Slice me nice! — `jsc::chunk_evenly` — Range adaptor for distributing work across tasks

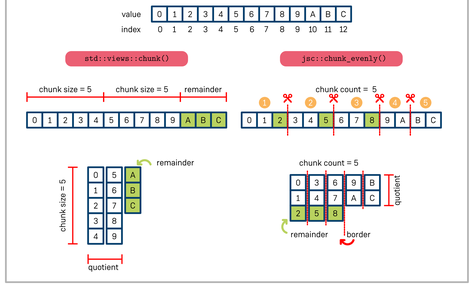
Mateusz Zych, Ivo Kabadshow
Jülich Supercomputing Centre, Research Centre Jülich, Germany

JÜLICH Forschungszentrum

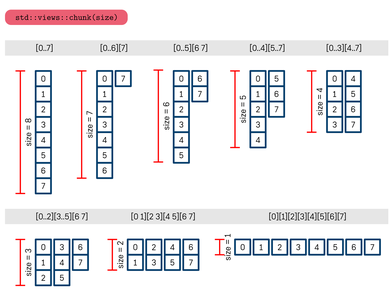
⓪ How to chunk a sequence?

- Different range adaptors dividing a range into subranges available:
  `std::views::chunk(chunk_size)`
  `std::views::chunk_by(predicate)`
  `std::views::split(delimiter)`

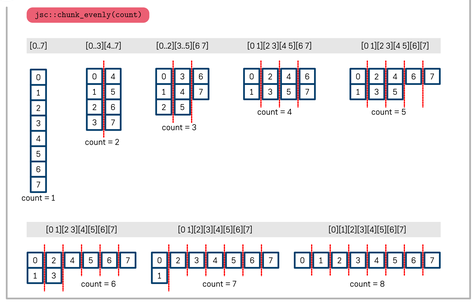
none is suitable to distribute work across tasks



① Chunk by size



② Chunk by count



③ How to chunk & chunk_evenly

```cpp
#include <jsc/chunk_evenly.hpp>
#include <fmt/ranges.h>
#include <array>
#include <ranges>

auto main () -> int
{
    auto range = std::array
    { 0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC };

    fmt::println("{::X}", range | std::views::chunk(5));
    fmt::println("{::X}", range | jsc::chunk_evenly(5));
}
```

Output:

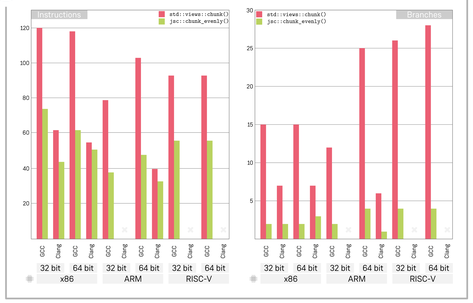
```
[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [A, B, C]]
[[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, A], [B, C]]
```

④ Code gen comparison

```cpp
#include <ranges>
#include <cstddef>
#include <jsc/chunk_evenly.hpp>

auto func (std::ranges::subrange<
           std::ranges::iterator_t<
           std::ranges::iota_view<std::ptrdiff_t,
                                  std::ptrdiff_t>>>) -> void;

auto std_views_chunk jsc_chunk_evenly (std::ptrdiff_t n,
                                       std::ptrdiff_t chunk_size chunk_count ) -> void
{
    for (auto chunk : std::views::iota(std::ptrdiff_t { 0 }, n)
                    | std::views::chunk(chunk_size) jsc::chunk_evenly(chunk_count))
    {
        func(chunk);
    }
}
```

additional header
change to count
change range adaptor

⑤ Number of instructions & branches



⑥ Following zero-overhead principle

`jsc::chunk_evenly()`

```cpp
auto func (std::ranges::subrange<
           std::ranges::iterator_t<
           jsc::iota_view<std::ptrdiff_t,
                          std::ptrdiff_t>>>) -> void;

auto jsc_chunk_evenly (std::ptrdiff_t n,
                       std::ptrdiff_t chunk_count) -> void
{
    for (auto chunk : jsc::iota(std::ptrdiff_t { 0 }, n)
                    | jsc::chunk_evenly(chunk_count))
    {
        func(chunk);
    }
}
```

`manual_loop`

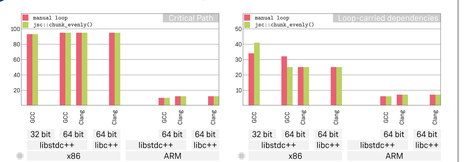
```cpp
struct chunk { std::ptrdiff_t begin; std::ptrdiff_t end; };

auto func (chunk) -> void;

auto manual_loop (std::ptrdiff_t n, std::ptrdiff_t chunk_count) -> void
{
    [[assume(n >= 0)]];
    [[assume(chunk_count > 0)]];

    const auto quotient  = n / chunk_count;
    const auto remainder = n % chunk_count;

    auto chunk_size  = quotient + std::ptrdiff_t { remainder > 0 };
    auto chunk_begin = std::ptrdiff_t { 0 };

    for (auto chunk_index = std::ptrdiff_t { 0 }; chunk_index != chunk_count;)
    {
        func({ chunk_begin, chunk_begin + chunk_size });

        chunk_begin += chunk_size;
        ++chunk_index;
        chunk_size -= std::ptrdiff_t { chunk_index == remainder };
    }
}
```

⑦ ASM comparison

- GCC RISC-V 64-bit assembly

```
manual_loop(long, long):                jsc_chunk_evenly(long, long):
        addi    sp,sp,-64                       addi    sp,sp,-64
        sd      a4,16(sp)                       sd      a4,16(sp)
        rem     a4,a0,a1                        rem     a4,a0,a1
        sd      a2,24(sp)                       sd      a0,48(sp)
        sd      s0,48(sp)                       sd      s1,40(sp)
        sd      s1,40(sp)                       sd      s2,32(sp)
        sd      s3,24(sp)                       sd      s3,24(sp)
        sd      ra,56(sp)                       sd      ra,56(sp)
        mv      s3,a1                           mv      s3,a1
        li      s1,0                            li      s1,1
        li      s0,0                            li      s0,0
        div     a0,a0,a1                        div     a0,a0,a1
        snez    s2,s4                           sgt     a5,a4,zero
        add     s2,s2,a0                        add     s2,a0,a5
.L2:                                    .L2:
        mv      a0,s0                           mv      a0,s0
        add     s0,s0,s2                        add     s0,s0,s2
        mv      a1,s0                           mv      a1,s0
        call    func(chunk)                     call    func()
        addi    s1,s1,1                         sub     a5,a1,a4
        sub     a5,a4,s1                        seqz    a5,a5
        seqz    a5,a5                           mv      a4,a1
        sub     s2,s2,a5                        sub     s2,s2,a5
        bne     s3,s1,.L2                       addi    s1,s1,1
        ld      ra,56(sp)                       bne     s3,a4,.L2
        ld      s0,48(sp)                       ld      ra,56(sp)
        ld      s1,40(sp)                       ld      s0,48(sp)
        ld      s2,32(sp)                       ld      s1,40(sp)
        ld      s3,24(sp)                       ld      s2,32(sp)
        ld      s4,16(sp)                       ld      s3,24(sp)
        addi    sp,sp,64                        ld      a4,16(sp)
        jr      ra                              addi    sp,sp,64
                                                jr      ra
```

⑧ CP and LCD comparison



⑨ Design principles

- Idea inspired by `std.range.evenChunks()` from the D language
- Expensive computations (div, mod) are computed once per iterator
- Branch-less programming for crossing border between smaller and larger chunks
- Zero overhead

std::range.evenChunks()

Ⓐ Improving std::execution example

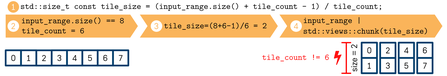
```cpp
stdexec::sender
auto async_inclusive_scan (stdexec::scheduler auto  scheduler,
                           std::span<const double>  input_range,
                           std::span<       double>  output_range,
                           std::ptrdiff_t           chunk_count)
{
    auto partials = std::vector<double>(chunk_count + 1);

    return stdexec::just(std::move(partials))
         | stdexec::continue_on(scheduler)
         | stdexec::bulk(chunk_count, [=] (std::ptrdiff_t       chunk_index,
                                           std::vector<double>& partials)
           {
             auto  input_chunks =  input_range | jsc::chunk_evenly(chunk_count);
             auto output_chunks = output_range | jsc::chunk_evenly(chunk_count);

             auto  input_chunk  =  input_chunks[chunk_index];
             auto output_chunk  = output_chunks[chunk_index];

             std::inclusive_scan(input_chunk.begin(),
                                 input_chunk. end (), output_chunk.begin());

             partials[chunk_index + 1] = output_chunk.back();
           })
         | stdexec::then([] (std::vector<double>& partials)
           {
             std::inclusive_scan(partials.begin(),
                                 partials. end (), partials.begin());

             return std::move(partials);
           })
         | stdexec::bulk(chunk_count, [=] (std::ptrdiff_t       chunk_index,
                                           std::vector<double>& partials)
           {
             auto output_chunks = output_range | jsc::chunk_evenly(chunk_count);

             for (auto& element : output_chunks[chunk_index])
             {
               element += partials[chunk_index];
             }
           })
         | stdexec::then([=] (std::vector<double>& partials)
           {
             return output_range;
           });
}
```

P2300 std::execution

async_inclusive_scan()

Ⓑ Discussion

- Original version chunks manually the input range using the same schema as `std::views::chunk()`, therefore will not always process tile_count tiles



- Also, fork & join model introduces unnecessary waiting
- If tasks are scheduled as early as possible, then distributing work *evenly* across tasks can improve performance

Ⓒ Future Directions

- Support random access in `jsc::chunk_evenly_view<>`
- Find more generic components in the JSC C++ library