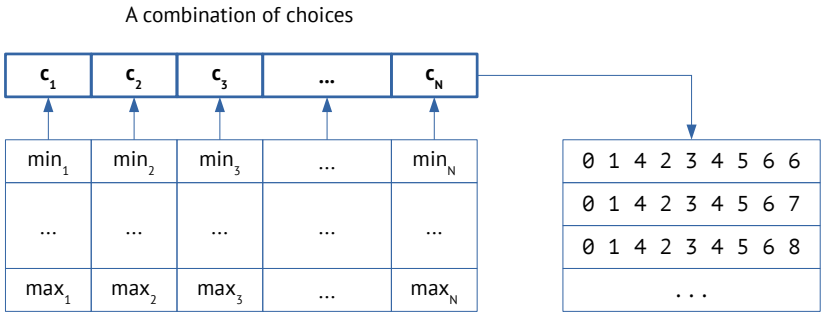
HPC Multi-iterator Engine Design & Implementation

Dmitry Mikushin Simon Scheidegger
Philipp Eisenhauer Moritz Mendel

May 4, 2021

UNIL | Université de Lausanne
HEC Lausanne

Multi-Iterator: the purpose



Each choice has its own constraints (runtime consts)

A client program receives combinations one-by-one without storing anything

Multi-Iterator: motivation

Three horrible ways to implement a multi-iterator:

- Store a matrix of all combinations in RAM (slow, only small problems could fit in)
- Generate combinations recursively (non-parallel)
- Implement bounds check with `if..else` (pipeline stalls)

Three optimization ideas for high-performance multi-iterator:

- On-the-fly generation (and use) of combinations (no memory penalty)
- Use JIT-compliation (trending in modern HPC, see ClangJIT talk by Hal Finkel)
- Use predicates (bitmasks) instead of branching

Simple example

```cpp
#include <multiit/multiit.h>

int main(int argc, char* argv[])
{
    multiit::runtime::MultiIterator mi({ 2, 3, 4 });
    // OR: multiit::compiletime::MultiIterator<2, 3, 4> mi;

    int niters = 0;
    const int size = mi.getSize();
    while (1)
    {
        niters++;
        if (!mi.next()) break;

        // TODO Use the current combination of choices in a target app.
        const auto& current = mi.getCurrent();
        for (int i = 0; i < size; i++)
            printf("%d ", current[i]);
        printf("\n");
    }

    printf("%d iterations visited\n", niters);

    return 0;
}
```

```
./multiit_example
1 0 0
0 1 0
1 1 0
0 2 0
1 2 0
0 0 1
1 0 1
0 1 1
1 1 1
0 2 1
1 2 1
0 0 2
1 0 2
0 1 2
1 1 2
0 2 2
1 2 2
0 0 3
1 0 3
0 1 3
1 1 3
0 2 3
1 2 3
24 iterations visited
```

Supported types of multi-iterators

- `MultiIterator`: A group of indexes that iterate from 0 to the given upper value
- `LimitedMultiIterator`: A group of indexes that iterates only through indexes with total sum no greater than limit
- `GenericMultiIterator`: A hosting group of indexes, whose indexes are themselves groups of indexes

$\Rightarrow$ A `GenericMultiIterator` of `MultiIterator` and `LimitedMultiIterator` elements can be used to express many kinds of combinatorial iterations

Programming package structure

- `multiit`: general-purpose multi-iterators in C++, with unit tests
- `kernit`: multi-iterator *kernel* generator based on JIT-compilation located in the cloud (web service)
- `respyit` : subjected multi-iterators, with Python API, which quiries `kernit` to provide a kernel with required specifics

Cloud serivce + end-user application pipeline

TODO Pipeline figure

End-user example

```
import respyit

# TODO
```