Arrow Street: Semi-automatic SOA / AOSOA

Pascal Costanza

ExaScience Lab, Intel, Belgium

Legal Notices

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

[*BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Core Inside, i960, Intel, the Intel logo, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, InTru, the InTru logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, skoool, the skoool logo, Sound Mark, The Journey Inside, vPro Inside, VTune, Xeon, and Xeon Inside*] are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Bluetooth is a trademark owned by its proprietor and used by Intel Corporation under license.

Intel Corporation uses the Palm OS® Ready mark under license from Palm, Inc.

ExaScience Lab

Flanders Exascale Lab

WAS LAUNCHED IN 2010 TO INVESTIGATE NEXT GENERATION **HIGH PERFORMANCE COMPUTING**

IS A COLLABORATION BETWEEN:

INTEL, FLANDERS, IMEC, KU LEUVEN, U GENT, VU BRUSSEL, U ANTWERPEN, U HASSELT

Intel Exascale Labs — Europe

Strong Commitment To Advance Computing Leading Edge:

Intel collaborating with HPC community & European researchers

4 labs in Europe - Exascale computing is the central topic

ExaScale Computing Research Lab, Paris

Performance and scalability of Exascale applications

Tools for performance characterization

ExaCluster Lab, Jülich

Exascale cluster scalability and reliability

ExaScience Lab, Leuven

Comms avoiding algorithms

Architectural simulation

Scalable kernels and RT

Intel and BSC Exascale Lab, Barcelona

Scalable RTS and tools

New algorithms

www.exascale-labs.eu

Overview

- Manual AOS, SOA, and AOSOA representations.
- Semi-automatic SOA and AOSOA representations.
- Early results.

Example Class

```
struct C {
    double x;
    double y;
    double z;

    C () {}

    C (double x, double y, double z) :
        x(x), y(y), z(z)
    {}
};
```

Different representations for “many” objects

	0	1	2	3	4	5	6	7	8	9	10	11	…
Standard AOS	x_0	y_0	z_0	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3	…

Different representations for “many” objects

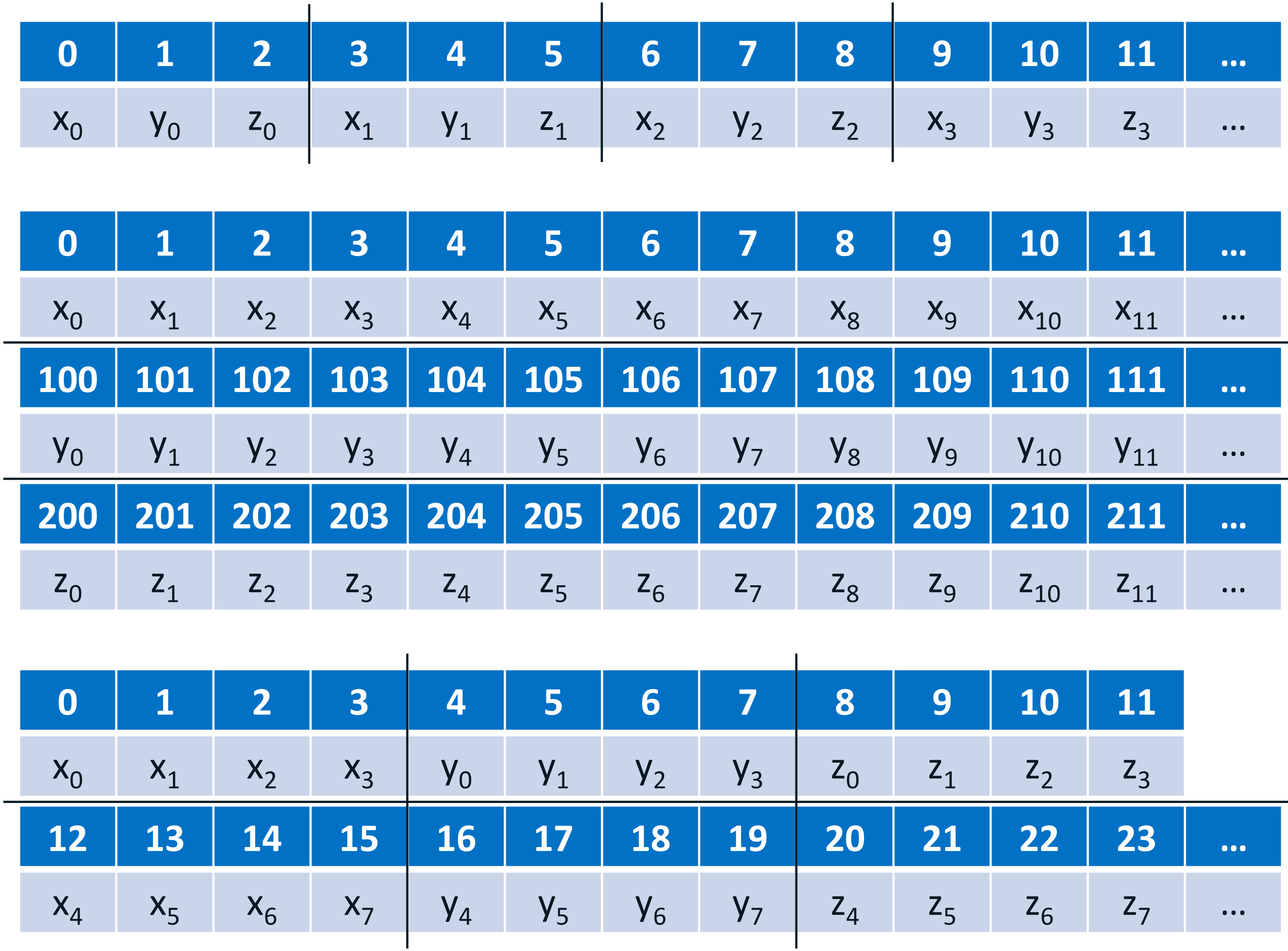
Standard AOS

0	1	2	3	4	5	6	7	8	9	10	11	…
x_0	y_0	z_0	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3	…

SOA

0	1	2	3	4	5	6	7	8	9	10	11	…
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	…
100	101	102	103	104	105	106	107	108	109	110	111	…
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	…
200	201	202	203	204	205	206	207	208	209	210	211	…
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	…

Different representations for “many” objects



Standard AOS

0	1	2	3	4	5	6	7	8	9	10	11	…
x_0	y_0	z_0	x_1	y_1	z_1	x_2	y_2	z_2	x_3	y_3	z_3	…

SOA

0	1	2	3	4	5	6	7	8	9	10	11	…
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	…
100	101	102	103	104	105	106	107	108	109	110	111	…
y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	…
200	201	202	203	204	205	206	207	208	209	210	211	…
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	…

AOSOA

0	1	2	3	4	5	6	7	8	9	10	11	
x_0	x_1	x_2	x_3	y_0	y_1	y_2	y_3	z_0	z_1	z_2	z_3	
12	13	14	15	16	17	18	19	20	21	22	23	…
x_4	x_5	x_6	x_7	y_4	y_5	y_6	y_7	z_4	z_5	z_6	z_7	…

Advantages and disadvantages of different representations

- AOS
  - All member of each instance next to each other.
  - Good for data locality, less good for vectorization.
  - Not good when only a subset of the members is needed.

- SOA
  - Each member of all instances next to each other.
  - Good for vectorization, less good for data locality. (SSE2, AVX, ...)
  - Good when only a subset of the members is needed.

- AOSOA
  - Balances advantages and disadvantages of AOS and SOA.

Standard AOS representation

```
C arr[len];
double result = 0;

for (auto i=0; i<len; ++i) {
  arr[i].x += arr[i].y * arr[i].z;
  result += arr[i].x;
}
```

Manual SOA representation

```
struct Cv {
  double* x;
  double* y;
  double* z;

  Cv (int len) :
    x(new double[len]), y(new double[len]), z(new double[len])
  {}

  ~Cv () {delete x[]; delete y[]; delete z[];}
};
```

Manual SOA representation

```
Cv arr(len);
double result = 0;

for (auto i=0; i<len; ++i) {
   arr.x[i] += arr.y[i] * arr.z[i];
   result += arr.x[i];
}
```

Manual SOA representation

- The user has to write the new class.
  (but that may be ok, because it could even be generated)

- The user has to recode all accesses to array members.
  (arr[i].x ⇔ arr.x[i])

- This requires invasive changes to the code.
  - Cumbersome and annoying,
    especially if it turns out that SOA is not better than AOS after all.
  - Developers are not willing to do this,
    and give up on the performance opportunity!

Manual SOA representation

- The user has to write the new class.
  (but that may be ok, because it could even be generated)
- The user has to recode all accesses to array members.
  (arr[i].x ⇔ arr.x[i])
- This requires invasive changes to the code.
  - Cumbersome and annoying,
    especially if it turns out that SOA is <u>not</u> better than AOS after all.
  - Developers are not willing to do this,
    and give up on the performance opportunity!
- Manual AOSOA representation is even more complicated!

Arrow Street

- A library for semi-automatic SOA/AOSOA data layouts. ("array / structures" => "arrow street")
- Based on modern C++11 language constructs, specifically:
  - std::tuple
  - variadic templates
  - type_traits
  - lambda expressions
- Available at https://github.com/ExaScience/arrow-street

Semi-automatic SOA representation with Arrow Street

```
struct Cr {
  double& x;
  double& y;
  double& z;

  typedef reference_type<double, double, double> reference;

  Cr (const reference::type& ref) :
    x(reference::get<0>(ref)),
    y(reference::get<1>(ref)),
    z(reference::get<2>(ref))
  {}
};
```

Semi-automatic SOA representation with Arrow Street

```
table<Cr,len> array;
double result = 0;

for (auto i=0; i<len; ++i) {
  arr[i].x += arr[i].y * arr[i].z;
  result += arr[i].x;
}
```

Semi-automatic SOA representation with Arrow Street

- The user has to write the new class.
  (but that may be ok, because it could even be generated)

- The user does not have to recode accesses to array members.
  (arr[i].x ⇔ arr[i].x)
  - Only variable and parameter declarations have to be changed, or can be turned into template parameters.
  - The accesses themselves can stay the same.
  - This allows for switching back and forth between AOS & SOA.

Semi-automatic AOSOA representation with Arrow Street

```
table_array<Cr,256> array(len);
double result  = 0;

for_each(array, [&] (Cr& element) {
   element.x += element.y * element.z;
   result += element.x;
});
```

Flexible representations with Arrow Street

-
  ```
  table_vector<Cr,tablesize> v(len);
  table_vector<Cr,len> v(len);
  table_vector<Cr,1> v(len);
  std::vector<C> v(len);
  ```

-
  ```
  table_array<Cr,tablesize,len> a;
  table_array<Cr,len,len> a;
  table_array<Cr,1,len> a;
  std::array<C,len> a;
  ```

First results (on Core i7 860 @ 2.80 GHz)

len: 100000
repeat: 1000000

flat AOS array:	171 sec
flat SOA array:	63 sec
std::array:	171 sec
nested SOA array, tablesize 1:	182 sec
nested SOA array, tablesize max:	63 sec
nested SOA array, tablesize 256:	70 sec
std::vector:	167 sec
nested SOA vector, tablesize 1:	212 sec
nested SOA vector, tablesize max:	64 sec
nested SOA vector, tablesize 256:	70 sec

First results: N-body kernel, original code

```
static float xx[N], yy[N], zz[N], mass[N], vx1[N], vy1[N], vz1[N];

for (i=0; i<M; ++i) {
    step(n, xx[i], yy[i], zz[i], c0, c1, xx, yy, zz, mass,
             &dx1, &dx2, &dx3);

    vx1[i] += dx1 * f;
    vy1[i] += dy1 * f;
    vz1[i] += dz1 * f;
}
```

N-body: New code

```
struct body {
   float xx, yy, zz, mass, vx1, vy1, vz1;
};

static body bodies[N];

for (i=0; i<M; ++i) {
  d1  = step(bodies, n, i, c0, c1);

  bodies[i].vx1  += d1.x * f;
  bodies[i].vy1  += d1.y * f;
  bodies[i].vz1  += d1.z * f;
}
```

N-body: Arrow Street code

```
struct body {
   float &xx, &yy, &zz, &mass, &vx1, &vy1, &vz1;
... };

static soa::table<body,N> bodies;

for (i=0; i<M; ++i) {
  d1  = step(bodies, n, i, c0, c1);

  bodies[i].vx1  += d1.x * f;
  bodies[i].vy1  += d1.y * f;
  bodies[i].vz1  += d1.z * f;
}
```

N-body: Arrow Street code

```
struct body {
   float &xx, &yy, &zz, &mass, &vx1, &vy1, &vz1;
... };

static soa::table<body,N> bodies;

for (i=0; i<M; ++i) {
  d1 = step(bodies, n, i, c0, c1);

  bodies[i].vx1 += d1.x * f;
  bodies[i].vy1 += d1.y * f;
  bodies[i].vz1 += d1.z * f;
}
```

Only declarations change.

Client code remains the same
as in traditional AOS code.

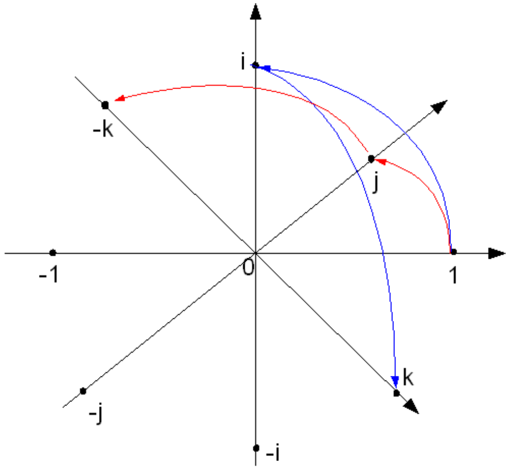
Performance of Arrow Street code is the
same as manually optimized data layout!!!

Other early results

- Helsim
  - 3D Electromagnetic Particle-In-Cell Simulation with In-Situ Visualization.
  - Developed at Intel ExaScience Lab, Belgium.
  - Arrow Street improves performance in charge/current deposition and particle moving.

Other early results

- Quaternion benchmark
  - Data representation used for animation and gaming systems.
  - Case study on Arrow Street performed at Intel.
  - Performance improvements in compute-intensive algorithms, competitive with Cilk Extended Array Notation.



Graphical representation of quaternion units product as 90°-rotation in 4D-space

Additional features

- Support for nested AOSOA representations.
- Support for complex data structures (composition/inheritance)
- Support for standard containers, especially iterators
- Parallel iteration using Cilk and TBB
  - ...including support for TBB range concept.
- No support for std::deque and OpenMP yet,
  but should be easy to add.
- Open source release at https://github.com/ExaScience/arrow-street

Conclusions

- SOA and AOSOA data representations are beneficial for SIMD instructions (SSE2, AVX).

- Current compilers do not support SOA/AOSOA well.

- Manual SOA/AOSOA is cumbersome and invasive, and developers give up on performance opportunity.

- Arrow Street is a pure library solution for C++11 that makes semi-automatic SOA/AOSOA substantially easier to express.

Thank You

intel®