# Domain Specific Language for high performance computing

Ryuki Hiwada

February 10, 2024

**Abstract**

## 1 Introduction

### 1.1 Background

Particle simulations have been executed in various fields such as astrophysics and fluid dynamics. For example, N-body simulations simulate the dynamical evolution due to gravity between planets, and models of fluid motion models simulate water. Particle simulations are computationally demanding with respect to the number of particles. As an example, we will discuss the case of an N-body simulation. In this simulation, most of the computational time is occupied by the calculation of the gravitational interaction. equation (2.2.1) is the equation for the gravitational interaction.

$$a_i = \sum_{j \neq i}^{N} G m_j \frac{\mathbf{x}_j - \mathbf{x}_i}{\left(|\mathbf{x}_j - \mathbf{x}_i| + \epsilon^2\right)^{3/2}} \tag{1}$$

$a_i$, $\mathbf{x}_i$ $m_j$,$\mathbf{x}_j$,$N$,$G$,$\epsilon$ are the acceleration and position of particle i, the mass of particle j, the number of position particles, respectively, and softening parameters to prevent gravitational constant divergence. From equation (2.2.1),time complexity is proportional to the number of particles N,

N-body simulations often require large numbers of particles. For example, a globular cluster contains about $10^4$ $10^5$ stars. Therefore, very long computation time is required. parallel computing is generally used to speed up the process.

The computation of interparticle interactions has high data parallelism because the computation of particles is independent of each other. Therefore, Parallelism by SIMD(Single Instruction/Multiple Data ), one of Flynn's taxonomy [1] , is suitable. SIMD means processing multiple data for a single instruction. However, to perform SIMD, code must be written for the computer architecture.

The goal of this study is to allow users to run particle interaction calculations in parallel without having to be aware of parallelization. We developed a Domain Specific Language(DSL )called Pyker. DSL is a programming language for solving specific problems. Pyker generates parallelized code with SIMD instructions for the calculation of interactions from the definition of variables and the description of formulas. The evaluation method for Pyker is to compare the performance of the automatic SIMD performed by the C compiler with that of the SIMD by Pyker. For this purpose, we measured the execution time of the code that does not adapt the SIMD instructions generated by Pyker for each of the three interaction calculations. The execution time of the code that does not adapt SIMD instructions generated from Pyker and the code that does adapt SIMD instructions are measured and compared.

In this paper, section 2 explains how Pyker defines the interparticle interaction equation, section 3 describes the process of code generation by Pyker, and section 4 presents the results of the performance evaluation experiments.

# 2 Definition of the interparticle interaction in Pyker

## 2.1 interparticle interaction

we explain the needed for information to the calculate interparticle interaction with reference to the actual code. gravity.cpp is the C++ source code that performs the calculation in equation (2.2.1).

Listing 1: skeleton code

```
1
2   for (int i = 0; i < n; i++) {
3     double ax = 0.0;
4     double ay = 0.0;
5     double az = 0.0;
6
7     for (int j = 0; j < n; j++) {
8         double dx = double(xj[j][0] - xi[i][0]);
9         double dy = double(xj[j][1] - xi[i][1]);
10        double dz = double(xj[j][2] - xi[i][2]);
11        double r2 = dx * dx + dy * dy + dz * dz + eps2;
12        double ri2 = 1.0f / sqrt(r2);
13
14        double mr = mj[j] * ri2 * ri2 * ri2;
15
16        ax += double(mr * dx);
17        ay += double(mr * dy);
18        az += double(mr * dz);
19    }
20
21    acci[i][0] = ax;
22    acci[i][1] = ay;
23    acci[i][2] = az;
24 }
```

We will explain how this code corresponds to the expression (2.2.1). acci,xi,mj,xj,eps2 correspond to $a_i$, $\mathbf{x}_i$ $m_j$,$\mathbf{x}_j$,$N$,$\epsilon$ respectively. The $G$ is not in the code because it can be calculated after the calculation of the interparticle interactions.

From this code, we can see that the variables to store the results, the variables for particle i and particle j respectively, and the variables for $N$ and $G$ can be calculated after the calculation of the particle-particle interaction. The other variables have different processing and data structures.

Therefore, there are three categories of information in a variable.

- Whether the variable is a particle i, a particle j, a variable that stores the result, or any other variable
- The type information of the variable.
- Whether the variable is a vector or not

With the definition of this variable and a description of the mathematical equation using the variable, a code to calculate the interparticle interaction can be generated. We explain the language specification in the next section.

## 2.2 language specification

There are two parts in Pyker. variable definition part and interaction description part. First, we explain about the variable definition part.

### 2.2.1 variable definition part

Variable definitions are shown below in regular expressions.

$$"(EPI|EPJ|FORCE)?\ \ (vec3<)?(F64)>?\ \ \backslash w+"$$

EPI and EPJ represent the variables for particles i and j, respectively, and FORCE indicates that they are variables that store the results. Other variables are not written.

Vector are denoted by vec3.If variable is a vector, the variable type is defined in '¡¿'.

Variable type is 'F64' for double-precision.In Pyker, Variable type is only 'F64'.

Finally, the name of the variable is written.

Next, the interaction description is explained.

In Pyker, you can use four arithmetic operations and sqrt(), power multiplication, vector inner product, assignment, and '+=' for formulas. The operations that can be used to describe interactions are shown in the following figure.

```
1 +, -, /, *, **, sqrt(), =, +=
```

- The $+, -, *, /$ denote addition, subtraction, multiplication, and division, respectively.
- $**$ denotes a power

- sqrt() denotes a square root.
- When both terms of $*$ are vectors and the base of $**$ is a vector and the exponent is 2, respectively, the calculation is determined to be an inner product.
- The $+=$ indicates that the result of the calculation is added to the FORCE variable.
- $=$ is Assignment

In principle, if the elements are vectors, the operation is performed on each element of the vector.

The objects that can be used for the description are variables that have been defined and temporary variables and floats defined prior to the description. The following is an example of a Pyker description of the calculation of the table

Listing 2: gravity_interparticle

```
EPI vec3<F64> ri
EPJ vec3<F64> rj
EPJ F64 mass
EPJ F64 eps2
FORCE vec3<F64> ai
rij = ri - rj
r2 = rij * rij + eps2
r_inv = 1.0 / sqrt(r2)
r2_inv = r_inv * r_inv
mr_inv = mass * r_inv
mr3_inv = r2_inv * mr_inv
ai += mr3_inv * rij
```

次に、これらの記述から SIMD 命令を行えるコードを生成する説明をする。

# 3 Parser of Pyker

Pyker では、このセクションでは、まずどのようにして SIMD による並列化を行うか説明し、その後 Pyker の記述の読み込みからコード生成までの処理内容と流れを説明します。

### 3.0.1 AVX2 for parallelization

### 3.0.2 Flow of Parse

# 4 Experiments

## 4.1 gravity interparticle

Listing 3: Nbody-kernel.pyker

```
EPI vec3<F64> ri
EPJ vec3<F64> rj
```

```
  EPJ F64 mass
  EPJ F64 eps2
  FORCE vec3<F64> ai
  rij = ri - rj
  r2 = rij * rij + eps2
  r_inv = 1.0 / sqrt(r2)
  r2_inv = r_inv * r_inv
  mr_inv = mass * r_inv
  mr3_inv = r2_inv * mr_inv
  ai = mr3_inv * rij
```

Listing 4: Nbody-kernel.pikg

```
  EPI F64vec ri:r
  EPJ F64vec rj:r
  EPJ F64 mj:m
  EPJ F64 eps2:eps
  FORCE F64vec ai:acc
  rij = ri - rj
  r2 = rij * rij + eps2
  r_inv = rsqrt(r2)
  r2_inv = r_inv * r_inv
  mr_inv = mj * r_inv
  mr3_inv = r2_inv * mr_inv
  ai += mr3_inv * rij
```

non-simd

g++ -O3

| N | Pyker | PIKG |
|---|---|---|
| 25000 | 6.725156 | 4.444766 |
| 10000 | 0.738918sec | 0.517324sec |

Listing 5: Nbody-kernel.pikg

```
  g++ -O3 -I ~/school/PIKG/PIKG/inc -mavx2 -mfma gravity_interparticle.cpp
```

| N | Pyker(sec) | PIKG(sec) |
|---|---|---|
| 50000 | 5.119071 | 3.333270 |
| 25000 | 1.218129 | 1.148511 |
| 10000 | 0.143777 | 0.101594 |
| 1000 | 0.003521 | 0.002168 |

## 4.2 LennardJones

Listing 6: LennardJones-kernel.pyker

```
EPI F64 rix
EPI F64 riy
EPI F64 riz
EPJ F64 rjx
EPJ F64 rjy
EPJ F64 rjz
```

```
EPJ F64 eps

FORCE F64 fx
FORCE F64 fy
FORCE F64 fz
FORCE F64 p
dx = rix - rjx
dy = riy - rjy
dz = riz - rjz
r2 = dx * dx + dy * dy + dz * dz + eps
r2i = 1.0 / r2
r6i = r2i * r2i * r2i
f = (48.0* r6i - 24.0) * r6i * r2i
fx = f * dx
fy = f * dy
fz = f * dz
p = 4.0 * (r6i) * (r6i - 1.0)
```

Listing 7: LennardJones-kernel.pyker

```
  EPI F64 rix:rx
  EPI F64 riy:ry
  EPI F64 riz:rz

  EPJ F64 rjx:rx
  EPJ F64 rjy:ry
  EPJ F64 rjz:rz
  EPJ F64 eps2:eps

  FORCE F64 fx:fx
  FORCE F64 fy:fy
  FORCE F64 fz:fz
  FORCE F64 p:p

  dx = rix - rjx
  dy = riy - rjy
  dz = riz - rjz
  r2 = dx * dx + dy * dy + dz * dz + eps2
  r2i = 1.0 / r2
  r6i = r2i * r2i * r2i
  f = (48.0 * r6i - 24.0) * r6i * r2i
  fx += f * dx
  fy += f * dy
  fz += f * dz
  p += 4.0 * r6i*(r6i - 1.0)
```

| | N | Pyker | PIKG |
|---|---|---|---|
| non-simd | 50000 | 40.736862 | 11.656278 |
| | 25000 | 10.662050 | 3.521253 |
| | 10000 | 0.985475 | 0.442775 |

| | N | Pyker | PIKG |
|---|---|---|---|
| | 50000 | 3.702871 | 2.838773 |
| simd | 25000 | 0.838122 | 0.9904373 |
| | 10000 | 0.134388 | 0.215270 |

# 5 Conclusion

# 6 Acknowledgement

# References

# References

[1] M.J. Flynn, "Some computer organizations and their effectiveness," IEEE transactions on computers, vol.100, no.9, pp.948–960, 1972.