



# O que é melhor? “Array de Structs” ou “Struct de Arrays”?

12º Encontro do grupo C & C++ do Brasil

*André Tupinambá*



# Arrays de Structs

- Meio clássico de definir um conjunto de estruturas.

```
struct XYV {  
    float x;  
    float y;  
    double value;  
};
```

```
XYV array_of_struct[100];  
XYV* pointer_to_array_of_struct = new XYV[100];  
std::vector< XYV > vector_of_struct(100);
```

```
struct XYV {  
    std::vector<float> x;  
    std::vector<float> y;  
    std::vector<double> value;  
};
```

- É uma alternativa viável?
- Rápida?
- Faz sentido?
- Por que não?



## Case Study

A case study comparing AoS (Arrays of Structures) and SoA (Structures of Arrays) data layouts for a compute-intensive loop run on Intel® Xeon® processors and Intel® Xeon Phi™ product family coprocessors

<http://intel.ly/1P6UTpe>



# Caso de estudo da Intel

```
typedef struct Point3d
{
    FLOATINGPTPRECISION x,y,z,t;
}
Point3d,*Point3dPtr;

#if AOSFLAG == 1
    Point3dPtr org = NULL;
    Point3dPtr tfm = NULL;
#endif
#if AOSFLAG == 0
    FLOATINGPTPRECISION *orgx = NULL;
    FLOATINGPTPRECISION *orgy = NULL;
    FLOATINGPTPRECISION *orgz = NULL;
    FLOATINGPTPRECISION *tfmx = NULL;
    FLOATINGPTPRECISION *tfmy = NULL;
    FLOATINGPTPRECISION *tfmz = NULL;
#endif
```



# Caso de estudo da Intel

```
#if AOSFLAG == 1
#pragma omp parallel for
for(i=0;i<nxfmpts;i++)
{
    x = tfm[i].x; y = tfm[i].y; z = tfm[i].z;
    tfm[i].x = Rf[0][0]*x + Rf[0][1]*y + Rf[0][2]*z + Tf[0];
    tfm[i].y = Rf[1][0]*x + Rf[1][1]*y + Rf[1][2]*z + Tf[1];
    tfm[i].z = Rf[2][0]*x + Rf[2][1]*y + Rf[2][2]*z + Tf[2];
}
#endif
#if AOSFLAG == 0
#pragma omp parallel for
for(i=0;i<nxfmpts;i++)
{
    x = tfmx[i]; y = tfmy[i]; z = tfmz[i];
    tfmx[i] = Rf[0][0]*x + Rf[0][1]*y + Rf[0][2]*z + Tf[0];
    tfmy[i] = Rf[1][0]*x + Rf[1][1]*y + Rf[1][2]*z + Tf[1];
    tfmz[i] = Rf[2][0]*x + Rf[2][1]*y + Rf[2][2]*z + Tf[2];
}
#endif
```



# E com ótimos resultados

Data	Precision	Arch	Instr	P=Parallel	V=Vector	B=P*V	GF/s	Run1500
Soa	Float	Host	Avx	14.08014	5.870481	82.65718	257.748	3146.14
Soa	Float	Host	Sse	16.07025	4.039411	64.91434	156.888	5163.16
Aos	Float	Host	Avx	16.53501	2.455873	40.60788	108.708	7459.55
Aos	Float	Host	Sse	16.28717	2.473247	40.28219	94.033	8613.25
Soa	Double	Host	Avx	16.2062	2.628223	42.5935	111.565	7261.86
Soa	Double	Host	Sse	16.76103	1.814117	30.40646	72.434	11201.7
Aos	Double	Host	Avx	16.64526	1.704506	28.37196	73.953	10954.1
AoS	Double	Host	Sse	17.52019	1.616346	28.31869	61.664	13139.5

2.4x mais  
rápido  
AVX-float

1.5x mais  
rápido  
AVX-double

Soa	Float	Phi	IMCI	124.9218	15.64726	1954.684	483.554	1675.99
Aos	Float	Phi	IMCI	117.2152	3.641535	426.8433	113.657	7126.97
Soa	Double	Phi	IMCI	130.8295	6.449287	843.7571	199.925	6510.98
Aos	Double	Phi	IMCI	131.9614	1.8744	247.3485	53.734	15074.4

Table 3. ICP Results Table for Xeon Phi SE10P 1.09 Ghz vs. Xeon Host E5-2670 2.6 GHz.

## Por quê?



## A explicação do paper é simples

“(...) AoS makes the compiler **create more instructions** even though AoS and SoA are both **vectorized**: 47 vs. 19 (AVX-1), 38 vs. 23 (IMCI), 48 vs. 26 (SSE). More instructions require more time (...).

**Even though the C source code looks innocuously similar, the assembly differences are substantial.”**





# Bora usar SoA então!

- Primeiro teste
- Usar com um container associativo
  - Como o `std::map` e o `boost::flat_map`
  - `boost::flat_map` utiliza um `vector< pair<K, V> >`
- Teste com uma implementação com `vector< K >` e `vector< V >`



# Casos de teste

- Mapa de `< size_t, size_t >`
  - Preenchimento de 0 a 100.000.000
    - Busca por todos os itens desse vetor
  - Preenchimento de 100.000 a 0
    - Busca por todos os itens desse vetor
- Comparando
  - `std::map`
  - `boost::container::flat_map`
  - `ccppbrasil::soa_map`






## Resultado...

	100.000.000		100.000	
	fwd fill	fwd find	rew fill	rew find
boost::container::flat_map	6,525368	7,363206	6,199282	0,006059
ccppbrasil::soa_map	7,278355	6,829322	6,035210	0,005151
std::map	60,627468	29,123738	0,019671	0,013398

É... Não parece tão bom assim...




## Vamos analisar – Forward fill

- `boost::flat_map`
  - Busca binária no vetor de pares
  - Inserir um par no final do vetor
- `ccppbrasil::soa_map`
  - Busca binária no vetor de chaves
  - Insere um item no final do vetor de chaves 
  - Insere um item no final do vetor de valores 
- `std::map`
  - Busca na árvore
  - Aloca de área de memória 
  - Cópia da chave e valor



# Vamos analisar – Reverse fill

- `boost::flat_map`
  - Busca binária no vetor de pares
  - Move todos os dados do vetor uma posição 
  - Inserir um par no início do vetor
- `ccppbrasil::soa_map`
  - Busca binária no vetor de chaves
  - Move todos os dados do vetor de chaves uma posição
  - Move todos os dados do vetor de valores uma posição
  - Insere um item no final do vetor de chaves
  - Insere um item no final do vetor de valores
- `std::map`
  - Busca na árvore
  - Aloca de área de memória
  - Cópia da chave e valor



## Vamos analisar – Find

- `boost::flat_map`
  - Busca binária no vetor de pares
  - Constrói iterador e retorna
- `ccppbrasil::soa_map`
  - Busca binária no vetor de chaves
  - Constrói iterador e retorna
- `std::map`
  - Busca na árvore
  - Constrói iterador e retorna

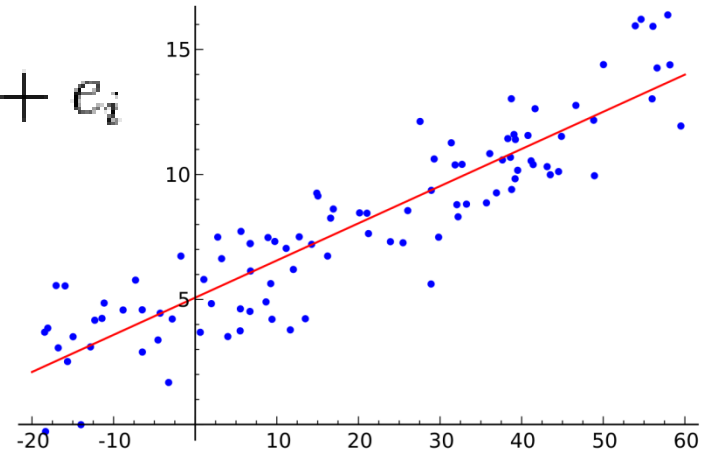


- Cálculo de regressão linear por mínimos quadrados.

$$y_i = a + bx_i + e_i$$

$$a = \bar{y} - b\bar{x}$$

$$b = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$



- Mesma implementação, duas estruturas de dados:
  - `vector< pair< float, float > > xy;`
  - `vector< float > x; vector< float > y;`



# Resultado...

Least Squares	seg
AoS	1,4650138
SoA	1,3761149

1,06x mais  
rápido







# Mais um teste...

- Soma de vetores...
  - Dois vetores de pares
    - 2x - `vector< pair < float, float > > xy;`
  - Dois pares de vetores
    - 2x - `vector< float > x; vector< float > y;`

2 float	seg
AoS	9,591524
SoA	13,0273727



1,36x mais  
**LENTO!**

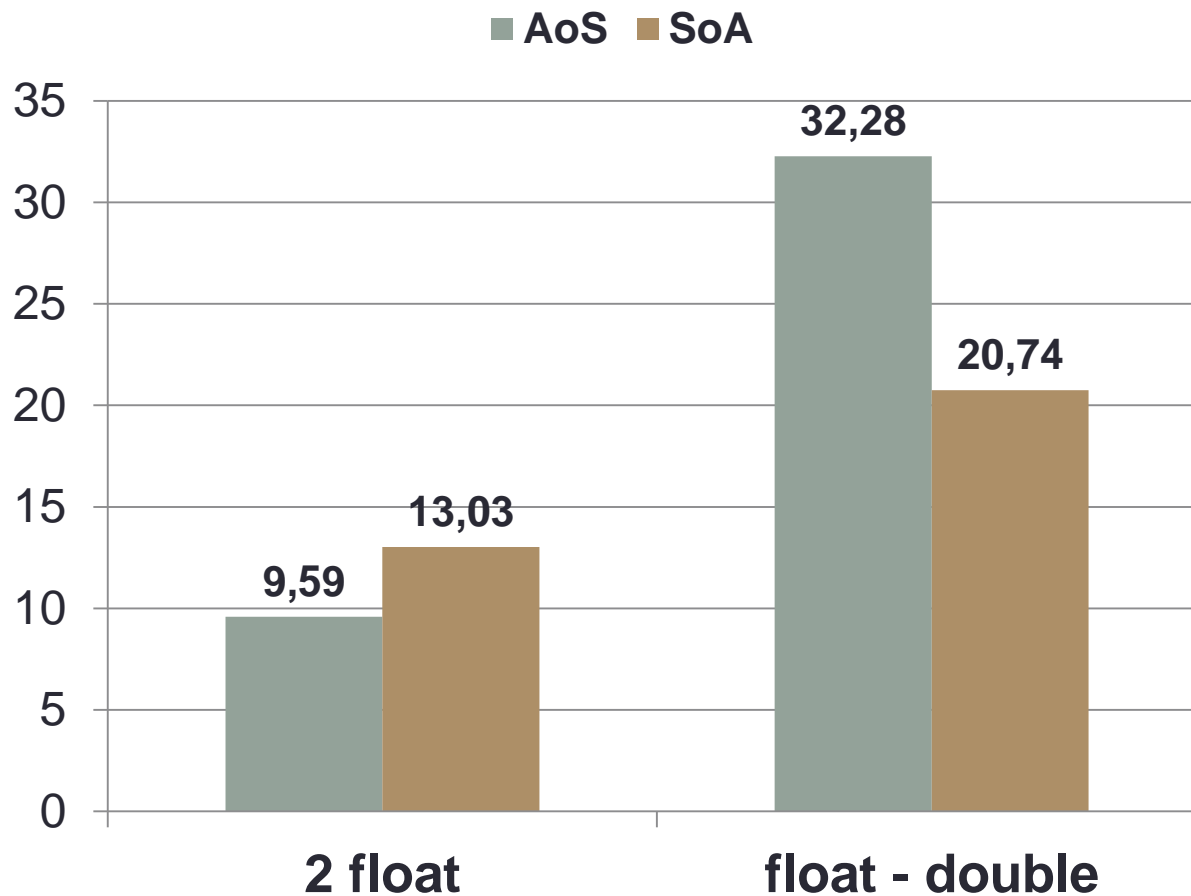


# E se tiverem tipos misturados?

- Dois vetores de pares
  - 2x - `vector< pair < float, double > > xy;`
- Dois pares de vetores
  - 2x - `vector< float > x; vector< double > y;`

float - double	seg
AoS	32,2758659
SoA	20,740165

1,56x mais rápido







# Conclusão

- Struct de Arrays é mais uma técnica de trabalho.
- Continua tudo o mesmo:
  - Conhecer o compilador
    - Entender o assembly que ele gera
  - Benchmark
  - Um pouco de tentativa e erro



# O que é melhor? “Array de Structs” ou “Struct de Arrays”?

12º Encontro do grupo C & C++ do Brasil

*André Tupinambá*