Eq Programming Language

Pavel Zaichenkov

University of Hertfordshire

July 26, 2011

Agenda

1. Introduction
   - A language concept
2. EqCode description
   - The structure of a compiler
   - Fibonacci numbers (example)
   - N-body problem (example)
   - Syntax and language constructions
3. Problems and restrictions

A language concept

In computational mathematics or physics all operations can be separated into two types.
Data parallel operation doesn't depend on previous iterations. It deals with independent data. In this way, all computational processes can be run separately.
Recurrent depended operations can't be run separately. Iterations have to be run in a queue, as an every next operation is going to use the result of the previous one.

In mathematics there is a widely used notation, which seems quite easy to understand because of formulas, equations and mathematical designations.

A language concept

Data parallel operation

$$f_0 = f(x_0, x_1, \ \dots \ x_n)$$
$$f_1 = f(x_0, x_1, \ \dots \ x_n)$$
$$\dots$$
$$f_i = f(x_0, x_1, \ \dots \ x_n)$$

where $\forall i \ x_i$ is data

A language concept

Recurrent depended operation

$$f_i = f(f_j, f_{j+1}, \ \dots f_{i-1}, x_1, \ \dots x_n)$$

where $\forall i \; x_i$ is data and $j \leq i - 1$

The structure of a compiler

1. **LaTeX front-end**. It's possible to write a program using LaTeX syntax. This allows to use any existing LaTeX tools (compile to pdf, ps, html, etc...).
2. **EqCode compiler**. We are going to write a compiler which will be able to compile an existing LaTeX code into any chosen back-end language.
3. **Custom back-end (SaC, S-Net, C, ...)**. It's possible to create a code-generator into any language we want to deal with. We are going to support SaC as it has a relevant data parallelism and recurrent dependency support.

Fibonacci numbers

Wikipedia

$$F_0 = 0$$
$$F_1 = 1$$
$$F_i = F_{i-1} + F_{i-2}$$

C / SaC

```
int f(int n)
{
  if ((n == 0) || (n == 1))
    return n;
  return f(n - 1) + f(n - 2);
}
```

N-body problem

Wikipedia

$$m_j\ddot{\mathbf{q}}_j = G\sum_{k\neq j}\frac{m_j m_k(\mathbf{q}_k - \mathbf{q}_j)}{|\mathbf{q}_k - \mathbf{q}_j|^3}, j = 1, \ldots, n$$

N-body problem

C (debian shootout)

```
void advance(int nbodies, struct planet * bodies, double dt)
{
  int i, j;
  for (i = 0; i < nbodies; i++) {
    struct planet * b = &(bodies[i]);
    for (j = i + 1; j < nbodies; j++) {
      struct planet * b2 = &(bodies[j]);
      double dx = b->x - b2->x; double dy = b->y - b2->y; double dz = b↩
          ->z - b2->z;
      double distance = sqrt(dx * dx + dy * dy + dz * dz);
      double mag = dt / (distance * distance * distance);
      b->vx -= dx * b2->mass * mag; b->vy -= dy * b2->mass * mag; b->vz ↩
          -= dz * b2->mass * mag;
      b2->vx += dx * b->mass * mag; b2->vy += dy * b->mass * mag; b2->vz↩
           += dz * b->mass * mag;
    }
  }
  for (i = 0; i < nbodies; i++) {
    struct planet * b = &(bodies[i]);
    b->x += dt * b->vx; b->y += dt * b->vy; b->z += dt * b->vz;
  }
}
```

N-body problem

EqCode

$$advance(p, v, m, dt) : \ \mathbb{R}^2_{5,3}, \mathbb{R}^2_{5,3}, \mathbb{R}^1_5, \mathbb{R} \ \rightarrow \ \mathbb{R}^3$$

$$accs_{i,j} \mid 0 \leq i \leq 4 \wedge 0 \leq j \leq 4 = \begin{cases} \dfrac{(p_j - p_i) \cdot m_j}{\rho(p_i, p_j)^3} & j < i \\ 0 & \text{otherwise} \end{cases}$$

$$accs_{i,j} \mid j > i = -accs_{j,i}$$

$$a_{i,j} = \sum_k accs_{i,k,j}$$

$$v = v + a \cdot dt$$

$$p = p + v \cdot dt$$

$$\text{return}(p, v)$$

Recurrent operations support

EqCode

$$f(n) : \mathbb{Z} \rightarrow \mathbb{Z}$$
$$F^{[0]} = 0$$
$$F^{[1]} = 1$$
$$F^{[i]} = F^{[i-1]} + F^{[i-2]}$$
$$\textbf{return}(\textbf{filter}(F^{[i]} \mid i = n))$$

LaTeX

```
\begin{eqcode}{n}{\type{Z}}{\type{Z}}
  F^{[0]} = 0 \lend
  F^{[1]} = 1 \lend
  F^{[i]} = F^{[i-1]} + F^{[i-2]} \lend
  \return{\filter{F^{[i]}\ |\ i = n}}
\end{eqcode}
```

Parallelized operations support

EqCode

$$a_{i,j} \mid 0 \leq i < 5 \land 2 \leq j < 6 = \begin{cases} 42 & 0 \leq i < 2 \land 0 \leq j < 3 \\ 0 & \text{otherwise} \end{cases}$$

LaTeX

```
a_{i,j}\  |\  0 \leq i < 5 \land 2 \leq j < 6 =
  \begin{cases}
    42 & 0 \leq i < 2 \land 0 \leq j < 3 \lend
    0 \otherwise
  \end{cases}
```

SaC

```
a = with {
    ([0,2] <= [i,j] < [5,6] ) : 42;
  } : genarray([5,6], 0);
```

Problems and restrictions

- **Types and type conversion issues**. We can't use just types that mathematicians are familiar with(natural numbers, whole numbers, etc.). There should be a type hierarchy to understand the representation of these types in the architecture.
- **Syntax restrictions**. The same algorithm can be represented In LaTeX in different ways. However, a source code should be translated unambiguously into the target code. That's why some syntax restrictions are needed.

Project repository

http://github.com/zayac/EqCode/

Contacts

Mail+Jabber: zaichenkov@gmail.com

Questions?