# Minkowski Polyhedral Model[1]

*Javier de Juan*[2]

**Abstract**

*Polyhedral loop transformatin is being around since decades. However, the initial model proposed by Feautrier remains unchanged since its creation. This paper restat the affine model utilizing modern geometry and advanced algebra topics (tensors). Secondly, one of the major deficiencies of the initial model is the absence of a space transformation where the variables live. In this paper we propose a space-time container based on Minkowski geometry which help us to unify both transformation in time and in space.*

## 1 The Polyhedral Model

State-of-the-art literature covering Polyhedral Model is often difficult to read, however the underlying idea is pretty straightforward: Statements enclosed in for loop sequences are scanned and execution dates are filtered out . The Polyhedral transformation then builds affine expressions over this execution dates in order to obtain an equivalent code optimizing parallel execution.

## 2 The Polyhedral Affine Transformation

Mathematically speaking a polyhedral scheduler is a vector-valued function from $\mathbf{N}^n$ to $\mathbf{N}^m$ which maps time points living in a convex hull from input space $S$ to output space $S'$. Roughly speaking a a polyhedral scheduler performs an affine transformation over the input scheduling vectors. For simplicity and a better understandying of the polyhedral transformation, we are going to work with the given C Kernel:

```
for(i=0;i<=N;i++)
  A[i]=0;              // X statement
for(j=0;j<=N;j++)
  for(k=0,k<=N;k++)
    B[j] += A[k];      // Y Statement
```

In a polyhedral transformation, we do not care of the computing statements, so the above Kernel can be rewritten as:

```
for(i=0;i<=N;i++)
  X(i);              // X statement
for(j=0;j<=N;j++)
  for(k=0,k<=N;k++)
    Y(i,j);     // Y Statement
```

Let's introduce the concept of scheduling vectors. A scheduling vector captures information regarding when a tuple $(i, j, k)$ is executed. Hence whe can reinterpret our X and Y statements like this:
`X(i,0,0)-> X` is executed at date `[i.days,0,0]` and
`Y(N,j,k)-> Y` is executed at date `[N.days, j.minutes, k.seconds]`

So, what really *is* a Polyhedral Transformation? Well, unfortunatly Polyhedral Theory is difficult to understand, but the underlying idea is quit straight forward. How can we rearrange X and Y execution dates to have a new Kernel which obtain the same result as source Kernel but takes less time to obtain it? This is done finding new X' and Y' such as

$$X' = (t^0_{xi}.i + t^0_{xj}.j + t^0_{xk}.k + t^0_{xN}.N + t^0_{x1}.1, t^1_{xi}.i + t^1_{xj}.j + t^1_{xk}.k + t^1_{xN}.N + t^1_{x1}.1)$$

and

$$Y' = (t^0_{yi}.i + t^0_{yj}.j + t^0_{yk}.k + t^0_{yN}.N + t^0_{y1}.1, t^0_{yi}.i + t^0_{yj}.j + t^0_{yk}.k + t^0_{yN}.N + t^0_{y1}.1)$$

where $t^d_{sr}$ coefficients are optimaly calculated to ensure that we get a code that yields the same result and execute faster relying on the possibilities

of parallelization of the hardware. In order to solve our problem, we need to feed our polyhedral model with a set of constraints that ensures legal excution of code: we define:

## References

[1] Link Creator(s). Link title. https://example.com/.

[2] FirstName LastName. mailto:email@example.com.

[3] Firstname1 Lastname1 and Firstname2 Lastname2. Article title. *Journal name*, Year. https://dx.doi.org/...