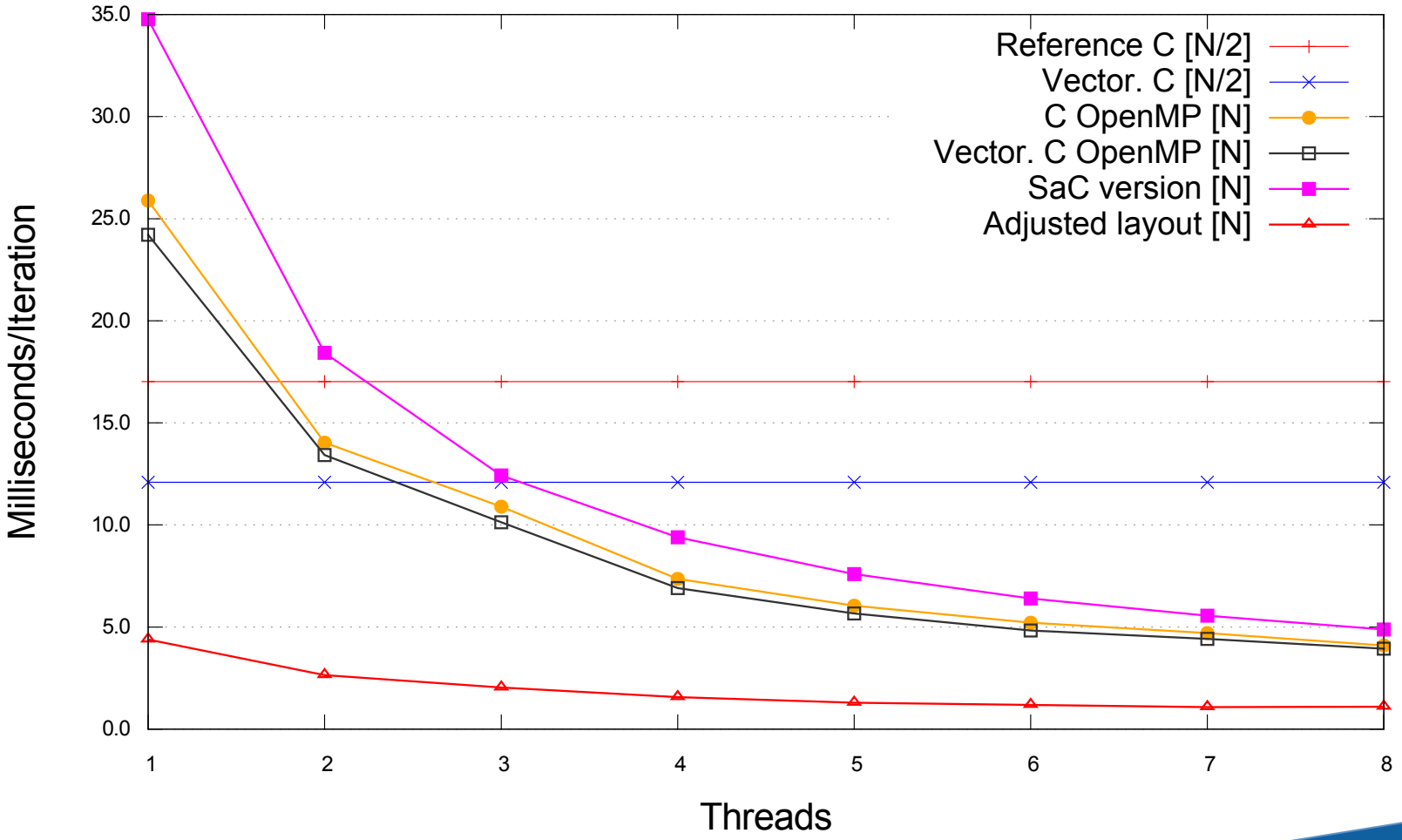
```
double[4] accelaration( double[4] pos1, double[4] pos2, double mass2) {
  distance = nonzerol2norm( pos2 - pos1);
  return( ( pos2 - pos1) * ( mass2 / ( distance * distance * distance)) );
}

double[4] acceleration( double[4] pos1, double[N,4] pos2s, double[N] masss) {
  res = [0d,0d,0d,0d];
  for( i=0; i< shape(masss)[[0]]; i++) {
    res += accelaration( pos1, pos2s[[i]], masss[[i]]);
  }
  return( res);
}
double[N,4], double[N,4]  advance( double [N,4] planet_pos, double [N,4] planet_vel,
                                   double [N] planet_mass, double dt) {
   accs = with {
          (.<= [i] <=.) : acceleration( planet_pos[[i]], planet_pos, planet_mass );
        } : genarray( shape( planet_mass), [0d,0d,0d,0d]);
   planet_vel = planet_vel + accs * dt;
   planet_pos = planet_pos + planet_vel * dt;
   return( planet_pos, planet_vel);
}
```

Vectorisation Attempt II

```
double[3,4] accelaration( double[3,4] pos1, double[3,4] pos2, double[4] mass2) {
  distance = nonzerol2norm( pos2 - pos1);
  return( ( pos2 - pos1) * ( mass2 / ( distance * distance * distance)) );
}
double[3,4] acceleration( double[3,4] pos1, double[N/4,3,4] pos2s, double[N/4,4] masss) {
  res = [0d,0d,0d];
  for( i=0; i< shape(masss)[[0]]; i++) {
    res += accelaration( pos1, pos2s[[i]], masss[[i]]);
  }
  return( res);
}
double[N/4,3,4], double[N/4,3,4]
advance( double [N/4,3,4] planet_pos, double [N/4,3,4] planet_vel,
                double [N/4,4] planet_mass, double dt) {
   accs = with {
          (.<= [i] <=.) : acceleration( planet_pos[[i]], planet_pos, planet_mass );
        } : genarray( shape( planet_mass), [0d,0d,0d]);
   planet_vel = planet_vel + accs * dt;
   planet_pos = planet_pos + planet_vel * dt;
   return( planet_pos, planet_vel);
}
```

Performance Impact



Conclusions

HERIOT WATT UNIVERSITY

- choice of data layout is critical for vectorisation
- layouts as types ensure consistency
- inference is possible
- enables high-level program transformation
- optimal layout depends on hardware!