**StaticArrays Syntax and benchmarking report**

**I. Svector and Mvector**

**1. Hard-code size-known creation:**

**TP1: Create a 3-element SVector{3, Int64} # or float64**

1. **Svector**

v0=**SA**[1,2,3]

v1 = **SVector**(1,2,3)

v2 = **@SVector** [1,2,3]

v3=**SVector**{3,Int}(1,2,3)

vect=(1,2,3) or [1,2,3]

v4=**SVector**{3,Int}(vect)

v5 = **@SVector** **zeros**(3) #Float64

v6 = **@SVector** **randn**(Float64, 40)

Access it via V[1];1

setIndex! Is illegal. Setindex Modification creates a new svector. v5=**setindex**(v5,10,2)

1. **Mvector**

Same as Svector change S to M except **SA**[1,2,3].

**2. Benchmark fetch and update of size-known created Svectors and Mvectors:**

**TP2: compare stack-fetch vs heap-fetch via Svectors,vectors,Mvectors**

|  |  |  |
| --- | --- | --- |
| function **simulatorSvector**()  s=0.0  v1=**@SVector**[1.1,2.2,3.3,4.4,5.5]  for i = 1:1e+6 for j = 1:5  s=s+v1[j]  end end  s  end | function **simulatorvector**()  s=0.0  v1 = [1.1,2.2,3.3,4.4,5.5]  for i = 1:1e+6 for j = 1:5  s=s+v1[j]  end end  s  end | function **simulatorSvector**()  s=0.0  v1=**@MVector**[1.1,2.2,3.3,4.4,5.5]  for i = 1:1e+6 for j = 1:5  s=s+v1[j]  end end  s  end |

**@btime** **simulatorSvector**()**@btime** **simulatorvector**()**@btime** **simulatorMvector**()

4.286 ms (0 allocations: 0 bytes)

4.289 ms (1 allocation: 96 bytes)

4.261 ms (0 allocations: 0 bytes)

**TP3: compare stack-fetch vs heap-fetch via Svectors,vectors,Mvectors and rand()**

v1 = **@SVector** **rand**(Float64, 100)

v2 = **rand**(100)

v3 = **@MVector** **rand**(Float64, 100)

**For states=40**

33.553 μs (0 allocations: 0 bytes)

32.709 μs (1 allocation: 400 bytes)

32.643 μs (1 allocation: 336 bytes)

**For states=100**

81.604 μs (0 allocations: 0 bytes)

81.623 μs (1 allocation: 896 bytes)

81.626 μs (1 allocation: 816 bytes)

**TP4: compare stack-fetch vs heap-fetch via Svectors,vectors,Mvectors and rand() and random access of the vectors.**

n=**rand**(1:100)

s=s+v1[n]

610.482 μs (0 allocations: 0 bytes)

605.228 μs (1 allocation: 896 bytes)

603.799 μs (1 allocation: 816 bytes)

**TP5: compare stack-recreate vs heap-update via Svectors,vectors,Mvectors**

|  |  |  |
| --- | --- | --- |
| function **simulatorSvector**()  v1 = **@SVector** **zeros**(5)  for i = 1:1e+6 for j = 1:5  v1=**setindex**(v1,(i-j)\*.54,j)  end end  v1  end | function **simulatorvector**()  v1 = **Vector**{Float64}(undef, 5)  for i = 1:1e+6 for j = 1:5  v1 = (i - j) \* 0.54  end end  v1  end | function **simulatorSvector**()  v1 = **@SVector** **zeros**(5)  for i = 1:1e+6 for j = 1:5  v1 = (i - j) \* 0.54  end end  v1  end |

**@btime** **simulatorSvector**()**@btime** **simulatorvector**()**@btime** **simulatorMvector**()

13.243 ms (0 allocations: 0 bytes)

5.086 ms (1 allocation: 96 bytes)

3.498 ms (1 allocation: 48 bytes)

For states=10

30.197 ms (0 allocations: 0 bytes)

7.967 ms (1 allocation: 144 bytes)

4.163 ms (1 allocation: 96 bytes)

**TP6: compare stack-recreate vs heap-update via vectors and svectors and Mvectors and random**

|  |  |  |
| --- | --- | --- |
| function **simulatorSvector**()  v1=**@SVector** **rand**(Float64,100)  for i = 1:1e+3  for j = 1:100  n=**rand**(1:100)  v1=**setindex**(v1,(i-j)\*0.54,n)  end  end  v1  end | function **simulatorMvector**()  v1=**@MVector** **rand**(Float64,100)  for i = 1:1e+3  for j = 1:100  n=**rand**(1:100)  v1[n] = (i - j)\*0.54  end  end  v1  end | function **simulatorvector**()  v1 = **rand**(100)  for i = 1:1e+3  for j = 1:100  n=**rand**(1:100)  v1[n] = (i - j) \* 0.54  end  end  v1  end |

**@btime** **simulatorSvector**()**@btime** **simulatorvector**()**@btime** **simulatorMvector**()

3.058 ms (0 allocations: 0 bytes)

645.668 μs (1 allocation: 896 bytes)

607.353 μs (1 allocation: 816 bytes)

**For i =1:10**

30.421 μs (0 allocations: 0 bytes)

6.648 μs (1 allocation: 896 bytes)

6.609 μs (1 allocation: 816 bytes)

**TP7: compare stack-fetch vs heap-fetch via vectors and svectors and immutable structs**

|  |  |
| --- | --- |
| struct DataSvector  x::SVector{5,Float64}  y::Int  end | struct DataNormal  x::Vector{Float64}  y::Int  end |

Instantiate the structs and access their vector fields showed the same results from TP2.

28.311 ns (1 allocation: 80 bytes)

1.648 ns (0 allocations: 0 bytes)

If you were trying to construct (or `convert` to) a `StaticArray` you

may need to add the size explicitly as a type parameter so its size is

inferrable to the Julia compiler (or performance would be terrible). For

example, you might try

m = zeros(3,3)

SMatrix(m) # this error

SMatrix{3,3}(m) # correct - size is inferrable

SArray{Tuple{3,3}}(m) # correct, note Tuple{3,3}

**3. Passed-size creation**

struct Data

x::SVector{N,Float64} where {N}

y::Int

end

n=2

v1 = **@SVector** **zeros**(n)

p = **Data**(**v1**, 5) #with zeros(n) gives error

**display**(p.x);

2-element SVector{2, Float64} with indices SOneTo(2):

0.0

0.0

**II.SMatrix**

m1 = **SMatrix**{2,2}(1, 2, 3, 4)